Dr. Arie Gunawan, S.Kom., M.M.S.I.

MOBILE PROGRAMMING

Menggunakan

Flutter & Visual Studio Code

untuk Pemula

Penerbit litnus.

Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula

Ditulis oleh:

Dr. Arie Gunawan, S.Kom., M.M.S.I.

Diterbitkan, dicetak, dan didistribusikan oleh
PT. Literasi Nusantara Abadi Grup
Perumahan Puncak Joyo Agung Residence Kav. B11 Merjosari
Kecamatan Lowokwaru Kota Malang 65144
Telp : +6285887254603, +6285841411519
Email: literasinusantaraofficial@gmail.com
Web: www.penerbitlitnus.co.id
Anggota IKAPI No. 340/JTI/2022

Cetakan I, April 2024

Perancang sampul: Muhammad Ridho Naufal
Penata letak: Bagus Aji Saputra

ISBN : 978-623-114-735-6

xii + 242 hlm. ; 15,5x23 cm.

KATA PENGANTAR

Mobile programming atau pengembangan aplikasi mobile telah menjadi bagian yang tak terpisahkan dalam kehidupan modern. Dalam dunia yang semakin berkembang pesat ini, masyarakat memerlukan aplikasi mobile yang dapat membantu mempermudah pekerjaan, berinteraksi dengan orang lain, serta mendapatkan informasi yang dibutuhkan secara cepat dan mudah.

Flutter dan Visual Studio Code adalah dua alat yang sangat populer dan digunakan oleh para pengembang aplikasi mobile. Flutter adalah sebuah framework open source yang dapat membantu pengembang untuk membuat aplikasi mobile dengan mudah dan cepat, sementara Visual Studio Code adalah salah satu Integrated Development Environment (IDE) yang dapat digunakan untuk mengembangkan aplikasi mobile secara efisien.

Buku ini ditujukan bagi para pemula yang ingin mempelajari pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code. Buku ini berisi panduan yang mudah dipahami dan step-by-step, yang akan membantu para pembaca untuk memahami konsep dasar Flutter dan Visual Studio Code serta mampu membuat aplikasi mobile dengan mudah.

Dalam buku ini, para pembaca akan mempelajari tentang cara instalasi dan konfigurasi Flutter dan Visual Studio Code, penggunaan widget dan state pada Flutter, layout pada Flutter, input pada aplikasi Flutter, navigasi

pada aplikasi Flutter, integrasi dengan API dan plugin, animasi pada Flutter, serta menerapkan desain dan tema pada aplikasi.

Buku ini juga dilengkapi dengan contoh koding yang jelas dan mudah dipahami. Semoga buku ini dapat membantu para pembaca dalam mempelajari pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code.

Jakarta,

Dr. Arie Gunawan, S.Kom., MMSI

PRAKATA

Mobile programming atau pengembangan aplikasi mobile telah menjadi salah satu bidang yang semakin diminati di era digital ini. Kebutuhan akan aplikasi mobile yang efektif dan efisien semakin meningkat, sehingga para pengembang perlu menguasai teknologi yang tepat untuk membuat aplikasi mobile yang baik.

Flutter dan Visual Studio Code adalah dua alat yang sangat populer dan digunakan oleh para pengembang aplikasi mobile. Flutter adalah sebuah framework open source yang memungkinkan pengembang untuk membuat aplikasi mobile dengan cepat dan mudah, sementara Visual Studio Code adalah salah satu Integrated Development Environment (IDE) yang sangat efektif dan populer di kalangan pengembang.

Buku "Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula" ini bertujuan untuk membantu para pemula dalam mempelajari pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code dengan mudah dan efektif. Buku ini dirancang dengan panduan langkah demi langkah dan contoh koding yang mudah dipahami, sehingga para pembaca dapat dengan mudah memahami dan mengikuti konsep dasar dari Flutter dan Visual Studio Code.

Dalam buku ini, para pembaca akan mempelajari tentang instalasi dan konfigurasi Flutter dan Visual Studio Code, dasar-dasar pemrograman pada Flutter, penggunaan widget dan state, layout pada Flutter, input pada aplikasi Flutter, navigasi pada aplikasi Flutter, integrasi dengan API dan

plugin, animasi pada Flutter, serta menerapkan desain dan tema pada aplikasi.

Buku ini diharapkan dapat menjadi panduan yang bermanfaat bagi para pemula yang ingin mempelajari pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code. Kami berharap buku ini dapat memberikan wawasan baru dan membantu para pembaca dalam mengembangkan aplikasi mobile dengan lebih baik.

Akhir kata, kami mengucapkan terima kasih kepada semua pihak yang telah membantu dalam penulisan buku ini. Semoga buku ini bermanfaat dan memberikan kontribusi positif bagi para pembaca.

DAFTAR ISI

BAB 1
PENDAHULUAN

A. Latar Belakang

Mengapa penting belajar Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula.

1. Tingginya permintaan pasar terhadap pengembangan aplikasi mobile
   Dalam era digital saat ini, penggunaan perangkat mobile semakin meningkat dan aplikasi mobile semakin menjadi kebutuhan bagi pengguna. Permintaan pasar terhadap aplikasi mobile yang efektif dan efisien sangat tinggi. Oleh karena itu, keahlian dalam pengembangan aplikasi mobile sangat diperlukan, dan ini menjadikan pentingnya belajar Mobile Programming menggunakan Flutter dan Visual Studio Code.
2. Penggunaan Flutter sebagai alat pengembangan aplikasi mobile yang efektif
   Flutter adalah sebuah framework open source yang memungkinkan pengembang untuk membuat aplikasi mobile dengan cepat dan mudah. Framework ini juga memberikan performa yang tinggi dan hasil yang konsisten pada berbagai platform mobile. Oleh karena itu, belajar menggunakan Flutter akan membantu pemula dalam mengembangkan aplikasi mobile dengan lebih efektif dan efisien.
3. Visual Studio Code sebagai IDE yang efektif dan populer di kalangan pengembang
   Visual Studio Code adalah salah satu Integrated Development Environment (IDE) yang sangat efektif dan populer di kalangan pengembang. IDE ini memiliki fitur-fitur yang sangat lengkap dan memudahkan pengembang dalam membuat aplikasi mobile. Dengan mempelajari Visual Studio Code, pemula dapat menggunakan IDE

ini untuk mengembangkan aplikasi mobile dengan lebih mudah dan cepat.

4. Penggunaan widget dan state pada Flutter
   Flutter memiliki widget sebagai salah satu fitur utamanya. Widget adalah elemen dasar pada Flutter yang dapat membantu pengembang dalam mengatur tampilan dan interaksi pada aplikasi mobile. Pemula perlu mempelajari penggunaan widget dan state pada Flutter agar dapat mengembangkan aplikasi mobile dengan tampilan dan interaksi yang lebih menarik dan efektif.
5. Penerapan desain dan tema pada aplikasi
   Selain penggunaan widget dan state, penerapan desain dan tema pada aplikasi mobile juga sangat penting untuk menarik perhatian pengguna. Belajar bagaimana menerapkan desain dan tema pada aplikasi mobile dapat membantu pemula dalam mengembangkan aplikasi yang lebih menarik dan profesional.

Belajar Mobile Programming menggunakan Flutter dan Visual Studio Code untuk pemula sangat penting karena meningkatkan keahlian dalam pengembangan aplikasi mobile, menggunakan Flutter sebagai alat pengembangan aplikasi mobile yang efektif, menggunakan Visual Studio Code sebagai IDE yang efektif dan populer, mempelajari penggunaan widget dan state pada Flutter, dan penerapan desain dan tema pada aplikasi mobile

B. Tujuan

Setelah membaca buku "Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula", pembaca diharapkan dapat mencapai beberapa hal berikut:

1. Memahami konsep dasar pengembangan aplikasi mobile
   Buku ini akan membantu pembaca memahami konsep dasar pengembangan aplikasi mobile, termasuk arsitektur aplikasi, alur data, dan interaksi antarmuka pengguna.

2. Mampu mengembangkan aplikasi mobile dengan menggunakan Flutter

   Buku ini akan membantu pembaca mempelajari penggunaan Flutter sebagai framework pengembangan aplikasi mobile, termasuk penggunaan widget, state, dan routing pada Flutter. Pembaca akan mempelajari cara membuat aplikasi mobile dari awal hingga siap dipublikasikan.

3. Mampu menggunakan Visual Studio Code sebagai IDE pengembangan

   Buku ini akan membantu pembaca memahami cara menggunakan Visual Studio Code sebagai Integrated Development Environment (IDE) yang efektif untuk pengembangan aplikasi mobile. Pembaca akan mempelajari cara mengkonfigurasi Visual Studio Code untuk pengembangan aplikasi mobile dan bagaimana menggunakan fitur-fitur yang disediakan.

4. Memahami best practice dalam pengembangan aplikasi mobile

   Buku ini akan membantu pembaca memahami best practice dalam pengembangan aplikasi mobile, termasuk penggunaan arsitektur aplikasi yang tepat, penggunaan konvensi penamaan yang baik, dan penggunaan metode pengujian yang tepat. Hal ini akan membantu pembaca mengembangkan aplikasi mobile yang baik dan mudah di-maintain.

5. Siap untuk memasuki industri pengembangan aplikasi mobile

   Buku ini akan membantu pembaca mempersiapkan diri untuk memasuki industri pengembangan aplikasi mobile. Dengan memahami konsep dasar dan best practice dalam pengembangan aplikasi mobile serta menggunakan Flutter dan Visual Studio Code sebagai alat pengembangan, pembaca akan siap untuk mengembangkan aplikasi mobile yang profesional dan sesuai dengan kebutuhan pasar.

Dengan mencapai hal-hal di atas, pembaca diharapkan dapat menjadi seorang pengembang aplikasi mobile yang handal dan siap untuk memasuki pasar pengembangan aplikasi mobile yang sedang berkembang pesat.

C. Sasaran Pembaca

Buku "Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula" ditujukan untuk pembaca yang memiliki minat dalam pengembangan aplikasi mobile namun belum memiliki pengalaman dalam pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code. Target pembaca buku ini adalah pemula atau orang-orang yang baru memulai belajar pengembangan aplikasi mobile dan ingin belajar cara membuat aplikasi mobile menggunakan Flutter dan Visual Studio Code dengan mudah dan cepat.

Buku ini juga cocok untuk mahasiswa yang mengambil jurusan teknologi informasi, komputer, atau jurusan terkait lainnya yang mempelajari pengembangan aplikasi mobile. Selain itu, buku ini juga dapat membantu programmer yang ingin memperluas pengetahuan dan keterampilan mereka dalam pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code.

D. Manfaat

Setelah membaca buku "Mobile Programming Menggunakan Flutter dan Visual Studio Code Untuk Pemula", pembaca dapat memperoleh beberapa manfaat berikut:

1. Peningkatan kemampuan dalam pengembangan aplikasi mobile
   Buku ini akan membantu pembaca mempelajari konsep-konsep dasar pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code. Pembaca akan belajar cara membuat aplikasi mobile yang fungsional dan menarik dengan menggunakan widget, state, dan routing pada Flutter. Selain itu, pembaca juga akan mempelajari cara menggunakan fitur-fitur pada Visual Studio Code yang membantu pengembangan aplikasi mobile.
2. Meningkatkan kepercayaan diri dalam pengembangan aplikasi mobile
   Buku ini dirancang untuk membantu pembaca membangun kepercayaan diri mereka dalam pengembangan aplikasi mobile.

Dengan pembahasan yang jelas dan mudah dipahami serta dilengkapi dengan contoh kode, pembaca akan merasa lebih percaya diri dalam membuat aplikasi mobile dengan menggunakan Flutter dan Visual Studio Code.

3. Memahami best practice dalam pengembangan aplikasi mobile
   Buku ini juga membahas best practice dalam pengembangan aplikasi mobile. Dengan memahami best practice, pembaca dapat mengembangkan aplikasi mobile yang baik dan mudah di-maintain. Hal ini akan membantu pembaca untuk menghasilkan aplikasi mobile yang memenuhi kebutuhan pengguna dengan efektif.
4. Meningkatkan peluang karir sebagai pengembang aplikasi mobile
   Dengan memperoleh kemampuan dan pengetahuan dalam pengembangan aplikasi mobile, pembaca dapat meningkatkan peluang karir mereka sebagai pengembang aplikasi mobile. Saat ini, industri aplikasi mobile sedang berkembang pesat dan permintaan akan pengembang aplikasi mobile terus meningkat. Dengan membaca buku ini dan mempelajari pengembangan aplikasi mobile menggunakan Flutter dan Visual Studio Code, pembaca dapat mempersiapkan diri untuk memasuki pasar kerja yang menjanjikan ini.
5. Memiliki dasar yang kuat untuk belajar pengembangan aplikasi mobile selanjutnya
   Buku ini membahas konsep-konsep dasar dalam pengembangan aplikasi mobile dan mengajarkan penggunaan Flutter dan Visual Studio Code untuk membuat aplikasi mobile. Dengan memahami konsep-konsep dasar ini, pembaca akan memiliki dasar yang kuat untuk belajar teknologi baru dalam pengembangan aplikasi mobile di masa depan.

Dengan manfaat-manfaat di atas, pembaca akan memiliki kemampuan yang lebih baik dalam mengembangkan aplikasi mobile dengan menggunakan Flutter dan Visual Studio Code, serta memiliki dasar yang kuat untuk mempelajari teknologi baru dalam pengembangan aplikasi mobile.

BAB 2
PENGENALAN FLUTTER

A. Tujuan Pembelajaran

1. Memahami konsep dasar tentang Flutter sebagai kerangka kerja pengembangan aplikasi mobile.
2. Mengenal peran dan fitur-fitur utama dalam Flutter yang membedakannya dari kerangka kerja lain.

B. Apa Itu Flutter?

Flutter adalah sebuah framework open-source yang dikembangkan oleh Google untuk membangun aplikasi mobile, web, dan desktop dengan menggunakan bahasa pemrograman Dart. Flutter menggunakan konsep widget untuk membangun antarmuka pengguna dan memberikan pengalaman pengguna yang konsisten di berbagai platform.

Flutter memungkinkan pengembang untuk membuat aplikasi mobile dengan cepat dan efisien karena fitur-fiturnya yang lengkap, seperti hot reload yang memungkinkan perubahan kode langsung terlihat pada aplikasi yang sedang berjalan, serta banyaknya widget dan plugin yang tersedia untuk mempermudah pembuatan aplikasi. Selain itu, Flutter juga menawarkan kemampuan untuk membangun aplikasi dengan tampilan yang menarik dan responsif, serta performa yang tinggi.

Flutter dapat digunakan untuk mengembangkan aplikasi mobile untuk Android, iOS, dan juga web dengan menggunakan Flutter for Web. Flutter juga dapat digunakan untuk mengembangkan aplikasi desktop untuk Windows, macOS, dan Linux dengan menggunakan Flutter for Desktop.

C. Fitur-Fitur Flutter

Flutter memiliki banyak fitur yang memudahkan pengembangan aplikasi mobile. Berikut adalah beberapa fitur utama yang terdapat di Flutter:

1. Widget: Widget merupakan salah satu fitur utama di Flutter. Widget adalah komponen dasar yang digunakan untuk membangun tampilan aplikasi Flutter. Flutter menyediakan berbagai jenis widget, seperti widget layout, widget tampilan, widget input, widget animasi, dan masih banyak lagi. Widget dapat digunakan secara terpisah atau dalam hierarki, sehingga pengembang dapat dengan mudah membuat tampilan aplikasi yang kompleks.
2. Hot Reload: Hot reload adalah fitur yang memungkinkan pengembang untuk mengedit kode sumber aplikasi dan melihat hasil perubahannya secara langsung di emulator atau perangkat fisik tanpa harus melakukan rebuild aplikasi. Hal ini memungkinkan pengembang untuk melihat efek dari perubahan yang dilakukan dengan cepat dan membantu mempercepat proses pengembangan.
3. Animasi: Flutter menyediakan dukungan untuk animasi yang halus dan menarik. Animasi dapat digunakan untuk memberikan pengalaman pengguna yang lebih baik, seperti animasi saat memuat data, animasi transisi antara tampilan, atau animasi interaktif lainnya. Flutter menyediakan widget animasi dan juga memiliki library animasi yang kaya.
4. State Management: State management adalah teknik untuk mengatur dan mengelola data dan keadaan aplikasi. Flutter menyediakan berbagai macam pendekatan untuk state management, seperti setState(), InheritedWidget, Provider, dan lainnya. Pendekatan yang digunakan tergantung pada kompleksitas aplikasi dan preferensi pengembang.
5. Dart: Flutter menggunakan bahasa pemrograman Dart, yang dikembangkan oleh Google. Dart adalah bahasa pemrograman modern dan efisien yang mudah dipelajari dan memungkinkan

pengembang untuk menulis kode aplikasi yang lebih cepat dan lebih mudah dipelihara.

6. Pengembangan Cross-Platform: Flutter memungkinkan pengembangan aplikasi mobile untuk Android dan iOS dalam satu kode sumber yang sama. Hal ini memungkinkan pengembang untuk menghemat waktu dan biaya dalam proses pengembangan.
7. Komunitas yang Aktif: Flutter memiliki komunitas yang aktif dan berkembang pesat. Hal ini memungkinkan pengembang untuk dengan mudah menemukan solusi untuk masalah teknis yang muncul, serta dapat berkolaborasi dengan pengembang Flutter lainnya untuk mengembangkan aplikasi yang lebih baik.
8. Material Design dan Cupertino: Flutter menyediakan dukungan penuh untuk Material Design dan Cupertino, desain tampilan yang digunakan oleh Android dan iOS. Dengan dukungan ini, pengembang dapat dengan mudah membuat aplikasi yang konsisten dengan platform yang ditujukan.

Dalam buku ini, pembaca akan belajar bagaimana menggunakan Flutter dan Visual Studio Code untuk membangun aplikasi mobile yang responsif dan menarik. Buku ini cocok untuk pembaca yang ingin mempelajari Flutter dari awal, termasuk pembaca yang belum memiliki pengalaman dalam pengembangan aplikasi mobile.

D. Keuntungan Menggunakan Flutter

Flutter memiliki beberapa keuntungan dibandingkan dengan teknologi pengembangan aplikasi mobile lainnya, di antaranya adalah:

1. Performa Tinggi: Flutter menggunakan mesin rendering yang dikembangkan sendiri oleh Google, yang memungkinkan aplikasi yang dibuat dengan Flutter untuk memiliki performa yang sangat baik. Selain itu, karena menggunakan bahasa pemrograman Dart, aplikasi yang dibuat dengan Flutter juga memiliki ukuran file yang lebih kecil dan waktu loading yang lebih cepat.

2. Widget yang Rich dan Kustomisasi Tinggi: Flutter menyediakan widget yang dapat disesuaikan dan dapat dikustomisasi dengan mudah untuk membangun tampilan aplikasi yang menarik. Widget-widget ini dapat digunakan untuk membuat tampilan aplikasi yang kaya dengan animasi dan efek visual yang menarik.
3. Mudah Dipelajari: Flutter menggunakan bahasa pemrograman Dart yang mudah dipelajari dan memiliki sintaks yang mirip dengan bahasa pemrograman Java atau JavaScript. Selain itu, dokumentasi Flutter yang lengkap dan komprehensif juga memudahkan pengembang untuk mempelajari Flutter dari awal.
4. Mudah di Maintain: Dengan menggunakan widget yang dapat dikustomisasi dan reusable, aplikasi yang dibuat dengan Flutter lebih mudah di maintain dan di-update, karena perubahan pada widget tertentu dapat diterapkan pada seluruh aplikasi dengan mudah.

Dalam pengembangan aplikasi mobile, memilih teknologi yang tepat sangatlah penting. Dengan menggunakan Flutter, pengembang dapat memanfaatkan keuntungan-keuntungan di atas untuk mempercepat proses pengembangan dan menghasilkan aplikasi mobile yang responsif, menarik, dan mudah di maintain.

E. Cara kerja Flutter

Flutter adalah sebuah framework open-source untuk membuat aplikasi mobile yang dikembangkan oleh Google. Flutter menggunakan bahasa pemrograman Dart yang dirancang untuk membuat aplikasi yang responsif, cepat, dan indah.

Flutter memiliki cara kerja yang berbeda dengan framework mobile lainnya. Saat sebuah widget diubah pada Flutter, Flutter akan membangun ulang seluruh widget tree (pohon widget) dan memperbarui hanya bagian yang berubah. Hal ini memungkinkan Flutter untuk mencapai performa yang lebih cepat dan efisien dibandingkan dengan framework mobile lainnya. Selain itu, karena menggunakan bahasa pemrograman Dart, Flutter memiliki fitur Just-In-Time (JIT) compilation, yang memungkinkan

pengembang untuk melihat perubahan yang dilakukan pada aplikasi secara real-time.

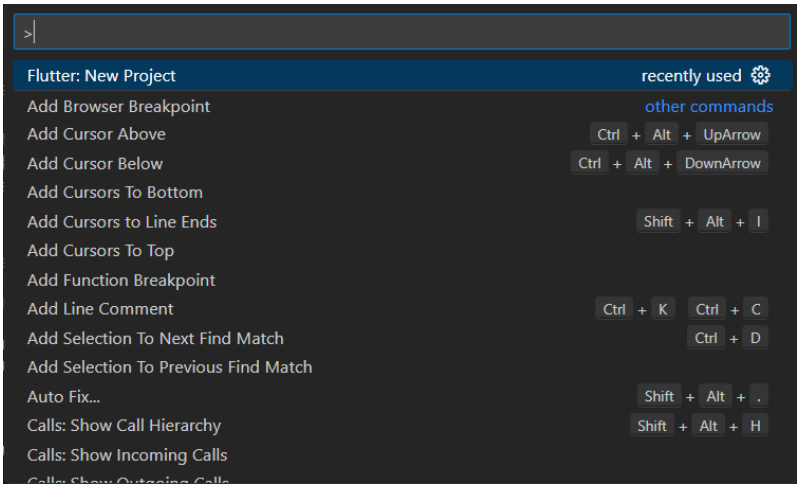
Flutter juga memiliki fitur Hot Reload, yang memungkinkan pengembang untuk mengubah kode sumber aplikasi dan melihat hasil perubahannya secara langsung tanpa harus melakukan proses rebuild aplikasi dari awal. Fitur ini memungkinkan pengembangan aplikasi menjadi lebih cepat dan efisien.

Selain itu, Flutter juga menyediakan widget kustom yang dapat disesuaikan dengan kebutuhan pengembang. Flutter juga menyediakan dukungan untuk platform mobile yang berbeda, seperti Android dan iOS, sehingga pengembang dapat membuat aplikasi yang berjalan di platform mobile yang berbeda hanya dengan menggunakan satu set kode sumber.

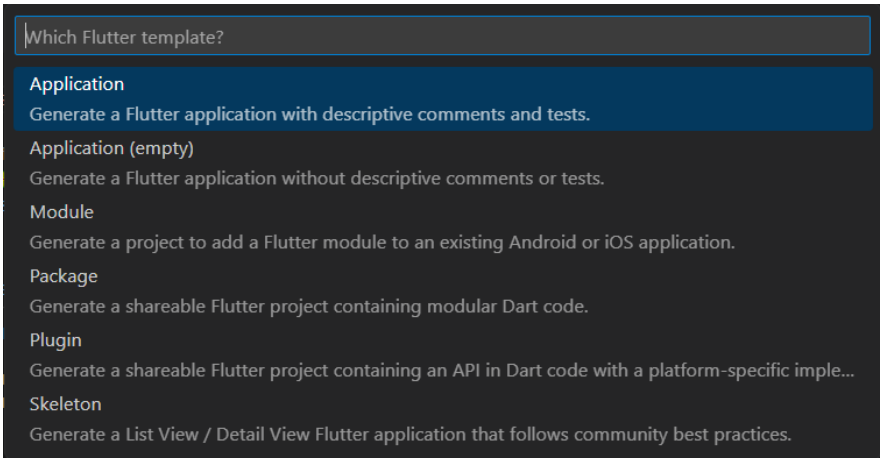
Secara keseluruhan, cara kerja Flutter memungkinkan pengembangan aplikasi mobile yang lebih cepat, responsif, dan efisien dengan menggunakan bahasa pemrograman yang modern dan fitur-fitur yang dapat mempercepat proses pengembangan.

F. Contoh Koding: Membuat "Hello World" Dengan Flutter

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

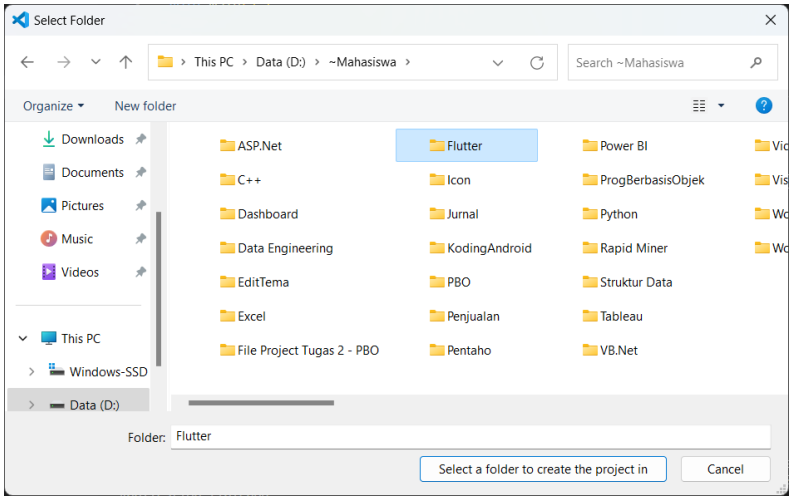


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **hello_world** kemudian tekan **Enter**.

```
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(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Hello World'),
        ),
        body: const Center(
          child: Text(
            'Hello, World!',
            style: TextStyle(
              fontSize: 24,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    );
  }
}
```

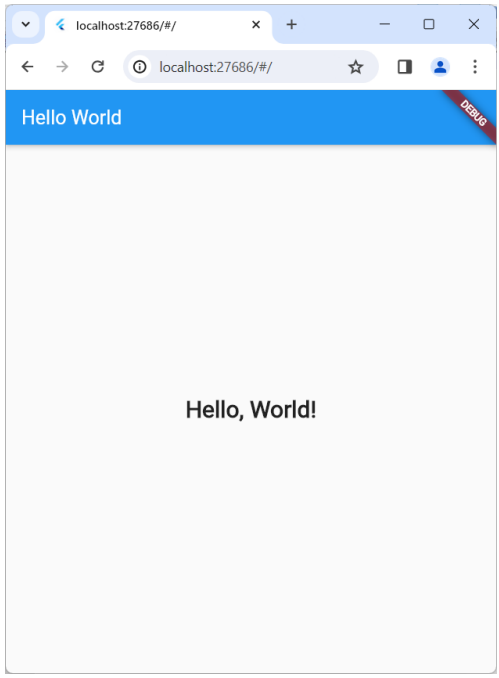
Dalam contoh di atas, kita menggunakan **MaterialApp** sebagai root widget aplikasi, dan kemudian kita menentukan **Scaffold** sebagai halaman utama aplikasi. Di dalam **Scaffold**, kita menentukan **AppBar** dengan judul "Hello World" dan body dengan **Center** yang berisi **Text** "Hello, World!".

Kita juga memberikan gaya pada teks menggunakan properti style pada widget Text, dengan mengatur ukuran font menjadi 24 dan bobot teks menjadi bold.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



G. Evaluasi/Soal Latihan

Apa itu Flutter, dan apa perbedaannya dengan kerangka kerja pengembangan aplikasi mobile lainnya?

BAB 3

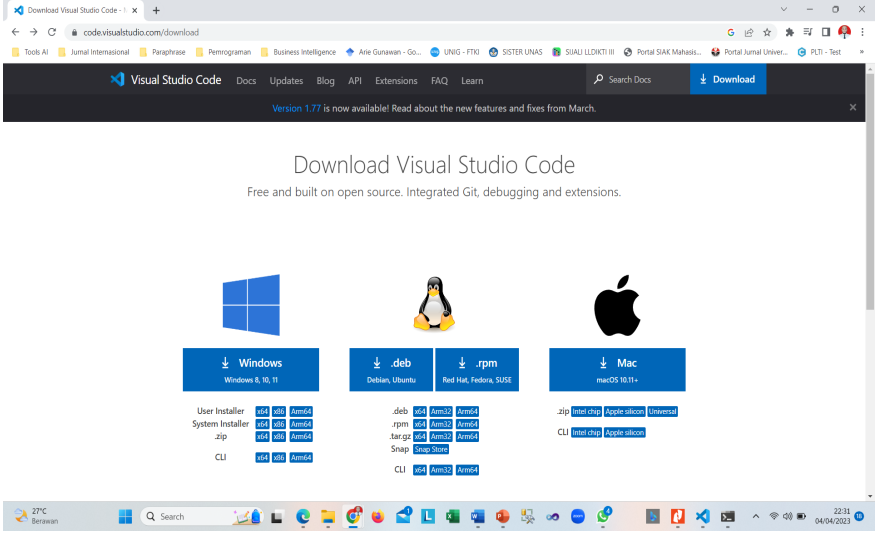
MENGATUR LINGKUNGAN PENGEMBANGAN

A. Tujuan Pembelajaran

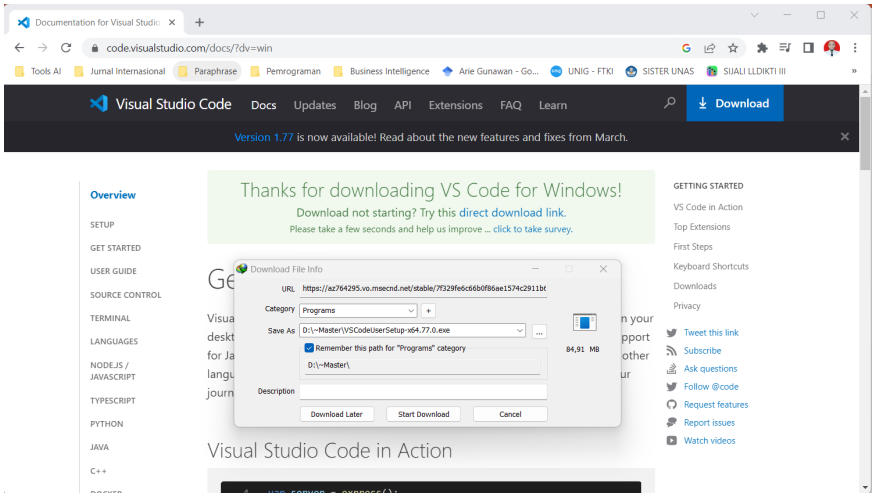
Menguasai instalasi dan konfigurasi Flutter serta penggunaan Visual Studio Code sebagai editor pengembangan.

B. Menginstal Visual Studio Code

Download Aplikasi Visual Code Studio Terbaru pada link https://code.visualstudio.com/Download.



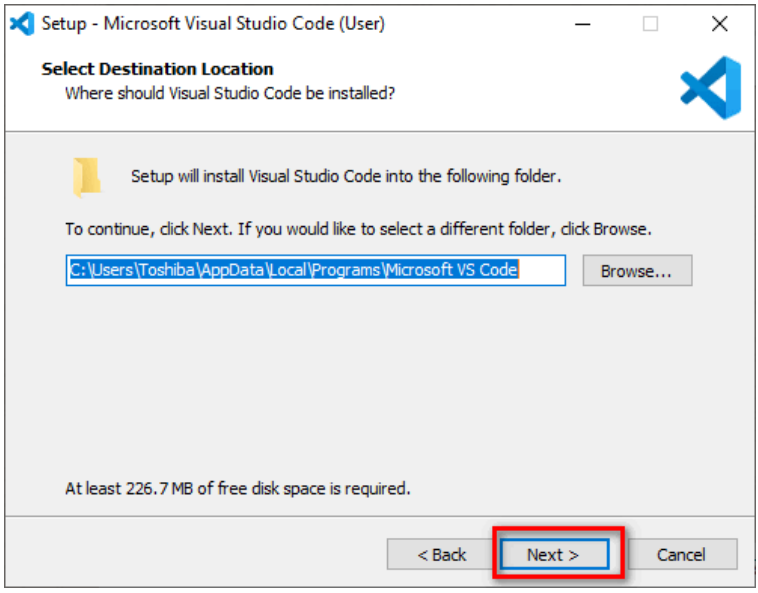
Pilih sistem operasi yang digunakan, disini penulis menggunakan OS Windows. Akan muncul tampilan seperti di bawah ini.



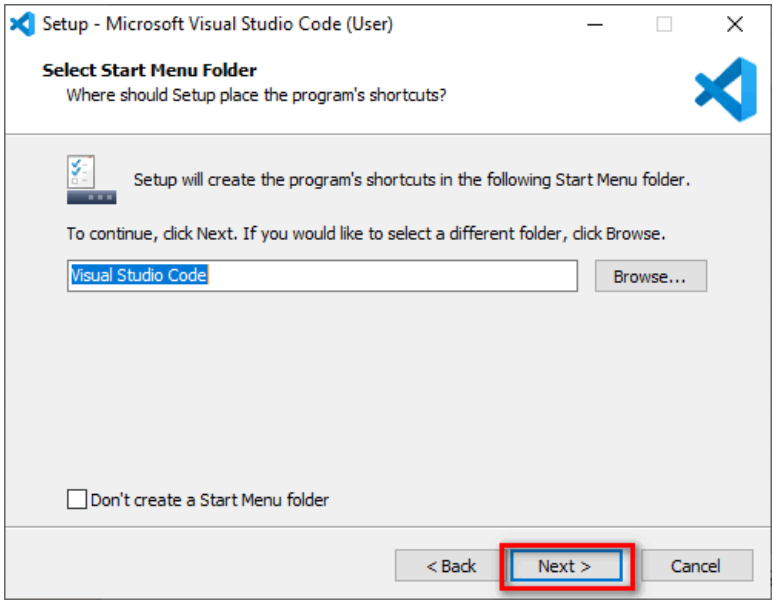
Tunggu hingga proses download selesai.

Double klik pada file installer nya atau klik kanan kemudian pilih Run as Administrator. Jika muncul peringatan Run as Administrator, silahkan klik Yes. Pilih "I accept the aggrement" untuk menyetujui "License Agreement", kemudian klik Next.

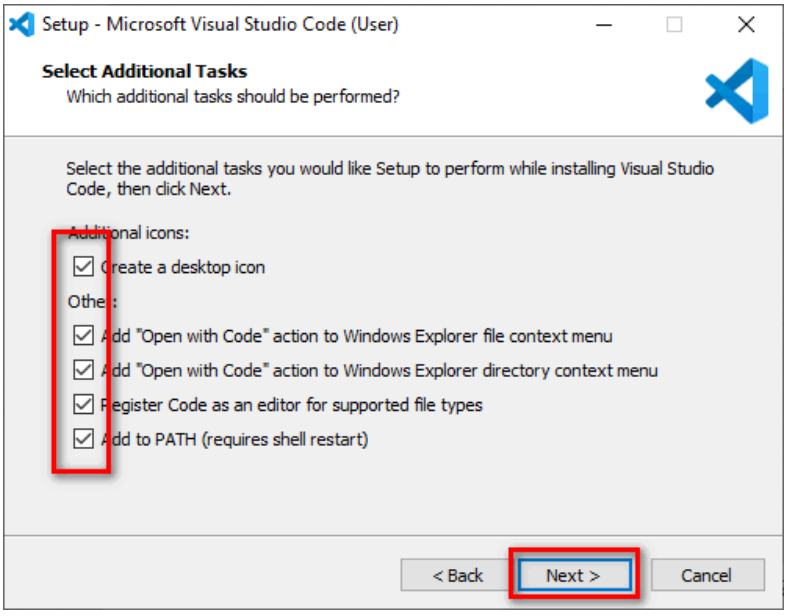
Untuk Select Destination Location bisa di biarkan saja jika lokasi instalasi tidak akan dirubah. Klik **Next**.



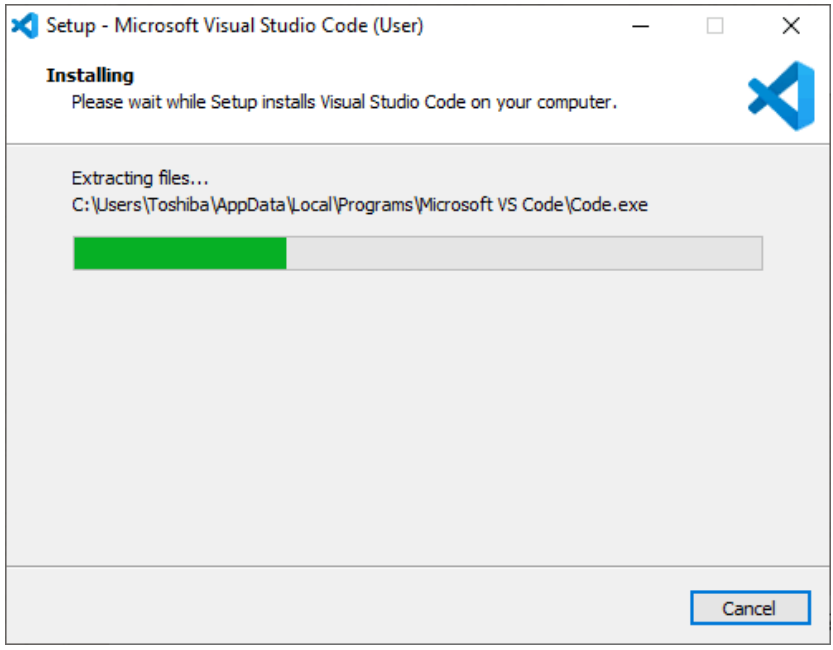
Klik **Next** lagi jika tidak akan merubah Start Menu Folder



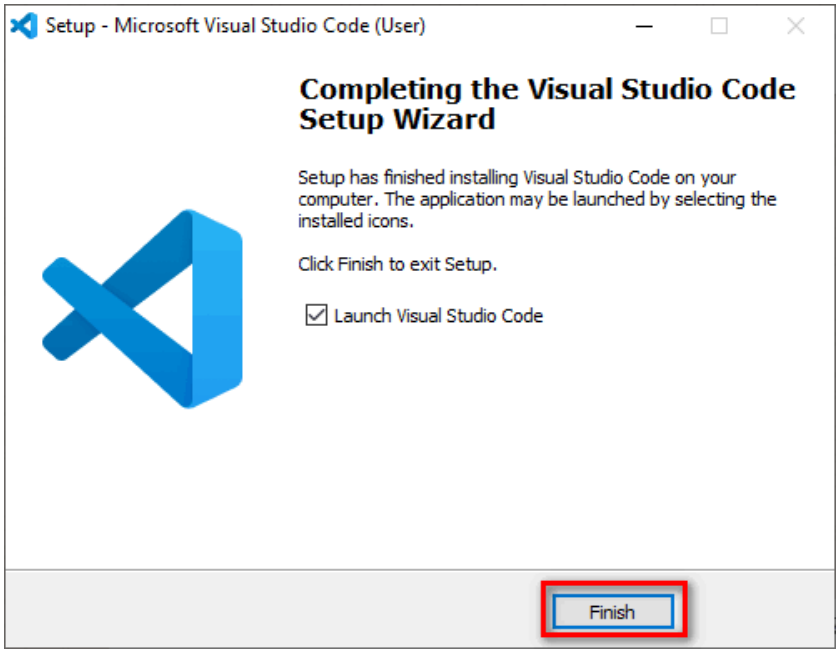
Di bagian Select Additional Tasks centang semua. Kemudian **Next**.



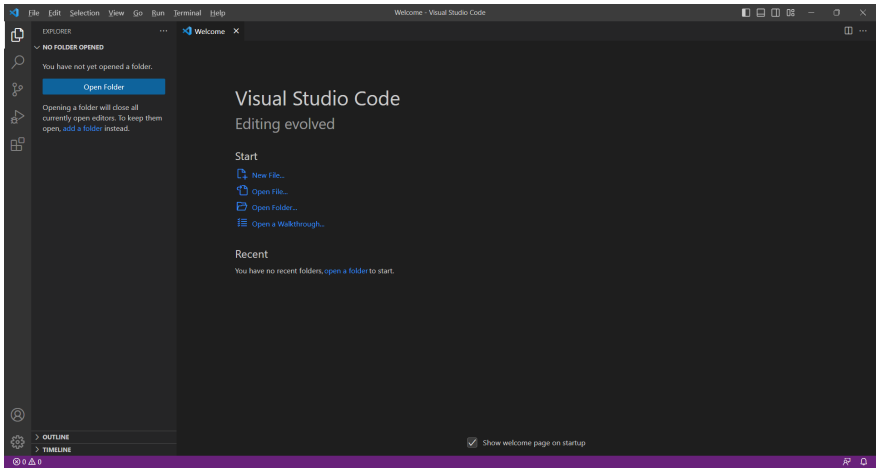
Tunggu sampai proses instalasi selesai.



Setelah selesai klik **Finish**.

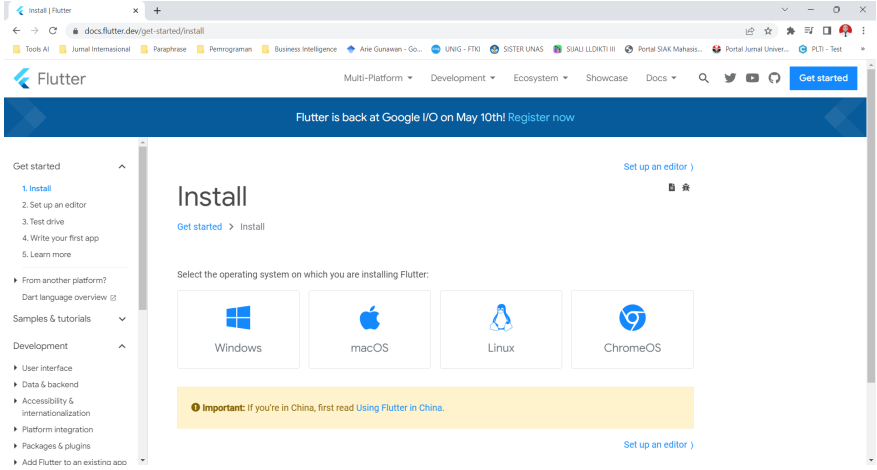


Tampilan Visual Studio Code



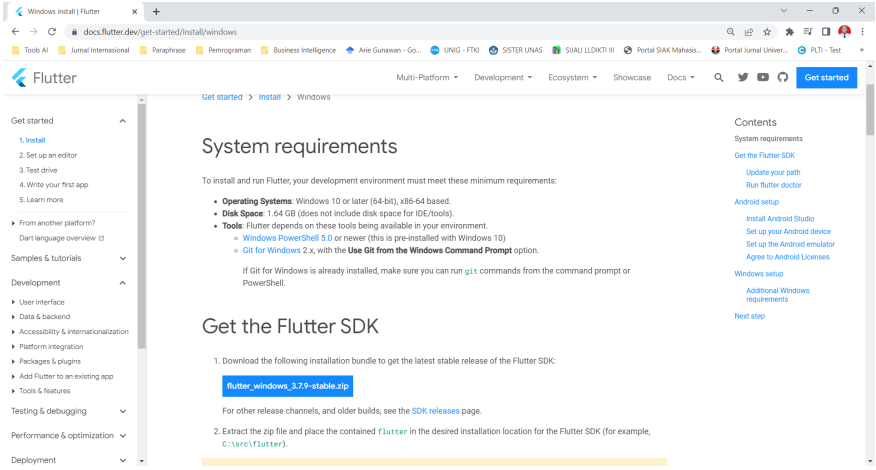
C. Menginstal Flutter SDK

Download Aplikasi Flutter Terbaru pada link https://docs.flutter.dev/get-started/install.



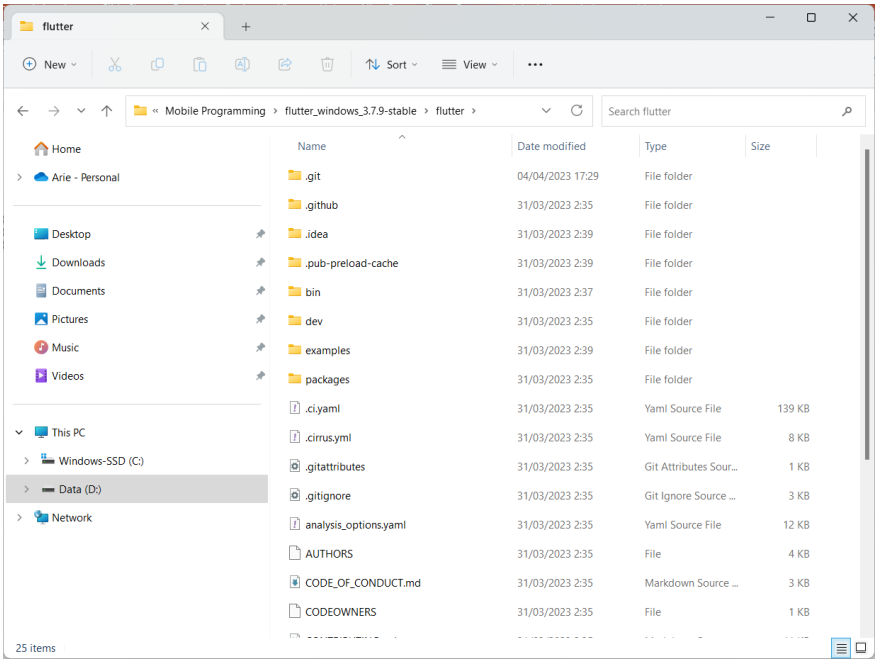
Pilih sistem operasi yang digunakan, disini penulis menggunakan OS Windows.

Setelah itu akan muncul tampilan seperti di bawah ini.



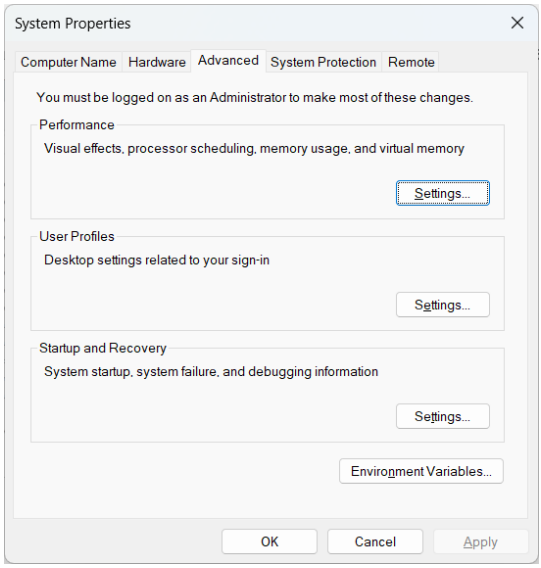
Klik tombol download flutter

Setelah berhasil di download, kemudian extract filenya.



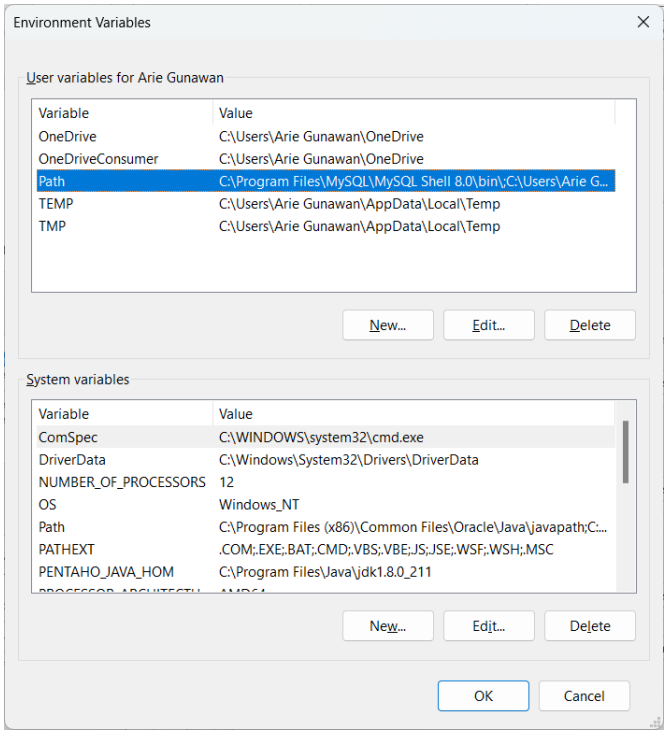
Disini penulis melakukan extract di drive D.

Setelah itu buka windows environtment, caranya klik start cari / search env pada bar pencarian sehingga muncul System Properties.

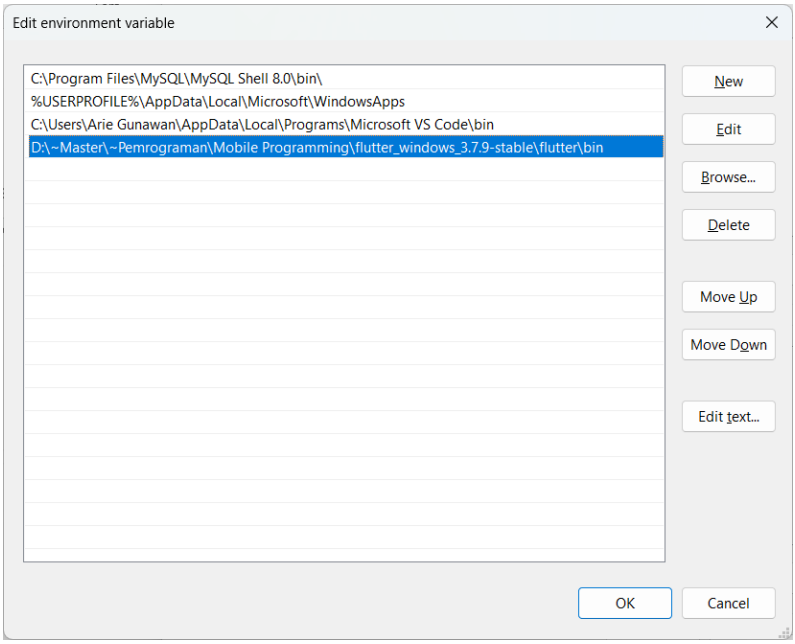


Klik tombol Environment Variables.

Di User variables ..., pilih Path kemudian klik tombol Edit.

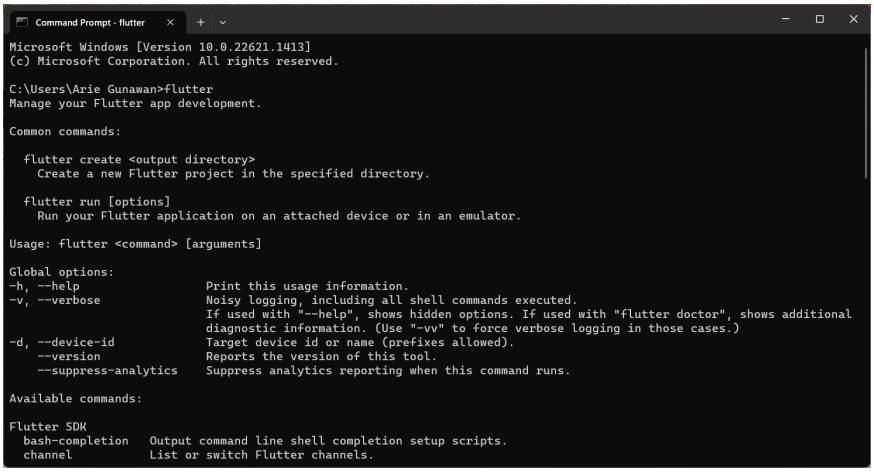


Lalu pada bagian PATH edit sesuaikan folder flutter yang di tempel pada tahap sebelumnya tadi



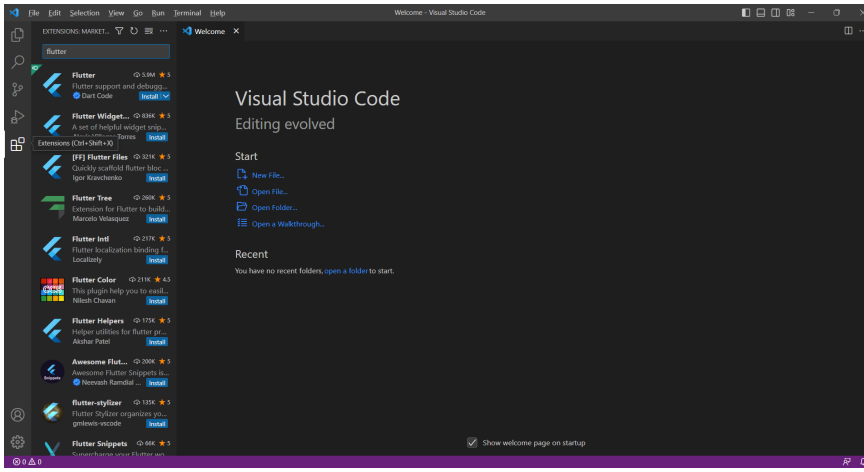
Setelah selesai klik OK. langkah berikutnya buka CMD lalu ketikan perintah > flutter

Bila proses instalasi berhasil maka akan tampil seperti gambar dibawah ini, apabila masih gagal ulangi langkah dari awal.

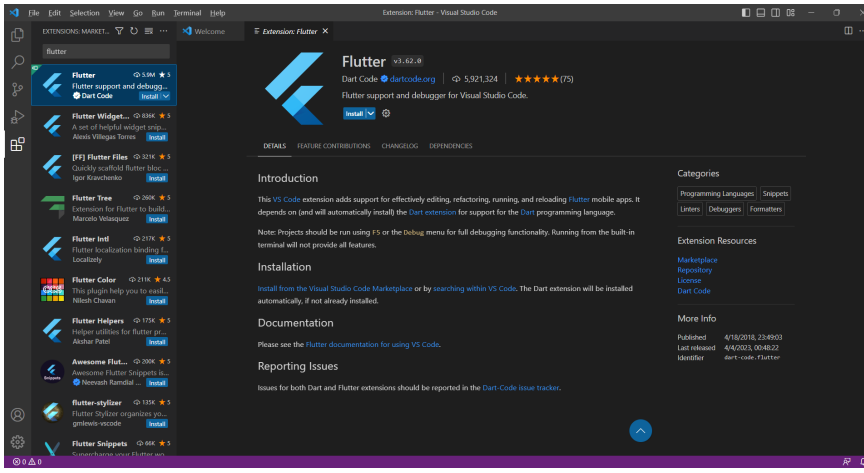


D. Konfigurasi Flutter dan Visual Studio Code

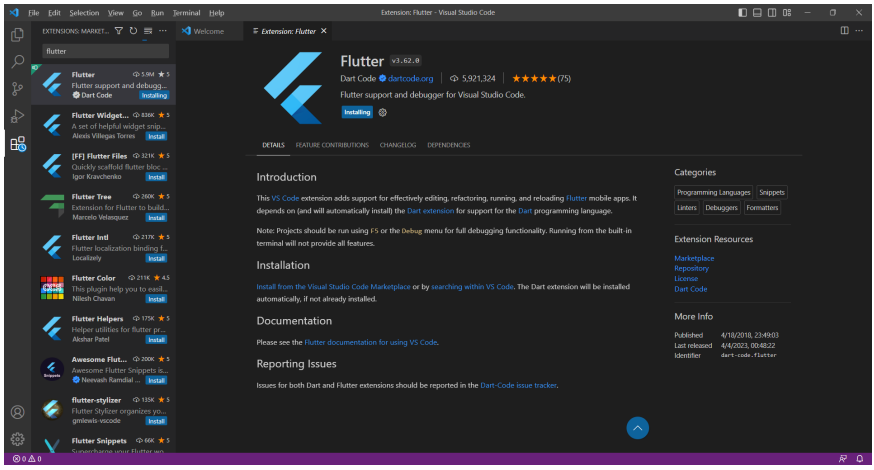
Klik menu Extensions di VS Code, kemudian ketikkan flutter.



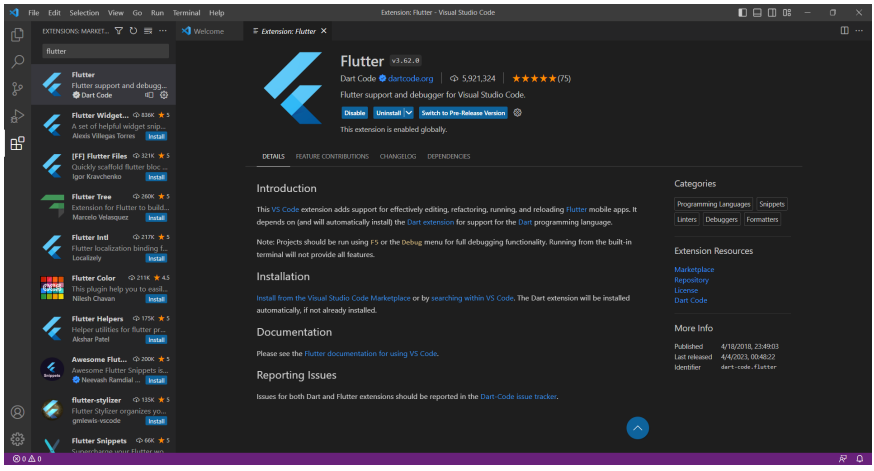
Klik Flutter, kemudian di sebelah kanan klik tombol Install.



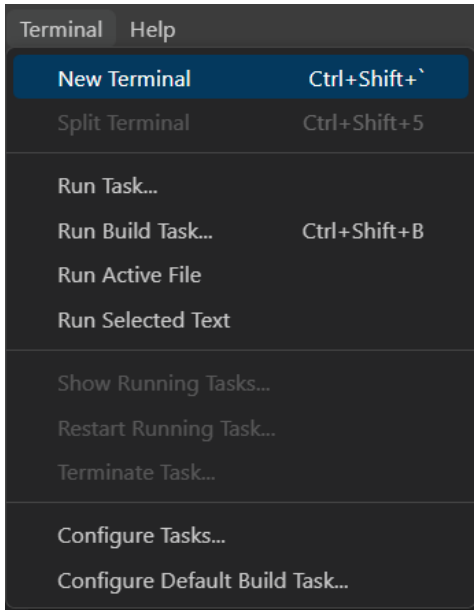
Tunggu hingga proses instalasi selesai



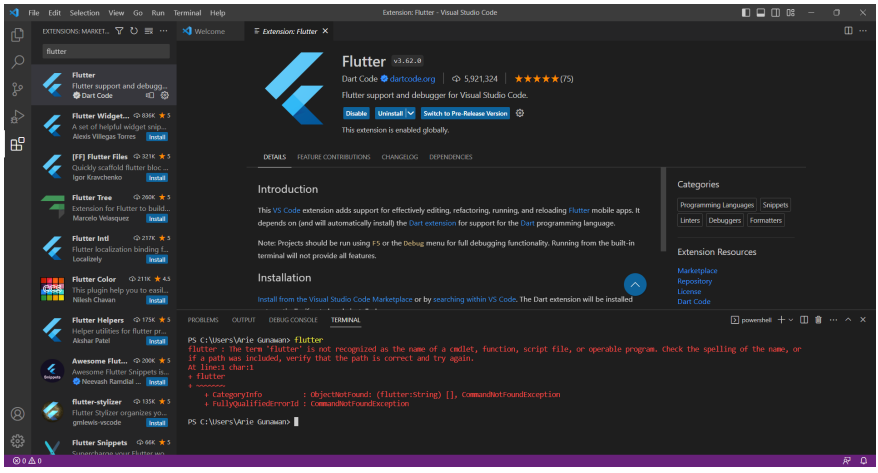
Tampilan setelah flutter selesai diinstall.



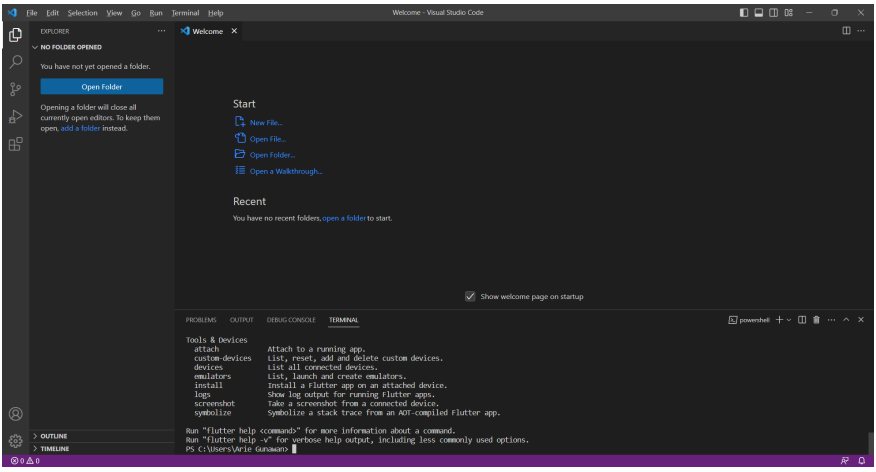
Lakukan cek apakah flutter berhasil diinstall di VS Code dengan cara klik menu Terminal dan pilih New Terminal.



Ketika diketikkan “flutter” dan muncul tampilan seperti di bawah ini berarti masih ada masalah. Path belum dikenali. Cara sederhananya, tutup VS Code kemudian buka lagi.

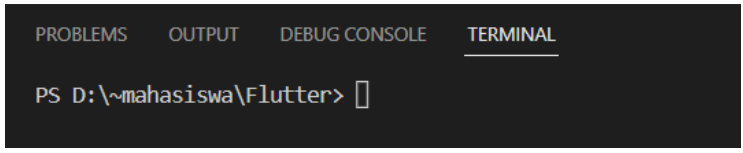


Setelah VS Code ditutup dan dijalankan kembali, maka sudah tidak masalah. Path sudah dikenali.



E. Contoh Koding: Membuat Proyek Flutter Baru

Langkah pertama yang harus dilakukan adalah membuat folder agar memudahkan untuk penyimpanan dan pencarian file. Di sini penulis sudah membuat folder di drive D, tepatnya di D:\~Mahasiswa\Flutter. Ada 2 (dua) cara untuk membuat project, yang pertama menggunakan command line dari terminalnya VS Code. Buka New Terminal dan pastikan path penyimpanan filenya sudah benar.

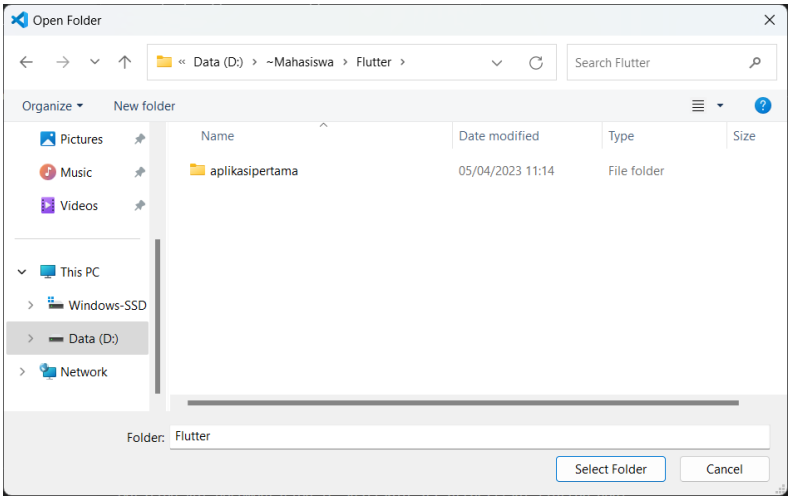


Kemudian ketikkan flutter create aplikasipertama, Enter.

Tunggu beberapa saat hingga tampilannya seperti di bawah ini.

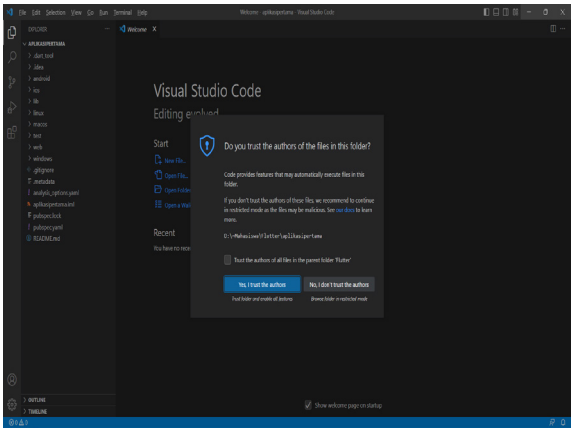
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

  source_maps 0.10.11 (0.10.12 available)
PS D:\~mahasiswa\Flutter> flutter create aplikasipertama
Creating project aplikasipertama...
Running "flutter pub get" in aplikasipertama...
Resolving dependencies in aplikasipertama... (1.5s)
+ async 2.10.0 (2.11.0 available)
+ boolean_selector 2.1.1
+ characters 1.2.1 (1.3.0 available)
+ clock 1.1.1
+ collection 1.17.0 (1.17.1 available)
+ cupertino_icons 1.0.5
+ fake_async 1.3.1
+ flutter 0.0.0 from sdk flutter
+ flutter_lints 2.0.1
+ flutter_test 0.0.0 from sdk flutter
+ js 0.6.5 (0.6.7 available)
+ lints 2.0.1
+ matcher 0.12.13 (0.12.15 available)
+ material_color_utilities 0.2.0 (0.3.0 available)
All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:

  $ cd aplikasipertama
  $ flutter run

Your application code is in aplikasipertama\lib\main.dart.
```

Langkah selanjutnya, klik menu File dan pilih Open Folder



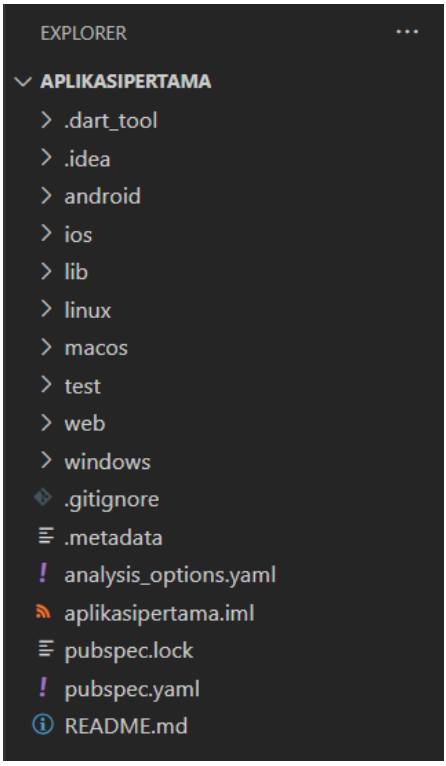
Pilih folder project yang telah dibuat, kemudian klik Select Folder.

Maka akan muncul tampilan seperti di bawah ini.

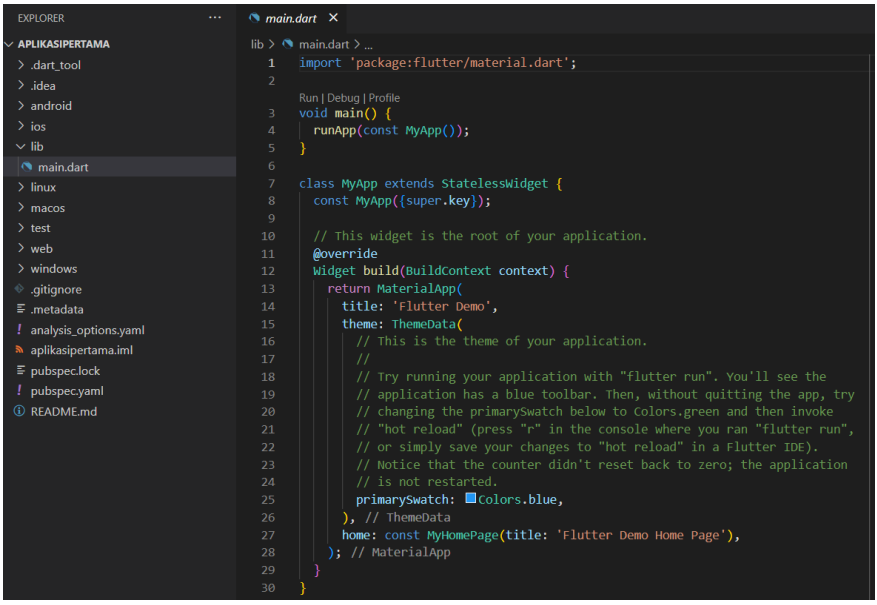


Klik tombol Yes, I trust the authors.

Kemudian akan muncul kerangka projectnya.



Untuk kodingannya ada di folder lib > main.dart



Untuk menjalankan koding, menggunakan terminal. Klik New Terminal. Pastikan folder projectnya sudah sesuai. D:\~Mahasiswa\Flutter\aplikasipertama>. Kemudian ketikkan flutter run, Enter.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
```

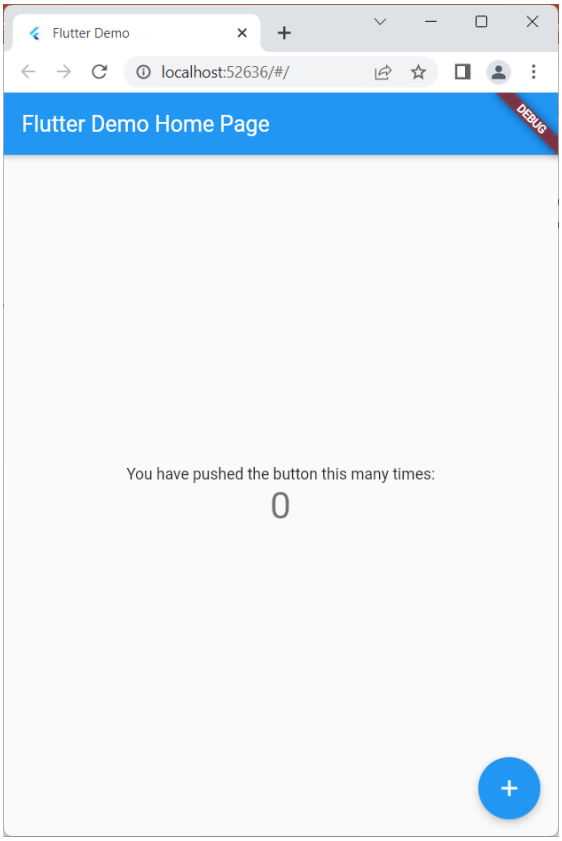
Akan muncul tampilan berikut

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"):
```

Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Tunggu beberapa saat hingga muncul tampilan berikut.



Tampilan di atas menunjukkan bahwa project berhasil dijalankan.

F. Evaluasi/Soal Latihan

Bagaimana cara menginstal Flutter dan konfigurasi Visual Studio Code untuk pengembangan aplikasi Flutter?

BAB 4

MEMULAI PROYEK FLUTTER PERTAMA

A. Tujuan Pembelajaran

Memahami struktur proyek dasar Flutter dan bagaimana cara membuat dan menjalankan proyek Flutter pertama.

B. Struktur Direktori Proyek Flutter

Struktur direktori proyek Flutter biasanya terdiri dari beberapa folder dan file yang berfungsi untuk mengorganisir kode, sumber daya, dan konfigurasi proyek. Berikut adalah penjelasan singkat tentang struktur direktori proyek Flutter yang umum:

1. android: Folder ini berisi proyek Android native untuk proyek Flutter Anda. Ini termasuk file-file seperti proyek Gradle, sumber kode Java/Kotlin, dan konfigurasi Android lainnya.
2. ios: Mirip dengan folder "android", folder ini berisi proyek iOS native untuk proyek Flutter Anda. Ini termasuk file-file seperti proyek Xcode, sumber kode Swift/Objective-C, dan konfigurasi iOS lainnya.
3. lib: Folder ini adalah tempat utama untuk kode Dart aplikasi Anda. Ini adalah tempat dimana Anda akan menulis kode Flutter Anda, termasuk file-file seperti widget, utilitas, layanan, dan logika bisnis aplikasi.
4. test: Folder ini berisi unit test dan widget test untuk proyek Anda. Ini memungkinkan Anda untuk menulis dan menjalankan tes otomatis untuk memastikan bahwa aplikasi Anda berperilaku sesuai yang diharapkan.
5. assets: Jika Anda memiliki file sumber daya statis seperti gambar, font, atau file konfigurasi yang digunakan dalam aplikasi Anda, Anda

dapat meletakkannya di sini. Mereka akan diakses oleh aplikasi Anda melalui Flutter menggunakan path relatif.

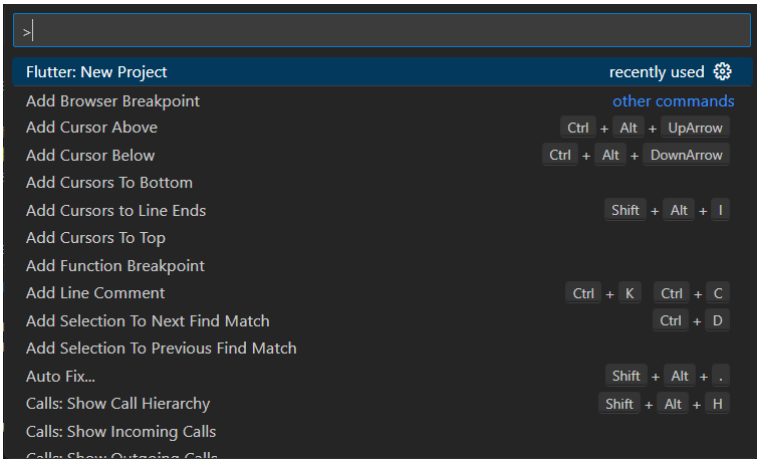
6. pubspec.yaml: File ini adalah file konfigurasi proyek Flutter Anda. Di sini Anda mendefinisikan paket-paket Dart yang digunakan dalam proyek Anda, sumber daya aplikasi seperti font atau gambar, serta konfigurasi proyek lainnya.
7. .metadata: File ini adalah file metadata yang digunakan oleh Flutter untuk melacak dependensi proyek Anda.

Selain itu, Anda juga mungkin akan menemukan file-file konfigurasi lainnya yang tergantung pada alat pengembangan atau editor yang Anda gunakan, seperti .gitignore untuk mengabaikan file saat Anda menggunakan Git, atau file konfigurasi spesifik editor seperti .vscode untuk Visual Studio Code.

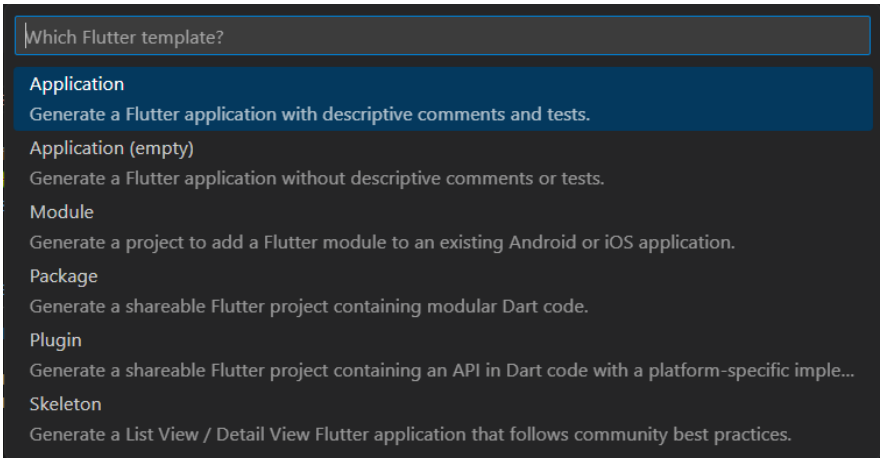
Struktur ini membantu dalam menjaga kode proyek Anda terorganisir dan mudah dikelola, serta memisahkan kode Flutter Anda dari kode platform spesifik seperti Android dan iOS.

C. Menjalankan Aplikasi Pertama

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

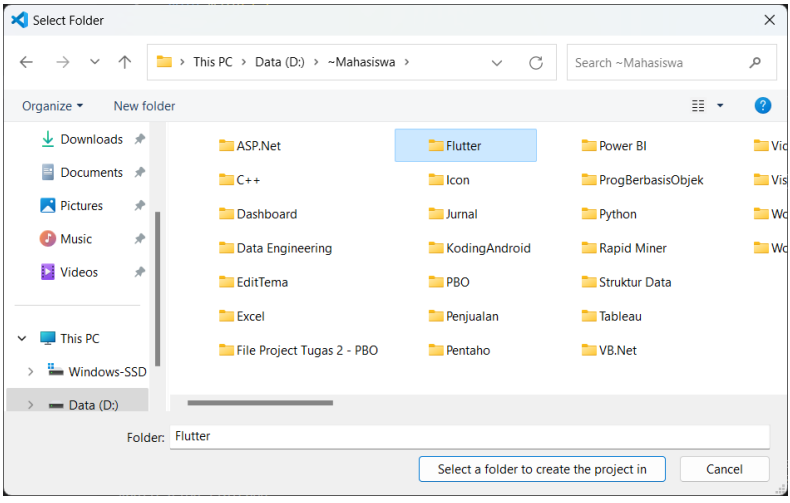


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **aplikasi_pertama** kemudian tekan **Enter**.

Saat membuat project baru, kode program awal yang akan didapatkan pada **main.dart** akan seperti ini:

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // Try running your application with "flutter run". You'll see the
        // application has a blue toolbar. Then, without quitting the app, try
        // changing the primarySwatch below to Colors.green and then
invoke
        // "hot reload" (press "r" in the console where you ran "flutter run",
        // or simply save your changes to "hot reload" in a Flutter IDE).
        // Notice that the counter didn't reset back to zero; the application
        // is not restarted.
        primarySwatch: Colors.blue,
      ),
      home: Center(child: MyHomePage(title: 'Aplikasi Flutter')),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  // This widget is the home page of your application. It is stateful,
meaning
  // that it has a State object (defined below) that contains fields that
affect
```

```
  // how it looks.

  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget)
and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something has
      // changed in this State, which causes it to rerun the build method
below
      // so that the display can reflect the updated values. If we changed
      // _counter without calling setState(), then the build method would
not be
      // called again, and so nothing would appear to happen.
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called, for instance as
done
    // by the _incrementCounter method above.
```

```
    //
    // The Flutter framework has been optimized to make rerunning build
methods
    // fast, so that you can just rebuild anything that needs updating rather
    // than having to individually change instances of widgets.
    return Scaffold(
      appBar: AppBar(
        // Here we take the value from the MyHomePage object that was
created by
        // the App.build method, and use it to set our appbar title.
        title: Text(widget.title),
      ),
      body: Center(
        // Center is a layout widget. It takes a single child and positions it
        // in the middle of the parent.
        child: Column(
          // Column is also layout widget. It takes a list of children and
          // arranges them vertically. By default, it sizes itself to fit its
          // children horizontally, and tries to be as tall as its parent.
          //
          // Invoke "debug painting" (press "p" in the console, choose the
          // "Toggle Debug Paint" action from the Flutter Inspector in Android
          // Studio, or the "Toggle Debug Paint" command in Visual Studio
Code)
          // to see the wireframe for each widget.
          //
          // Column has various properties to control how it sizes itself and
          // how it positions its children. Here we use mainAxisAlignment to
          // center the children vertically; the main axis here is the vertical
          // axis because Columns are vertical (the cross axis would be
          // horizontal).
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
```

```
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.display1,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ), // This trailing comma makes auto-formatting nicer for build
methods.
  );
 }
}
```

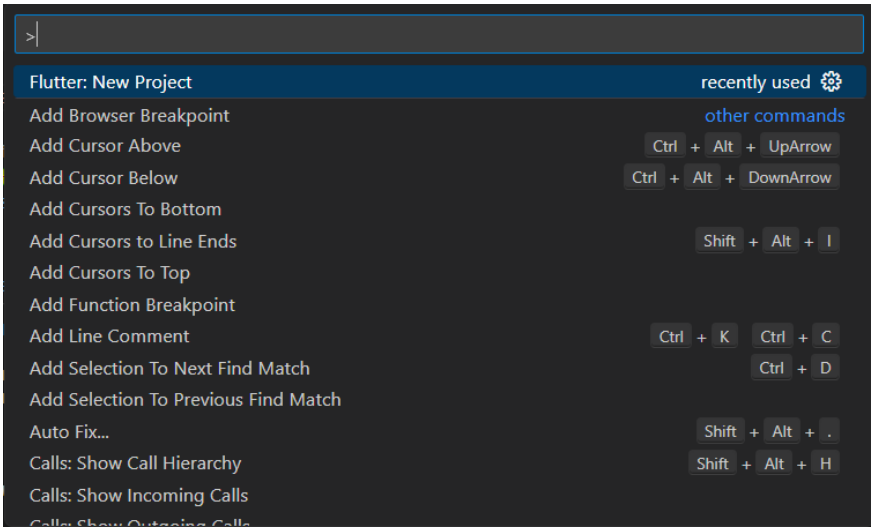
Struktur kode di atas sebenarnya terdiri dari tiga bagian:

1. Bagian import;
   Bagian import adalah tempat kita mendeklarasikan atau mengimpor library yang dibutuhkan pada aplikasi.
2. Bagian main;
   Bagian main adalah fungsi utama dari aplikasi yang akan menjadi entri point. Fungsi ini akan dieksekusi pertama kali saat aplikasi dibuka.
3. Bagian widget.
   Bagian widget adalah tempat kita membuat widget. Aplikasi Flutter sebenarnya terdiri dari susunan widget. Widget bisa kita bilang elemen-elemen seperti Tombol, Teks, Layaout, Image, dan sebagainya.

D. Contoh Koding: Membuat Tampilan Awal Aplikasi Flutter

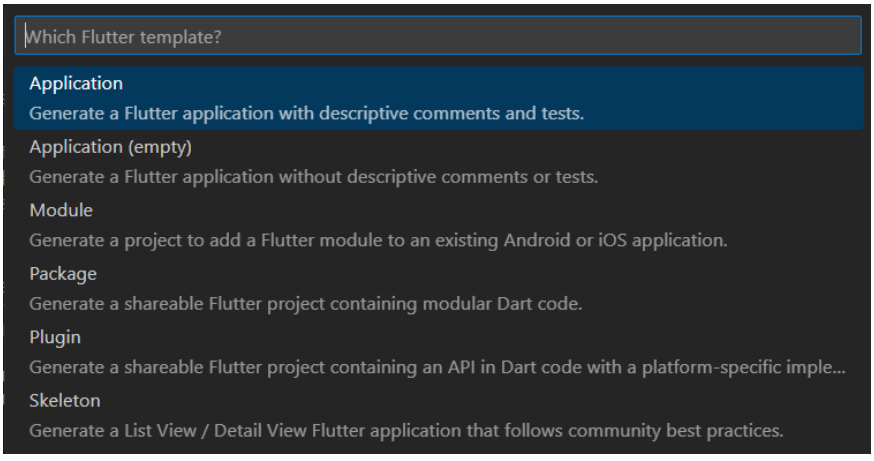
Pada bab sebelumnya telah dibahas cara memulai project flutter, untuk kali ini penulis akan menggunakan cara berbeda dalam memulai project

flutter, yaitu menggunakan Command Pallete. Pilih menu View dan klik Command Pallete sehingga muncul tampilan berikut.

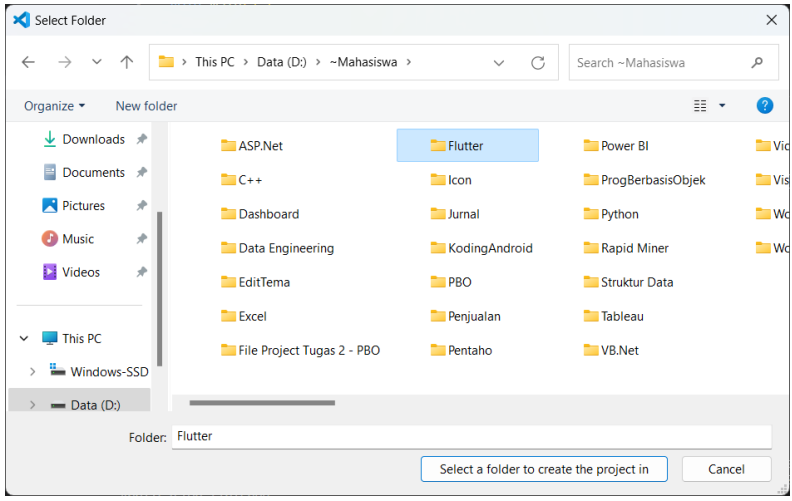


Kemudian ketikkan flutter, pilih Flutter New Project.

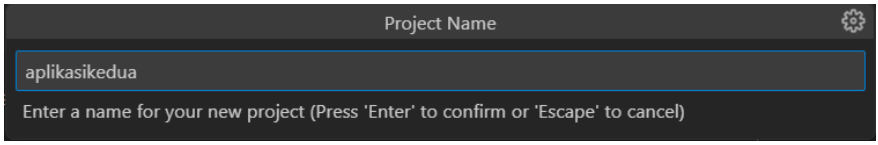
Pilih menu Application.



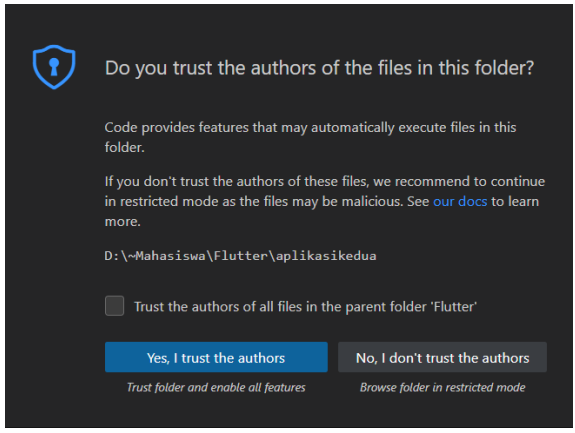
Pilih folder yang diinginkan, kemudian klik tombol Select a folder to create the project in.



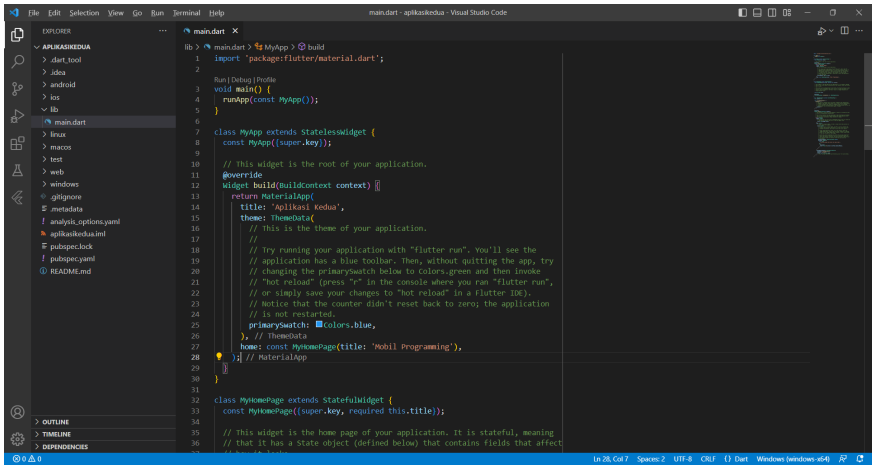
Beri nama projectnya, disini penulis memberi nama aplikasikedua.



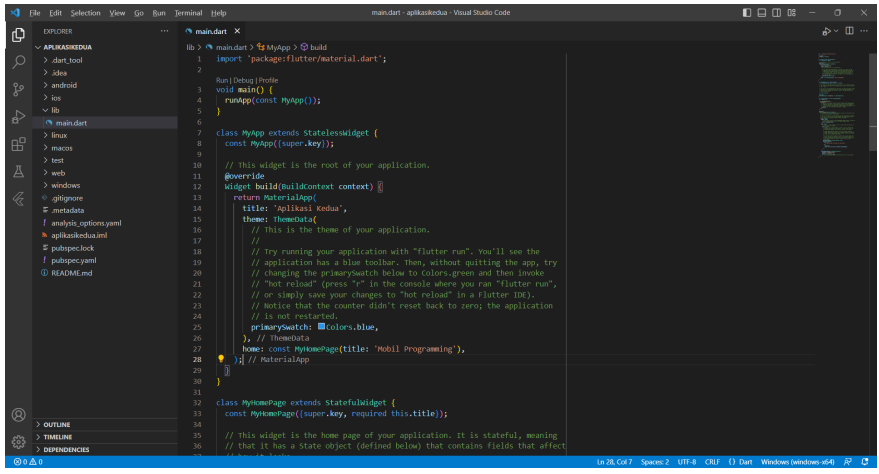
Maka akan muncul tampilan seperti di bawah ini.



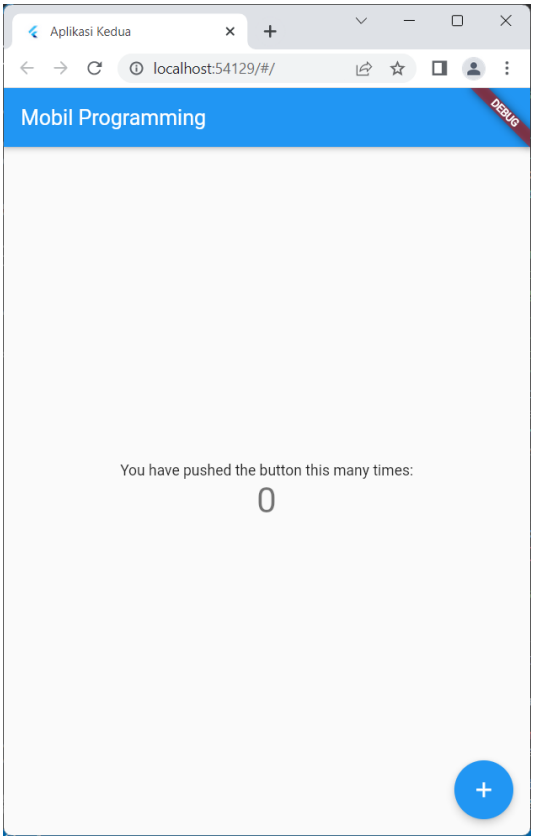
Klik tombol Yes, I trust the authors. Projectnya akan langsung terbuka.



Pada aplikasikedua, penulis merubah titlenya menjadi : 'Aplikasi Kedua', dan home: const MyHomePage(title: 'Mobil Programming')



Untuk menjalankannya masih sama, yaitu menggunakan New Terminal dan ketikkan flutter run.



E. Evaluasi/Soal Latihan

1. Jelaskan struktur dasar proyek Flutter dan bagaimana cara membuat proyek Flutter baru!
2. Sebutkan langkah-langkah yang diperlukan untuk menjalankan aplikasi Flutter pertama kali.

BAB 5

WIDGET DAN LAYOUT DI FLUTTER

A. Tujuan Pembelajaran

1. Mengerti konsep widget dalam Flutter dan bagaimana cara menggunakan widget-widget tersebut untuk membangun antarmuka pengguna (UI).
2. Belajar tentang pembuatan tata letak (layout) sederhana menggunakan widget-widget Flutter.

B. Apa Itu Widget?

Dalam konteks Flutter, "widget" merujuk pada konsep dasar untuk membangun antarmuka pengguna (UI). Widget adalah bagian-bagian kecil dari antarmuka pengguna yang dapat berupa elemen visual seperti tombol, teks, gambar, atau bahkan layout yang lebih kompleks seperti daftar atau kartu.

Di Flutter, hampir semua yang Anda lihat di layar adalah widget. Bahkan, aplikasi Flutter sendiri adalah widget. Widget dalam Flutter dapat dibagi menjadi dua jenis utama:

StatelessWidget: Widget yang tidak memiliki keadaan (state) internal. Artinya, mereka tidak dapat berubah setelah dibuat. Stateless widget bergantung pada input dan konfigurasi yang diberikan untuk menampilkan antarmuka pengguna. Contohnya adalah widget Text yang menampilkan teks statis atau widget Icon yang menampilkan ikon tertentu.

StatefulWidget: Widget yang memiliki keadaan (state) internal yang dapat berubah sepanjang waktu. Stateful widget dapat merespons perubahan input, interaksi pengguna, atau perubahan kondisi aplikasi lainnya dengan memperbarui UI sesuai dengan keadaan (state) yang

berubah. Contohnya adalah widget TextField yang memungkinkan pengguna untuk memasukkan teks dan merespons perubahan input.

Widget dalam Flutter dibangun dengan cara yang berlapis-lapis. Anda dapat menggabungkan widget-widget ke dalam struktur yang lebih kompleks, yang kemudian dapat digunakan kembali sebagai widget yang lebih besar. Konsep ini dikenal sebagai komposisi widget, dan memungkinkan pengembangan UI yang bersih, terorganisir, dan mudah dipelihara.

Selain itu, Flutter memiliki konsep widget "Stateless" dan "Stateful" yang merupakan bagian dari paradigma pembangunan UI deklaratif. Ini berarti UI didefinisikan dalam kode sebagai representasi dari keadaan (state) aplikasi saat ini, dan Flutter akan secara otomatis mengelola perubahan UI sesuai dengan perubahan keadaan aplikasi. Ini adalah salah satu kekuatan utama Flutter yang membuat pengembangan antarmuka pengguna menjadi lebih cepat, mudah, dan konsisten di seluruh platform.

C. Memahami Tata Letak (Layout)

Dalam Flutter, tata letak (layout) merujuk pada cara mengatur dan menempatkan widget-widget dalam antarmuka pengguna. Flutter menyediakan sejumlah widget tata letak yang dapat digunakan untuk mengontrol bagaimana widget-widget tersebut disusun dalam layar perangkat.

Berikut adalah beberapa konsep penting yang perlu dipahami tentang tata letak di Flutter:

1. Widget Layout: Setiap widget memiliki properti tata letak yang memungkinkan Anda mengontrol bagaimana widget tersebut diposisikan dan diatur dalam antarmuka pengguna. Properti tata letak ini dapat digunakan untuk menentukan tinggi, lebar, margin, padding, dan posisi widget.
2. Widget Layout Dasar: Flutter menyediakan widget-layout dasar seperti Container, Row, Column, Stack, dan ListView. Dengan menggunakan

kombinasi dari widget-layout ini, Anda dapat membuat tata letak yang kompleks dan fleksibel untuk aplikasi Anda.

3. Layout Row dan Column: Widget Row digunakan untuk menempatkan widget secara horizontal, sedangkan widget Column digunakan untuk menempatkan widget secara vertikal. Anda dapat menambahkan widget ke dalam Row atau Column dan mengontrol perilaku tata letaknya menggunakan properti seperti mainAxisAlignment, crossAxisAlignment, dan mainAxisSize.
4. Layout Stack: Widget Stack digunakan untuk menempatkan widget di atas satu sama lain, sehingga memungkinkan Anda untuk membuat tumpukan tata letak yang kompleks. Dalam Stack, Anda dapat mengontrol posisi relatif widget menggunakan properti seperti alignment dan Positioned.
5. Layout Container: Widget Container digunakan sebagai wadah umum untuk widget lainnya, dan memberikan fleksibilitas dalam mengatur tata letak dan penampilan widget di dalamnya. Anda dapat mengatur properti seperti margin, padding, color, dan decoration untuk mengubah penampilan dan tata letak widget di dalam Container.
6. Layout ListView: Widget ListView digunakan untuk menampilkan daftar scrollable dari widget, yang berguna ketika Anda memiliki banyak item yang ingin ditampilkan dalam aplikasi Anda. Anda dapat mengontrol perilaku scroll dan penampilan daftar menggunakan properti seperti scrollDirection, shrinkWrap, dan physics.

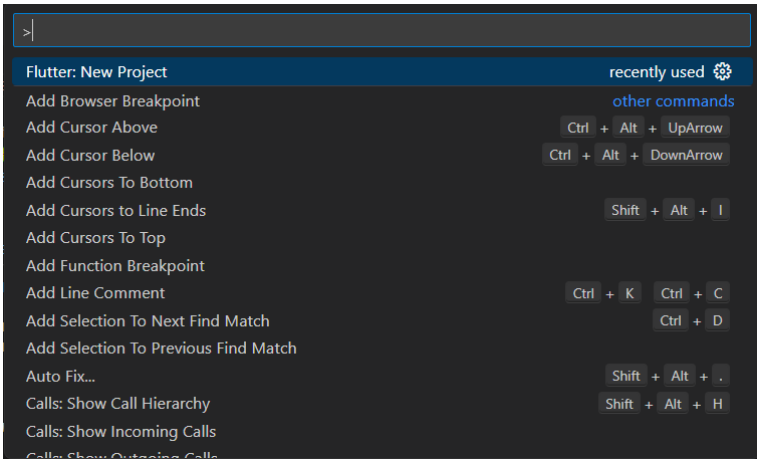
Memahami konsep dasar tata letak di Flutter sangat penting karena membantu Anda membuat antarmuka pengguna yang responsif, estetis, dan mudah dipelihara. Dengan menggunakan widget-layout yang tepat dan memahami cara mengatur widget-widget dalam aplikasi Anda, Anda dapat menciptakan pengalaman pengguna yang optimal untuk pengguna Anda.

D. Membuat Widget Sederhana

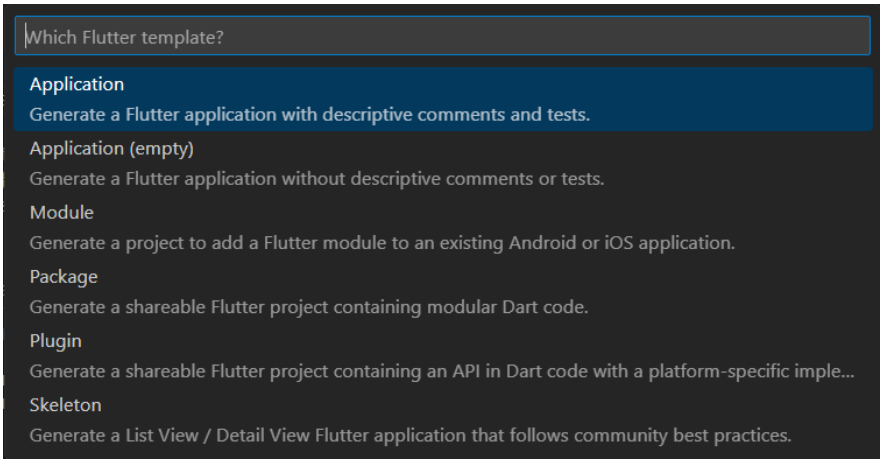
AppBar mungkin bisa disinonimkan dengan tag <header> pada HTML, sebab AppBar merupakan fungsi untuk membuat head dari sebuah

aplikasi, dimana didalamnya terdapat title yang dapat digunakan untuk menampilkan brand atau page apa yang sedang dibuka dari aplikasi tersebut.

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

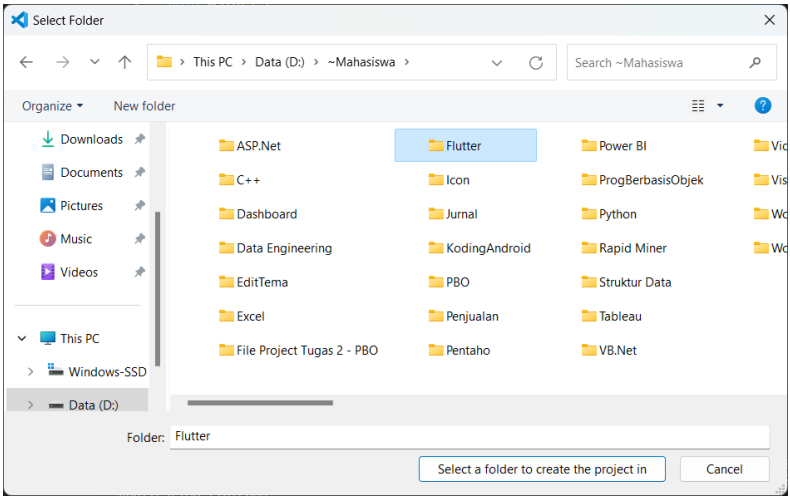


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **widget_sederhana** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

void main() {
  runApp(HomePage());
}

class HomePage extends StatelessWidget {
  build(context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.red[800],
          leading: Icon(Icons.home),
          title: Text('Flutter Widget Sederhana')
        ),
      )
```

```
    );
  }
}
```

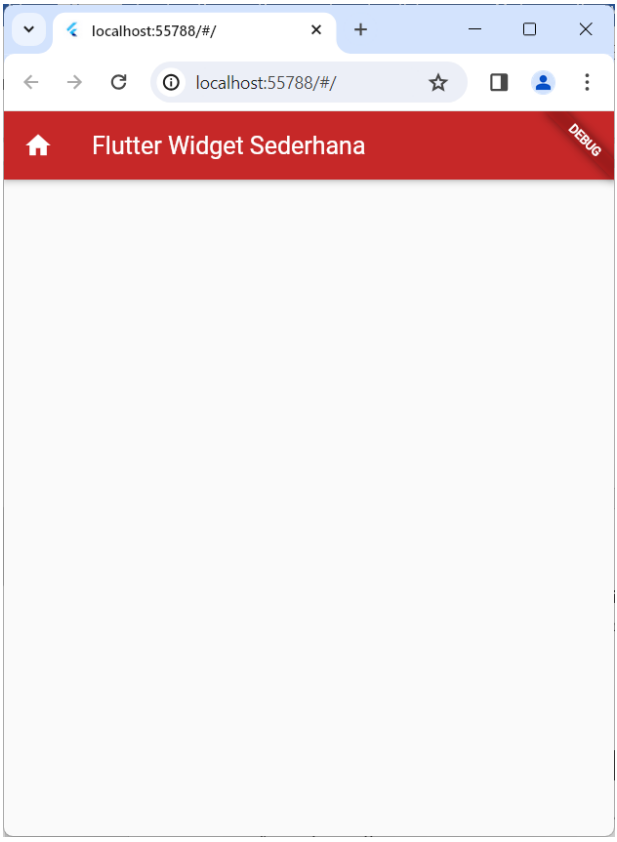
Penjelasan:

1. Line-1, import package yang dibutuhkan, dalam hal ini material.dart. main() adalah fungsi yang pertama kali diajalankan ketika aplikasi sedang di-load, maka apapun yang di-apit didalam main() maka code tersebut akan di-eksekusi.
2. Line-4 kita menggunakan runApp untuk me-render code kedalam screen aplikasi, dalam hal ini terdapat class HomePage() yang akan di-eksekusi.
3. Line-7, kita mendefinisikan sebuah class yang bernama HomePage.
4. Line-9, memberikan nilai balik yang berisi MaterialApp dari package yang telah di-import pada awal code.
5. MaterialApp memiliki property home yang berisi Scaffold widget. Perlu diketahui bahwa Scaffold widget inilah yang memiliki property appBar untuk membuat bar dari sebuah aplikasi, selain appBar, widget ini juga memiliki property lainnya, yakni: BottomAppBar, FloatingActionButton, dan lain sebagainya.
6. Property appBar dari Scaffold berisi AppBar widget.
7. AppBar widget juga memiliki banyak property, diantaranya: title, leading, actions, dan lain sebagainya. Tapi dalam case kali ini kita akan menggunakan property title yang berisi Text widget untuk menampilkan teks yang di-inginkan, leading untuk menampilkan icon home tepat sebelum teks dari title ditampilkan dan backgroundColor untuk memberikan warna pada AppBar.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



E. Mengatur Tata Letak

Untuk mengatur tata letak widget di Flutter, Anda dapat menggunakan berbagai widget-layout yang disediakan oleh Flutter, seperti Row, Column, Stack, Container, dan lainnya. Di bawah ini adalah panduan umum untuk mengatur tata letak widget di Flutter:

1. Menggunakan Widget-Layout Dasar:
    - Row: Digunakan untuk menempatkan widget secara horizontal. Anda dapat menambahkan widget ke dalam Row dan mengatur tata letaknya menggunakan properti seperti mainAxisAlignment, crossAxisAlignment, dan mainAxisSize.
    - Column: Digunakan untuk menempatkan widget secara vertikal. Anda dapat menambahkan widget ke dalam Column dan mengatur tata letaknya menggunakan properti yang sama seperti Row.
    - Stack: Digunakan untuk menempatkan widget di atas satu sama lain. Anda dapat menambahkan widget ke dalam Stack dan mengatur posisi relatifnya menggunakan widget Positioned.
2. Menggunakan Widget Container:
    - Container: Digunakan sebagai wadah umum untuk widget lainnya. Anda dapat mengatur tata letak dan penampilan widget di dalam Container menggunakan properti seperti margin, padding, alignment, color, decoration, dan lainnya.
3. Menggunakan Widget ListView:
    - ListView: Digunakan untuk menampilkan daftar scrollable dari widget. Anda dapat menambahkan widget ke dalam ListView dan mengontrol perilaku scroll dan penampilannya menggunakan properti seperti scrollDirection, shrinkWrap, dan physics.
4. Menggunakan Widget Expanded dan Flexible:
    - Expanded: Digunakan untuk memperluas widget dalam Row atau Column untuk mengisi ruang yang tersedia tambahan.
    - Flexible: Digunakan untuk memberikan fleksibilitas dalam menyesuaikan ukuran widget dalam Row atau Column sesuai kebutuhan.
5. Menggunakan Widget SizedBox dan Spacer:
    - SizedBox: Digunakan untuk menambahkan ruang kosong dengan lebar dan tinggi tertentu di antara widget lainnya.

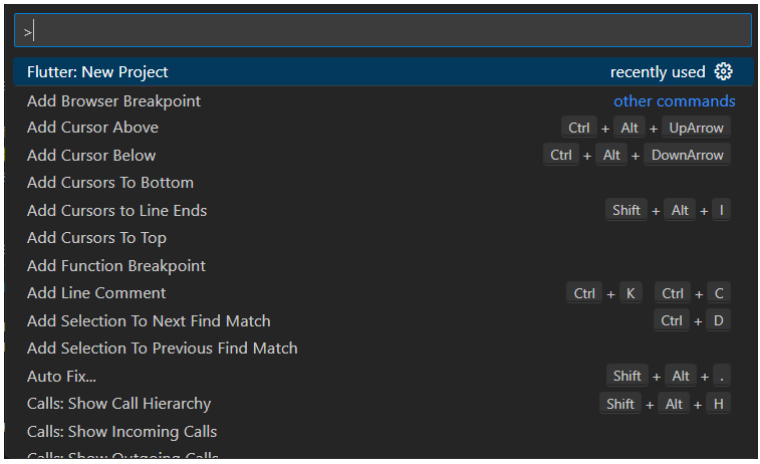
- Spacer: Digunakan untuk menambahkan ruang kosong yang dapat mengisi ruang yang tersedia dalam Row atau Column.

6. Menggunakan Properti Padding:
   - Properti padding dapat digunakan pada widget tertentu untuk menambahkan ruang di sekeliling widget tersebut.

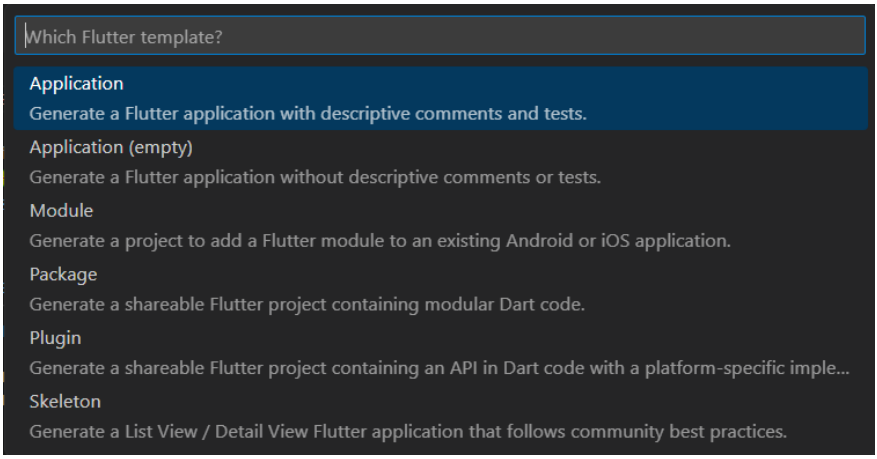
Dengan menggabungkan dan mengatur widget menggunakan widget-layout dan properti yang sesuai, Anda dapat membuat tata letak yang sesuai dengan kebutuhan aplikasi Anda dalam Flutter. Penting untuk memahami konsep tata letak dan memilih widget dan properti yang tepat untuk mencapai tata letak yang diinginkan.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

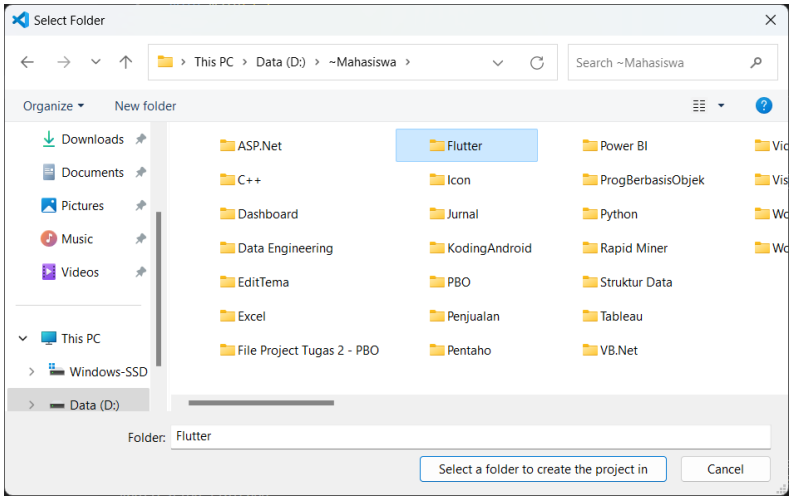


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **widget_layout** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

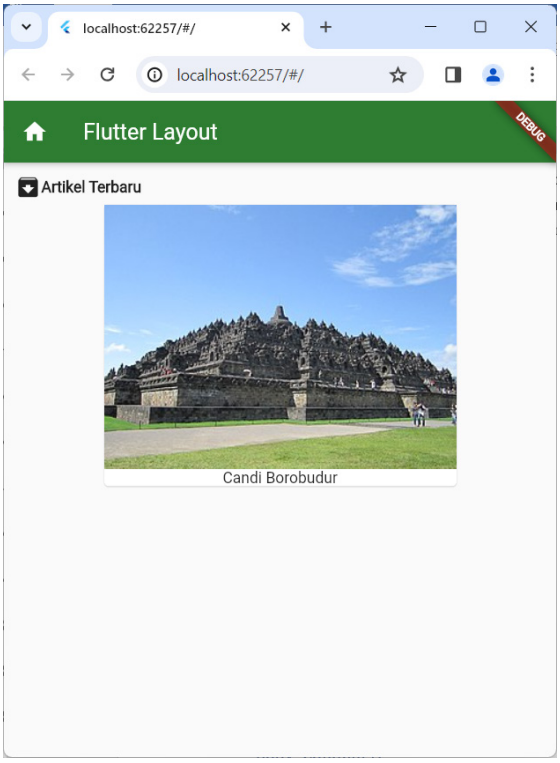
```
import 'package:flutter/material.dart';

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

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

 @override
 build(context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green[800],
        leading: const Icon(Icons.home),
        title: const Text('Flutter Layout')),

      //YANG DIMODIFIKASI
      body: Container(
        margin: const EdgeInsets.all(
          10.0), //CODE BARU UNTUK MENGATUR MARGIN
        child: Column(children: <Widget>[
         Row(children: const <Widget>[
          Icon(Icons.archive),
          Text('Artikel Terbaru',
            style: TextStyle(fontWeight: FontWeight.bold))
         ]),
         Card(
          child: Column(children: <Widget>[
           Image.network(

'https://upload.wikimedia.org/wikipedia/commons/thumb/7/73/Boro
budur_Temple.jpg/320px-Borobudur_Temple.jpg'),
            const Text('Candi Borobudur')
```

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

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

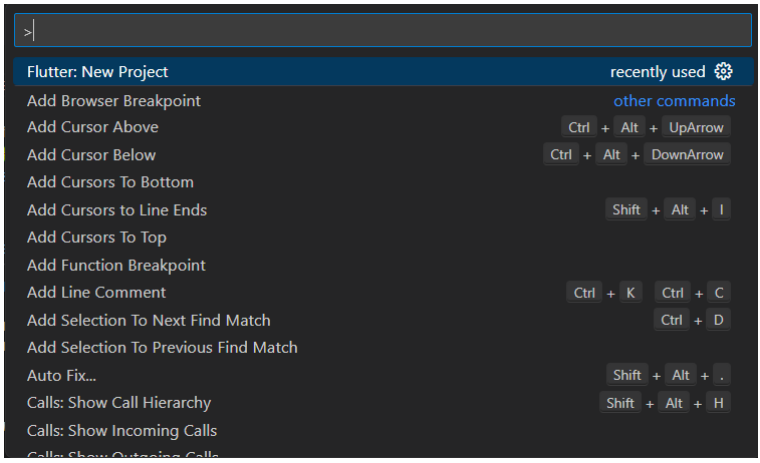


F. Widget Bertingkat

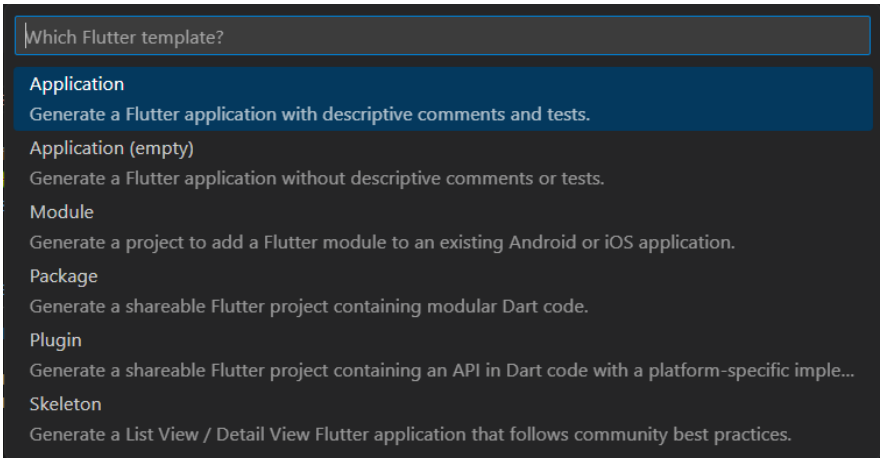
Widget bertingkat dalam konteks Flutter merujuk pada penggunaan widget di dalam widget lainnya secara berlapis-lapis. Dalam pengembangan aplikasi Flutter, seringkali akan membuat tata letak atau struktur UI yang kompleks dengan menyusun widget-widget ke dalam hierarki yang terorganisir.

Konsep widget bertingkat ini memungkinkan untuk membuat tampilan yang kompleks dengan cara yang terstruktur dan mudah dipahami. Di bawah ini adalah contoh sederhana tentang bagaimana widget bertingkat dapat digunakan:

Misalkan jika ingin membuat tampilan sederhana yang terdiri dari gambar diikuti dengan teks di bawahnya, bisa menggunakan widget bertingkat. Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

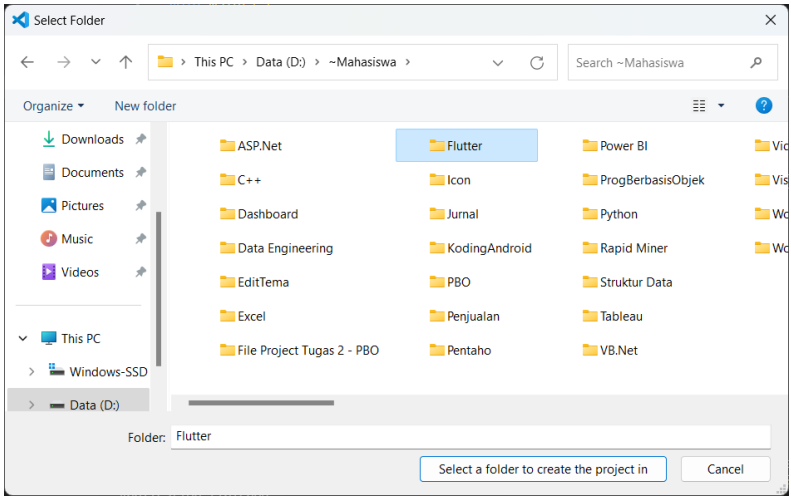


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



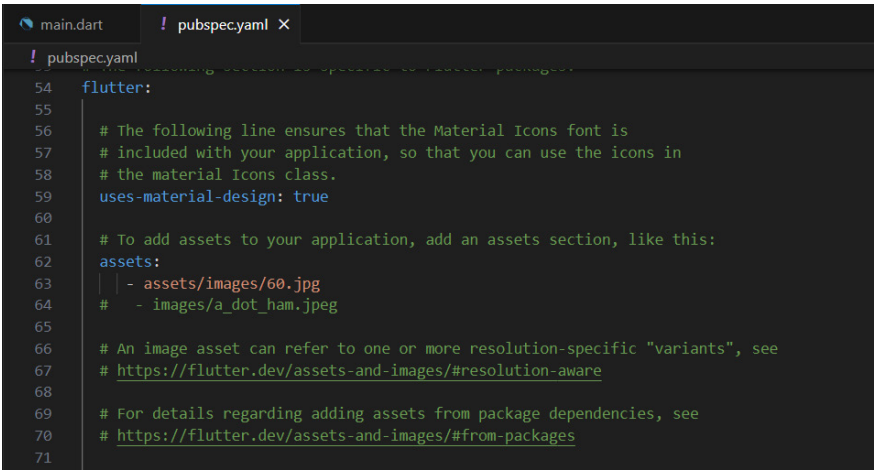
Beri nama projectnya, dengan mengetik **widget_bertingkat** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

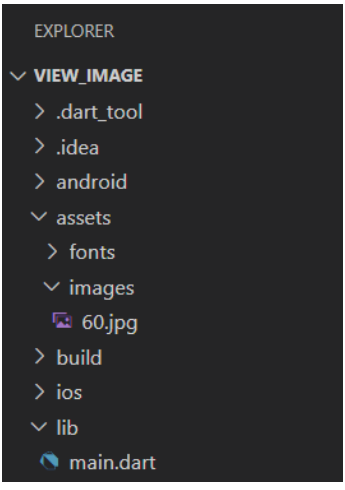
```
import 'package:flutter/material.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Widget Bertingkat'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Image.asset(
                'assets/images/60.jpg',
                width: 150,
                height: 150,
              ),
              const SizedBox(height: 20),
              const Text(
                'Ini adalah aplikasi Flutter',
                style: TextStyle(fontSize: 20),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Kemudian buka file pubspec.yaml dan tambahkan baris berikut ini:

```
assets:
  - assets/images/60.jpg
```



Sebelumnya, terlebih dahulu buatlah folder baru dan beri nama assets, kemudian di dalam folder assets buat folder lagi dan beri nama images. Copy kan image yang dibutuhkan dan paste di folder assets/images (untuk images nya silahkan googling kemudian namanya disesuaikan dengan pubspec.yaml.

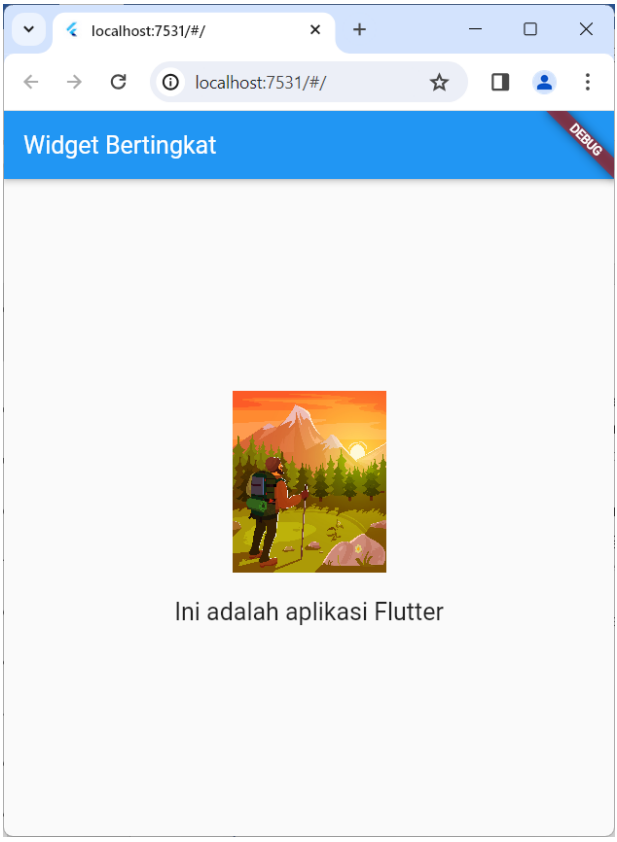


Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah

emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



Dalam contoh di atas, kita menggunakan widget bertingkat sebagai berikut:

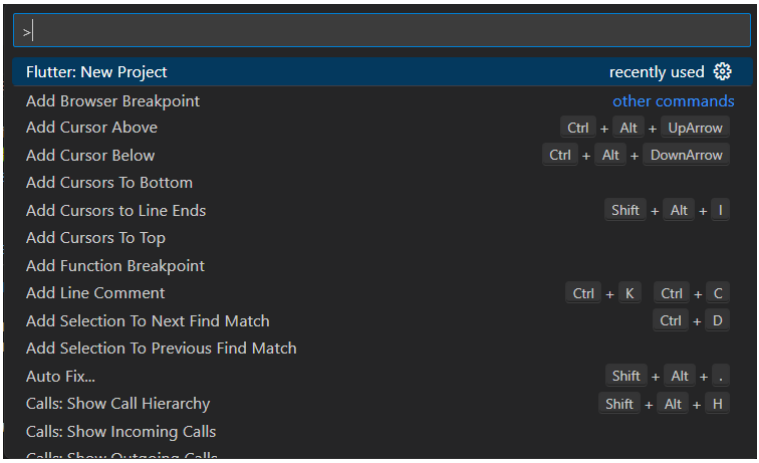
1. Column digunakan sebagai widget layout utama untuk menempatkan widget secara vertikal.

2. Di dalam Column, kita menempatkan Image (gambar) yang diikuti dengan SizedBox untuk memberikan jarak vertikal.
3. Setelah SizedBox, kita menempatkan Text yang berisi teks yang kita ingin tampilkan.

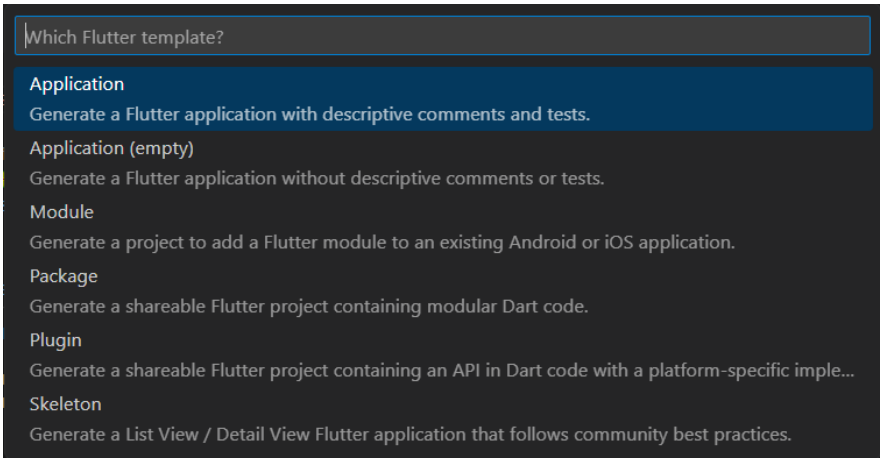
Dengan menggunakan widget bertingkat seperti ini, kita dapat dengan mudah membuat tampilan yang kompleks dengan susunan yang terstruktur dan mudah dimengerti. Penting untuk memahami bagaimana widget bertingkat dapat digunakan untuk mengatur tata letak dan struktur UI dalam aplikasi Flutter.

G. Contoh Koding: Membuat Tampilan Login Sederhana

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

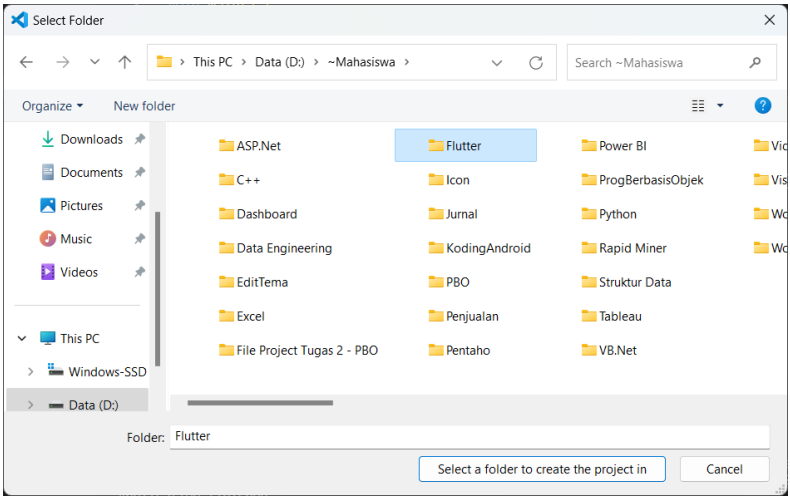


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **login_sederhana** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Login App',
   theme: ThemeData(
    primarySwatch: Colors.blue,
    visualDensity: VisualDensity.adaptivePlatformDensity,
   ),
   home: const LoginPage(),
  );
 }
}

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

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Login'),
   ),
   body: Center(
    child: Padding(
     padding: const EdgeInsets.all(20.0),
     child: Column(
```

```
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const TextField(
            decoration: InputDecoration(
              hintText: 'Username',
            ),
          ),
          const SizedBox(height: 20),
          const TextField(
            obscureText: true,
            decoration: InputDecoration(
              hintText: 'Password',
            ),
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              // Tombol login ditekan
              // Tempatkan logika autentikasi di sini
              // Misalnya, periksa kredensial dengan backend server
              // dan arahkan pengguna ke halaman beranda jika berhasil
            },
            child: const Text('Login'),
          ),
        ],
      ),
    ),
  );
 }
}
```

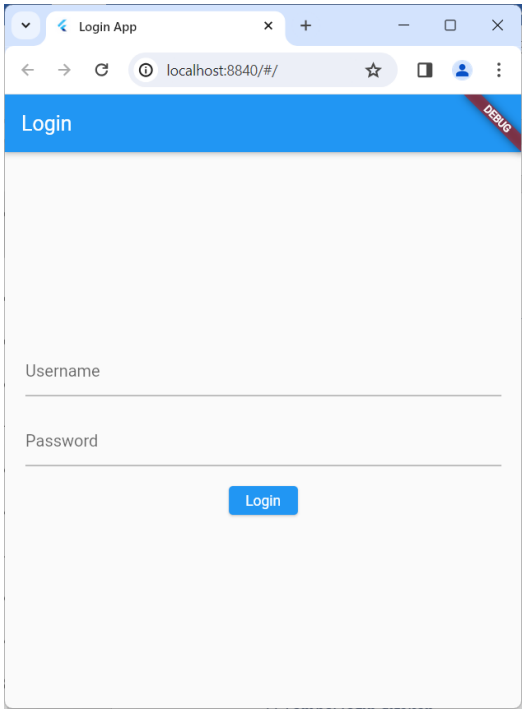
- LoginApp adalah kelas utama aplikasi yang mengatur tema dan halaman awal.
- LoginPage adalah halaman login utama yang berisi TextField untuk memasukkan nama pengguna dan kata sandi, serta ElevatedButton untuk melakukan login.

Saat tombol login ditekan, Anda dapat menambahkan logika autentikasi sesuai kebutuhan aplikasi Anda. Dalam contoh ini, logika autentikasi ditandai dengan komentar, tetapi biasanya Anda akan melakukan permintaan HTTP atau komunikasi dengan backend untuk memverifikasi kredensial pengguna.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



H. Evaluasi/Soal Latihan

1. Apa yang dimaksud dengan widget dalam konteks pengembangan aplikasi Flutter?
2. Bagaimana cara membuat tata letak (layout) sederhana menggunakan widget-widget Flutter?
3. Apa perbedaan antara stateless widget dan stateful widget dalam Flutter?

BAB 6

STATE DAN INTERAKSI PENGGUNA

A. Tujuan Pembelajaran

Memahami konsep state dalam Flutter dan bagaimana cara mengelola keadaan dalam aplikasi menggunakan stateful widget.

B. Memahami Konsep State Di Flutter

Di dalam Flutter, terdapat dua jenis state utama yang perlu dipahami: stateless dan stateful. Stateless widget adalah widget yang tidak memiliki data atau keadaan internal yang berubah sepanjang waktu. Mereka hanya bergantung pada input yang diberikan dan menampilkan tampilan yang sesuai berdasarkan input tersebut. Sebaliknya, stateful widget adalah widget yang memiliki data atau keadaan internal yang dapat berubah. Mereka dapat merespons perubahan input atau interaksi pengguna dengan memperbarui tampilan berdasarkan keadaan yang berubah.

Penggunaan stateful widget memerlukan pemahaman tentang lifecycle widget. Setiap stateful widget memiliki metode khusus yang dipanggil dalam siklus hidupnya, seperti **initState()** yang dipanggil sekali ketika widget pertama kali dibuat, dan **setState()** yang digunakan untuk memperbarui tampilan widget ketika keadaan internal berubah. Memahami lifecycle ini penting untuk menjaga konsistensi antara data dan tampilan dalam aplikasi Flutter.

Flutter menggunakan pendekatan "pemrograman reaktif" untuk mengelola state, di mana UI merespons perubahan state secara otomatis. Ini berarti ketika state berubah, Flutter secara otomatis membangun ulang bagian-bagian UI yang berdampak dan memperbarui tampilannya. Pendekatan ini membuat pengembangan UI menjadi lebih mudah karena Anda tidak perlu secara manual memperbarui UI setiap kali state berubah.

Salah satu pendekatan yang umum digunakan untuk mengelola state di Flutter adalah dengan menggunakan provider. Provider adalah paket Flutter yang digunakan untuk mengelola state dan berbagi data antara widget di seluruh aplikasi. Dengan menggunakan provider, Anda dapat mengatur state aplikasi secara terpusat dan memperbarui tampilan secara otomatis ketika state berubah.

Selain itu, Flutter juga menyediakan widget ValueNotifier dan ChangeNotifier yang dapat digunakan untuk mengelola state lokal di dalam widget. ValueNotifier digunakan untuk menyimpan dan memperbarui nilai tunggal, sedangkan ChangeNotifier digunakan untuk menyimpan state yang kompleks dan memberikan notifikasi ketika state berubah.

Memahami konsep state di Flutter juga melibatkan pengelompokan state menjadi dua jenis: state lokal dan state global. State lokal adalah state yang terkait dengan satu widget atau subtree widget tertentu, sedangkan state global adalah state yang dapat diakses dan diubah dari mana saja dalam aplikasi. Penggunaan state lokal dan global harus dipertimbangkan sesuai dengan kebutuhan aplikasi dan kompleksitasnya.

Penting untuk diingat bahwa penggunaan state dalam Flutter harus dikelola dengan hati-hati untuk menghindari masalah seperti state yang tidak konsisten atau overhead yang berlebihan. Anda harus mempertimbangkan berbagai faktor seperti kompleksitas aplikasi, kinerja, dan pengalaman pengguna saat memilih pendekatan pengelolaan state yang tepat.

Ketika mengembangkan aplikasi Flutter, pemahaman yang baik tentang konsep state adalah kunci untuk membuat aplikasi yang responsif, dinamis, dan mudah dipelihara. Dengan menggunakan pendekatan yang tepat untuk mengelola state dan memperbarui tampilan sesuai kebutuhan, Anda dapat menciptakan pengalaman pengguna yang luar biasa dan memuaskan.

C. Memperbarui Tampilan Dengan Setstate()

Memperbarui tampilan dengan **setState()** adalah konsep kunci dalam pengembangan aplikasi Flutter yang dinamis. Dalam Flutter, **setState()** adalah metode yang digunakan untuk memberitahu framework bahwa state internal dari suatu widget telah berubah dan perlu membangun ulang tampilan widget tersebut. Ketika Anda memanggil **setState()**, Flutter akan menjadwalkan ulang pemanggilan **build()** pada widget yang bersangkutan, sehingga memperbarui tampilannya sesuai dengan perubahan state.

Penggunaan **setState()** sangat berguna ketika Anda memiliki stateful widget yang bergantung pada keadaan internal yang dapat berubah, misalnya ketika ada perubahan data yang diterima dari sumber eksternal seperti interaksi pengguna atau pembaruan data dari server. Dengan memanggil **setState()** setelah perubahan state, Anda memastikan bahwa tampilan widget diperbarui sesuai dengan keadaan yang baru.

Penting untuk dipahami bahwa setiap pemanggilan **setState()** menyebabkan seluruh widget subtree yang bersangkutan dibangun ulang. Ini termasuk widget-widget turunan dari widget yang memanggil **setState()**. Oleh karena itu, Anda harus memperhatikan kinerja dan efisiensi saat menggunakan **setState()** dalam aplikasi Flutter Anda.

Pemanggilan **setState()** harus dilakukan secara asinkron, yang berarti bahwa pemanggilan **build()** selanjutnya akan dijadwalkan untuk dijalankan di urutan berikutnya setelah metode saat ini selesai dieksekusi. Ini memastikan bahwa pembaruan tampilan tidak menghalangi eksekusi kode lain dalam aplikasi Anda.

Selain pembaruan tampilan, **setState()** juga dapat digunakan untuk memicu tindakan tambahan setelah pembaruan tampilan selesai, seperti menampilkan pesan pemberitahuan kepada pengguna atau memicu pembaruan data lainnya.

Penggunaan yang cerdas dari **setState()** sangat penting untuk menghindari membangun ulang tampilan yang tidak perlu atau

mempengaruhi kinerja aplikasi secara negatif. Anda harus memanggil **setState()** hanya ketika ada perubahan yang benar-benar mempengaruhi tampilan widget atau ketika Anda memerlukan pembaruan UI secara dinamis.

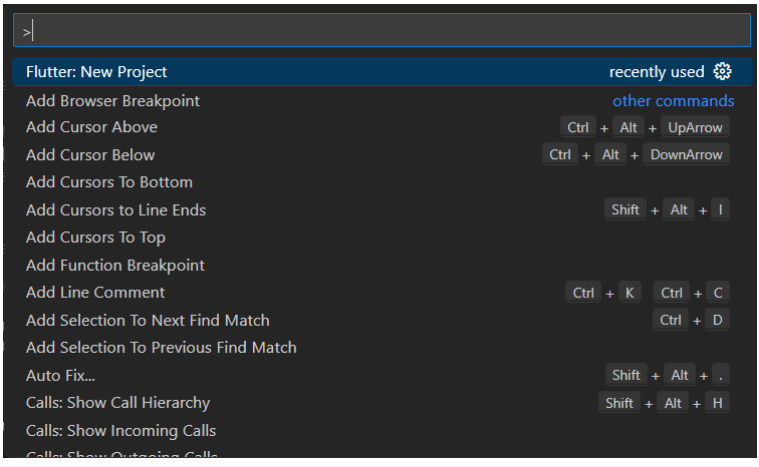
Memahami konsep **setState()** adalah kunci dalam mengembangkan aplikasi Flutter yang responsif dan dinamis. Dengan menggunakan metode ini dengan bijaksana, Anda dapat menciptakan pengalaman pengguna yang mulus dan memuaskan dalam aplikasi Anda.

Selain itu, penting juga untuk memahami bahwa setState() memungkinkan Anda untuk mengatur tampilan dengan responsif sesuai dengan perubahan state, sehingga memungkinkan Anda untuk membuat aplikasi Flutter yang interaktif dan dinamis.

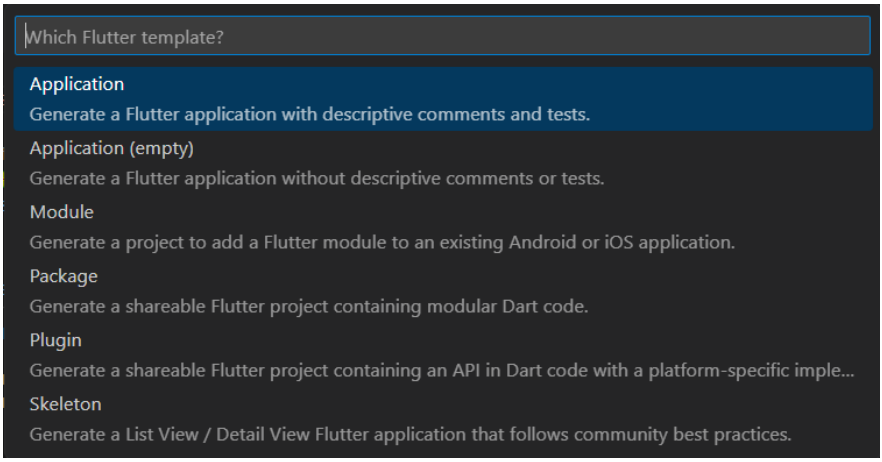
Ketika Anda menggunakan **setState()**, pastikan untuk memperbarui state dengan benar dan memastikan bahwa setiap pemanggilan **setState()** sesuai dengan perubahan yang terjadi dalam aplikasi Anda. Dengan demikian, Anda dapat memanfaatkan kekuatan Flutter untuk membuat aplikasi yang mengesankan bagi pengguna Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

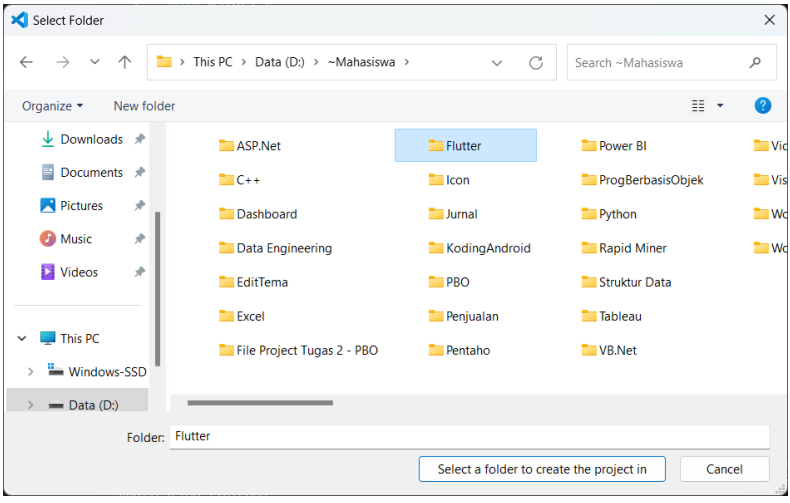


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **flutter_state** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   title: 'SetState Example',
   home: CounterPage(),
  );
 }
}

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

 @override
 // ignore: library_private_types_in_public_api
 _CounterPageState createState() => _CounterPageState();
}

class _CounterPageState extends State<CounterPage> {
 int _counter = 0;

 void _incrementCounter() {
  setState(() {
   _counter++;
  });
 }
```

```
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text('Counter Example'),
    ),
    body: Center(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
       const Text(
        'Counter:',
       ),
       Text(
        '$_counter',
        style: Theme.of(context).textTheme.headlineMedium,
       ),
      ],
     ),
    ),
    floatingActionButton: FloatingActionButton(
     onPressed: _incrementCounter,
     tooltip: 'Increment',
     child: const Icon(Icons.add),
    ),
   );
  }
}
```

Dalam contoh di atas, kita memiliki sebuah stateful widget bernama **CounterPage** yang memiliki state _counter yang bertipe integer. Ketika tombol floating action button ditekan, metode **_incrementCounter()** dipanggil yang memanggil **setState()** untuk memperbarui nilai _counter dan memicu pembaruan tampilan widget.

Ketika **setState()** dipanggil, Flutter menjadwalkan ulang pemanggilan **build()** pada widget **_CounterPageState**, sehingga tampilan widget diperbarui sesuai dengan nilai **_counter** yang baru. Sebagai hasilnya,

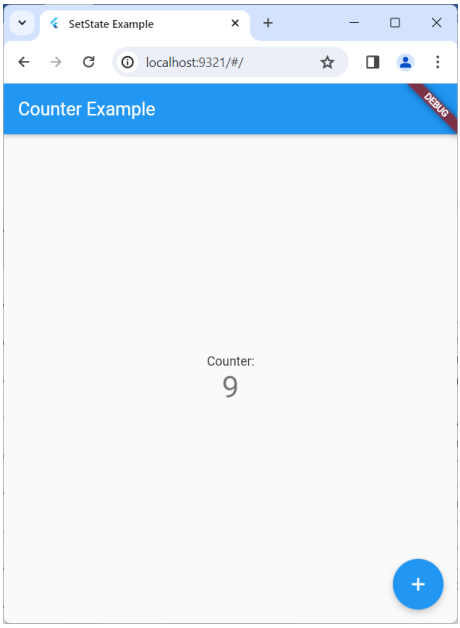
setiap kali tombol ditekan, nilai **_counter** bertambah dan tampilan widget diperbarui untuk mencerminkan perubahan tersebut.

Contoh ini mengilustrasikan bagaimana **setState()** digunakan untuk memperbarui tampilan widget secara dinamis dalam respons terhadap perubahan state, sehingga memungkinkan Anda untuk menciptakan antarmuka pengguna yang interaktif dan responsif dalam aplikasi Flutter Anda.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



D. Memproses Masukan Pengguna

Memproses masukan pengguna adalah salah satu aspek penting dalam pengembangan aplikasi Flutter yang interaktif dan responsif. Di dalam Flutter, ada berbagai cara untuk memproses masukan pengguna, seperti input teks, tap, swipe, dan banyak lagi, dan setiap jenis masukan ini dapat memicu tindakan yang berbeda dalam aplikasi Anda.

Salah satu cara yang umum digunakan untuk memproses masukan pengguna adalah melalui widget interaktif seperti tombol, field input, atau widget gesture seperti **GestureDetector**. Tombol digunakan untuk menanggapi ketika pengguna melakukan tap pada mereka, sementara field input memungkinkan pengguna memasukkan data seperti teks. Widget gesture seperti **GestureDetector** memungkinkan Anda menanggapi berbagai gerakan pengguna, seperti tap, swipe, atau drag.

Penggunaan widget seperti **InkWell**, **RaisedButton**, atau **GestureDetector** memungkinkan Anda menambahkan perilaku interaktif ke dalam aplikasi Anda dengan mudah. Anda dapat menetapkan fungsi atau metode ke properti seperti onPressed atau onTap pada widget-widget ini untuk menentukan tindakan yang akan dilakukan ketika pengguna melakukan interaksi.

Selain itu, Anda juga dapat menggunakan widget seperti **TextField** untuk memungkinkan pengguna memasukkan teks ke dalam aplikasi Anda. Anda dapat mengakses nilai yang dimasukkan oleh pengguna melalui properti seperti **onChanged** atau **onSubmit**, dan kemudian memprosesnya sesuai kebutuhan aplikasi Anda.

Flutter juga menyediakan widget khusus untuk memproses masukan pengguna yang lebih kompleks, seperti **Checkbox**, **RadioButton**, atau **Slider**. Widget-widget ini memungkinkan Anda untuk membuat antarmuka pengguna yang lebih interaktif dan dinamis, dan Anda dapat menanggapi perubahan nilai yang dihasilkan oleh pengguna dengan cara yang sesuai.

Penting untuk mempertimbangkan responsif aplikasi Anda terhadap masukan pengguna dengan benar. Ini mencakup menanggapi masukan pengguna dengan cepat dan memberikan umpan balik yang jelas kepada pengguna ketika mereka berinteraksi dengan aplikasi Anda.

Ketika memproses masukan pengguna, penting untuk memvalidasi dan membersihkan data yang dimasukkan oleh pengguna sebelum menggunakannya dalam aplikasi Anda. Ini membantu mencegah kesalahan atau masalah keamanan yang mungkin timbul akibat data yang tidak valid atau berbahaya.

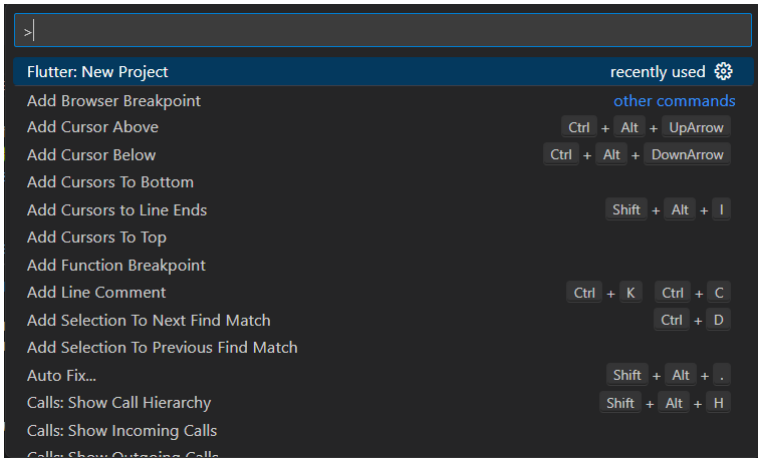
Selain itu, Anda juga dapat menggunakan animasi dan efek visual untuk meningkatkan pengalaman pengguna saat memproses masukan mereka. Misalnya, Anda dapat menggunakan animasi untuk memberikan umpan balik visual ketika pengguna menekan tombol atau memasukkan data ke dalam aplikasi Anda.

Penggunaan instrumen pengembangan seperti Widget Inspector atau Flutter DevTools dapat membantu Anda memahami dan menguji interaksi pengguna dalam aplikasi Anda. Ini memungkinkan Anda untuk melihat hierarki widget, menganalisis kinerja, dan memecahkan masalah yang mungkin timbul dalam respons terhadap masukan pengguna.

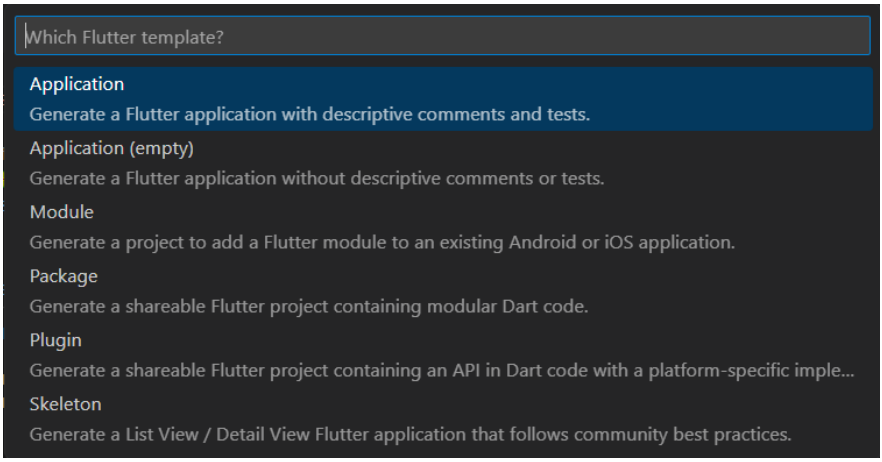
Memproses masukan pengguna dengan baik adalah kunci untuk menciptakan pengalaman pengguna yang menyenangkan dan produktif dalam aplikasi Flutter Anda. Dengan menggunakan berbagai widget dan teknik yang tersedia, Anda dapat membuat antarmuka pengguna yang interaktif dan responsif yang memenuhi kebutuhan pengguna Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

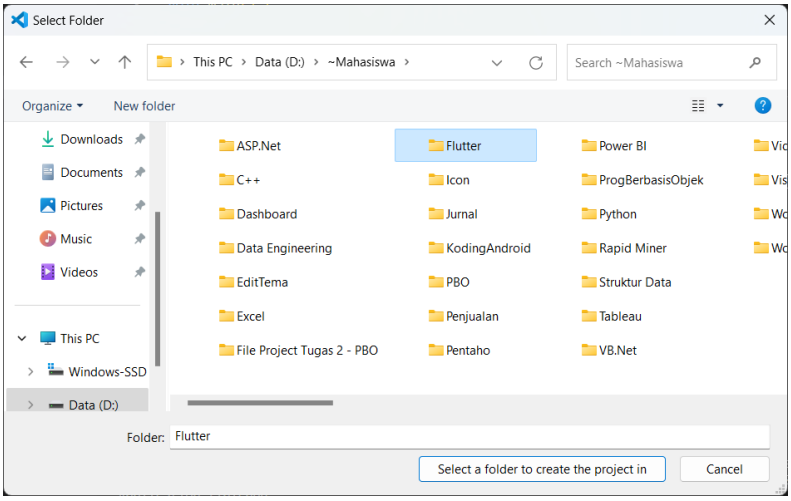


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **proses_input** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   title: 'User Input Example',
   home: UserInputPage(),
  );
 }
}
```

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

  @override
  // ignore: library_private_types_in_public_api
  _UserInputPageState createState() => _UserInputPageState();
}

class _UserInputPageState extends State<UserInputPage> {
  String _inputText = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('User Input Example'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            TextField(
              onChanged: (value) {
                setState(() {
                  _inputText = value;
                });
              },
              decoration: const InputDecoration(
                labelText: 'Enter some text',
                border: OutlineInputBorder(),
              ),
            ),
```

```
        decoration: const InputDecoration(
          labelText: 'Enter some text',
          border: OutlineInputBorder(),
        ),
      ),
```

Dalam contoh di atas, kita menggunakan widget TextField untuk memungkinkan pengguna memasukkan teks. Saat pengguna memasukkan teks, kita menggunakan properti **onChanged** untuk menetapkan fungsi yang akan dipanggil setiap kali teks berubah. Di dalam fungsi tersebut, kita memanggil **setState()** untuk memperbarui tampilan dan menetapkan nilai _inputText sesuai dengan teks yang dimasukkan oleh pengguna.

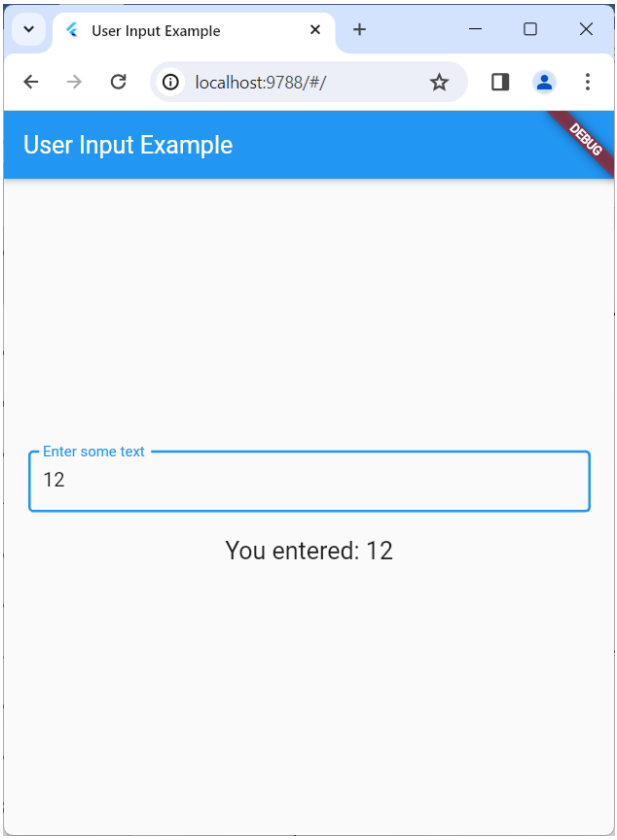
Kemudian, kita menampilkan teks yang dimasukkan oleh pengguna di bawah TextField menggunakan widget Text. Nilai **_inputText** diperbarui setiap kali pengguna memasukkan atau mengubah teks, sehingga tampilan Text juga diperbarui secara otomatis untuk mencerminkan perubahan tersebut.

Dengan menggunakan widget TextField dan properti **onChanged**, kita dapat memproses masukan pengguna dalam aplikasi kita. Anda dapat menyesuaikan logika di dalam **onChanged** sesuai dengan kebutuhan aplikasi Anda, misalnya untuk validasi data atau pemrosesan tambahan sebelum menggunakan data yang dimasukkan oleh pengguna.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



E. Menggunakan Fungsi Callback

Sebuah fungsi callback adalah fungsi yang diteruskan sebagai argumen ke fungsi lain, dan akan dipanggil oleh fungsi lain tersebut pada waktu tertentu. Dengan kata lain, Anda memberikan suatu fungsi kepada fungsi lain sebagai "callback" dan fungsi tersebut akan dipanggil kembali ketika diperlukan, seringkali sebagai respons terhadap suatu peristiwa.

Dalam Flutter, fungsi callback sering digunakan dalam berbagai konteks, seperti menanggapi interaksi pengguna, pembaruan data, atau penyelesaian operasi asinkron. Contoh umum penggunaan fungsi callback adalah dalam menanggapi aksi pengguna seperti ketika tombol ditekan atau ketika data diterima dari sumber eksternal seperti server.

Penggunaan fungsi callback memungkinkan Anda untuk membuat kode yang lebih terstruktur dan modular. Misalnya, Anda dapat memisahkan logika bisnis dari tampilan dengan menetapkan fungsi callback untuk menangani tindakan pengguna, sehingga memisahkan antara bagian UI dengan logika aplikasi.

Fungsi callback juga memungkinkan untuk membuat komponen yang dapat digunakan kembali dengan lebih baik. Anda dapat membuat widget yang menerima fungsi callback sebagai argumen, sehingga memungkinkan pengguna widget untuk menentukan tindakan yang akan dilakukan ketika suatu peristiwa terjadi.

Dengan menggunakan fungsi callback, Anda dapat membuat aplikasi Flutter yang lebih responsif dan dinamis. Misalnya, Anda dapat memberikan umpan balik langsung kepada pengguna saat mereka berinteraksi dengan aplikasi Anda, seperti menampilkan pesan pemberitahuan atau memicu animasi ketika aksi tertentu dilakukan.

Salah satu contoh penggunaan fungsi callback adalah dalam widget **GestureDetector**, di mana Anda dapat menetapkan fungsi callback untuk menanggapi berbagai gerakan pengguna seperti tap, drag, atau swipe. Ketika pengguna melakukan gerakan tertentu, fungsi callback yang sesuai akan dipanggil.

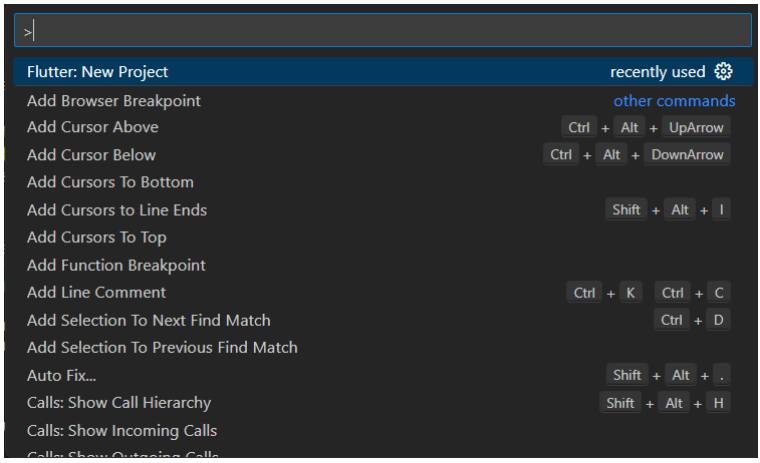
Penting untuk memperhatikan bahwa dalam Flutter, fungsi callback sering disampaikan sebagai argumen ke dalam widget atau metode lain, yang kemudian akan dipanggil oleh framework saat diperlukan. Oleh karena itu, pemahaman tentang bagaimana fungsi callback bekerja dalam konteks Flutter sangat penting untuk mengembangkan aplikasi yang efisien dan responsif.

Dalam pengembangan aplikasi Flutter yang lebih besar, penggunaan fungsi callback dapat membantu dalam memisahkan tanggung jawab dan menyederhanakan struktur kode. Ini memungkinkan pengembang untuk fokus pada satu aspek dari aplikasi pada satu waktu, dan membuat kode lebih mudah dipelihara dan diperbarui di masa mendatang.

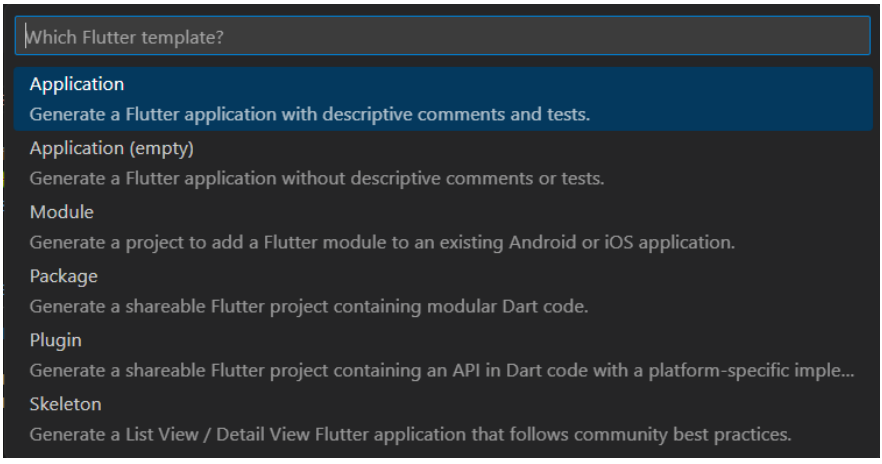
Dengan menggunakan fungsi callback secara bijaksana, Anda dapat meningkatkan fleksibilitas, modularitas, dan responsivitas aplikasi Flutter Anda. Ini memungkinkan Anda untuk membuat aplikasi yang lebih interaktif, dinamis, dan mudah diadaptasi sesuai dengan kebutuhan pengguna dan perubahan dalam aplikasi Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

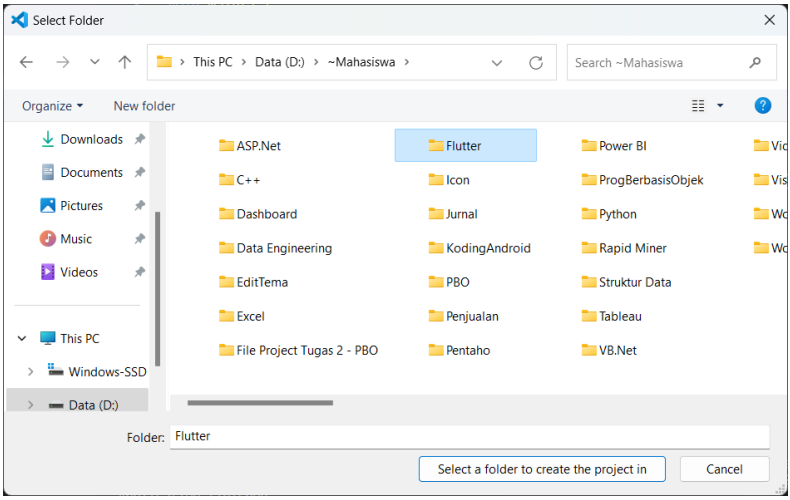


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **flutter_callback** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   title: 'Callback Example',
```

```
    home: CallbackExample(),
   );
  }
}

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

  // Membuat fungsi callback yang menerima string sebagai argumen
  void _showMessage(String message) {
   debugPrint('Message received: $message');
   // Di sini, Anda dapat menambahkan logika lain sesuai kebutuhan
aplikasi Anda
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text('Callback Example'),
    ),
    body: Center(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
       ElevatedButton(
        onPressed: () {
         // Memanggil fungsi callback saat tombol ditekan
         _showMessage('Button pressed');
        },
        child: const Text('Press Me'),
       ),
      ],
     ),
    ),
   );
  }
}
```

Dalam contoh di atas, kita memiliki aplikasi Flutter sederhana dengan satu tombol yang akan memanggil sebuah fungsi callback saat tombol ditekan.

Fungsi callback kita bernama **_showMessage**, yang menerima sebuah string sebagai argumen. Di dalam fungsi callback ini, kita hanya mencetak pesan ke konsol untuk tujuan demonstrasi, tetapi Anda dapat menambahkan logika lain sesuai dengan kebutuhan aplikasi Anda, seperti menampilkan pesan pemberitahuan atau memicu perubahan tampilan.

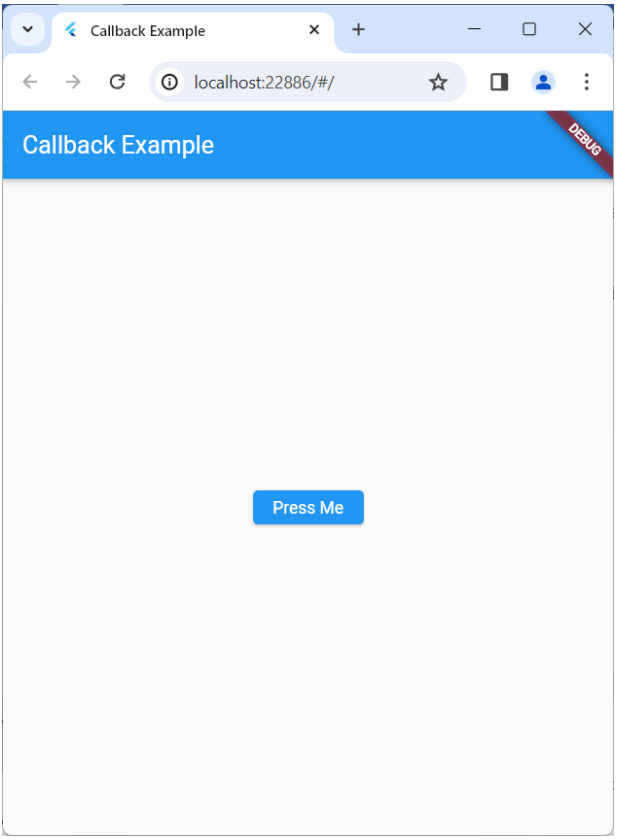
Ketika tombol ditekan, kita memanggil fungsi callback **_showMessage** dengan melewatkan string **'Button pressed'**. Ini memicu pemanggilan fungsi callback, yang kemudian akan mengeksekusi kode di dalamnya, seperti mencetak pesan ke konsol.

Dengan menggunakan fungsi callback seperti ini, Anda dapat mengintegrasikan logika pemrosesan ke dalam aplikasi Anda dengan cara yang modular dan fleksibel. Hal ini memungkinkan Anda untuk memisahkan tanggung jawab dan membuat kode Anda lebih mudah dipelihara dan diperluas di masa mendatang.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



F. Menambahkan Animasi

Dalam pengembangan aplikasi Flutter, animasi dapat diterapkan pada berbagai elemen antarmuka pengguna, mulai dari perubahan posisi, ukuran, rotasi, hingga perubahan warna dan opasitas.

Salah satu keuntungan utama menggunakan animasi dalam aplikasi Flutter adalah bahwa Flutter menyediakan berbagai widget dan library yang memudahkan penggunaan dan implementasi animasi. Ini memungkinkan pengembang untuk menciptakan animasi yang kompleks dengan relatif mudah, bahkan tanpa pengetahuan khusus tentang animasi.

Dalam Flutter, animasi dapat diterapkan dengan menggunakan widget seperti **AnimatedContainer**, **AnimatedOpacity**, atau **AnimatedBuilder**.

Widget-widget ini memungkinkan Anda untuk menganimasikan perubahan properti tertentu dari widget, seperti ukuran, posisi, atau opasitas, dengan mulus.

Selain menggunakan widget bawaan Flutter, Anda juga dapat membuat animasi kustom menggunakan **AnimationController** dan beberapa jenis animasi yang disediakan oleh Flutter, seperti **Tween**, **CurvedAnimation**, dan banyak lagi. Dengan menggunakan **AnimationController**, Anda dapat mengontrol durasi, interpolasi, dan berbagai aspek lain dari animasi Anda.

Penggunaan animasi dalam aplikasi Flutter memerlukan pemahaman tentang konsep-konsep dasar animasi, seperti durasi, interpolasi, dan kurva perpindahan. Durasi mengacu pada berapa lama animasi berlangsung, sedangkan interpolasi mengontrol bagaimana nilai-nilai antara perubahan properti dihitung. Kurva perpindahan mengontrol bagaimana animasi berubah dari satu nilai ke nilai lainnya.

Selain itu, dalam menggunakan animasi dalam Flutter, penting untuk mempertimbangkan kinerja aplikasi Anda. Animasi yang terlalu rumit atau terlalu banyak dapat mempengaruhi kinerja aplikasi, terutama pada perangkat dengan spesifikasi rendah. Oleh karena itu, Anda harus menggunakan animasi dengan bijaksana dan mempertimbangkan kinerja saat merancang aplikasi Anda.

Animasi tidak hanya berguna untuk menarik perhatian pengguna, tetapi juga dapat membantu menyampaikan informasi dengan lebih baik. Misalnya, Anda dapat menggunakan animasi untuk memberikan umpan balik visual saat pengguna berinteraksi dengan elemen antarmuka pengguna, atau untuk menyampaikan transisi antara layar dalam aplikasi Anda.

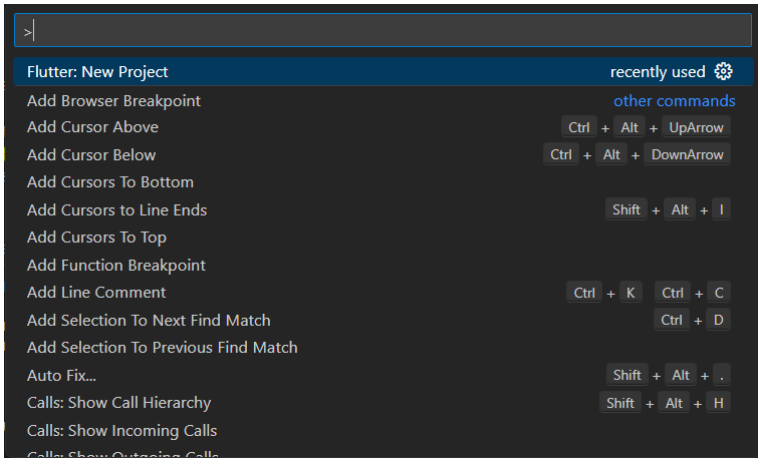
Penting juga untuk diingat bahwa animasi harus digunakan dengan bijaksana dan disesuaikan dengan konteks aplikasi Anda. Animasi yang berlebihan atau berlebihan dapat mengganggu pengalaman pengguna dan mengurangi kesan keseluruhan dari aplikasi Anda.

Seiring dengan menambahkan animasi, Anda juga harus memperhatikan antarmuka pengguna Anda secara keseluruhan. Animasi harus terintegrasi secara mulus dengan desain dan alur kerja aplikasi Anda, sehingga memberikan pengalaman pengguna yang konsisten dan intuitif.

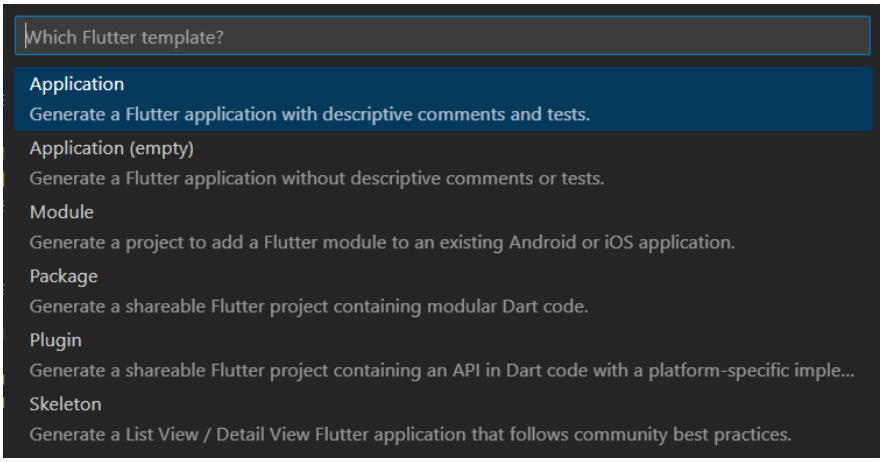
Pada akhirnya, menambahkan animasi ke dalam aplikasi Flutter adalah tentang meningkatkan interaksi dan pengalaman pengguna, serta membuat aplikasi Anda lebih menarik dan memuaskan untuk digunakan. Dengan menggunakan animasi dengan bijaksana dan memperhatikan prinsip desain yang baik, Anda dapat menciptakan aplikasi yang lebih menarik dan berkesan bagi pengguna Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

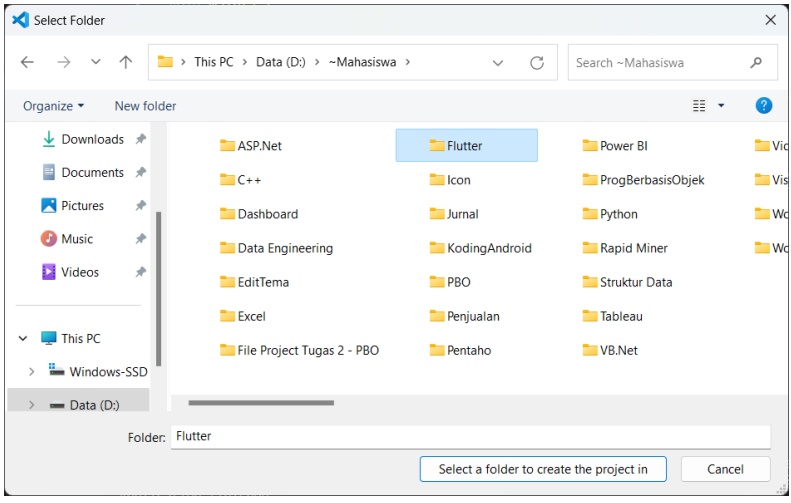


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **flutter_animasi** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   title: 'Animation Example',
   home: AnimationExample(),
  );
 }
}

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

 @override
 // ignore: library_private_types_in_public_api
 _AnimationExampleState createState() => _AnimationExampleState();
}

class _AnimationExampleState extends State<AnimationExample> {
 Color _backgroundColor = Colors.blue;

 void _changeColor() {
  setState(() {
   // Ubah warna latar belakang secara acak
   _backgroundColor = Colors.pink;
  });
 }
```

```
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text('Animation Example'),
    ),
    body: Center(
     child: AnimatedContainer(
      duration: const Duration(seconds: 1),
      curve: Curves.easeInOut,
      color: _backgroundColor,
      width: 200,
      height: 200,
      child: const Center(
       child: Text(
        'Tap me!',
        style: TextStyle(
         fontSize: 20,
         fontWeight: FontWeight.bold,
         color: Colors.white,
        ),
       ),
      ),
     ),
    ),
    floatingActionButton: FloatingActionButton(
     onPressed: _changeColor,
     child: const Icon(Icons.color_lens),
    ),
   );
  }
}
```

Dalam contoh di atas, kita memiliki sebuah aplikasi Flutter sederhana yang menampilkan sebuah AnimatedContainer di tengah layar. Ketika tombol ditekan, warna latar belakang AnimatedContainer akan berubah secara mulus dari warna biru menjadi warna merah muda, dengan durasi animasi 1 detik.

Perhatikan penggunaan properti duration dan curve pada **AnimatedContainer**. Properti **duration** menentukan berapa lama animasi akan berlangsung, sementara properti **curve** mengontrol interpolasi animasi, dalam hal ini menggunakan **Curves.easeInOut** untuk memberikan efek perpindahan yang halus.

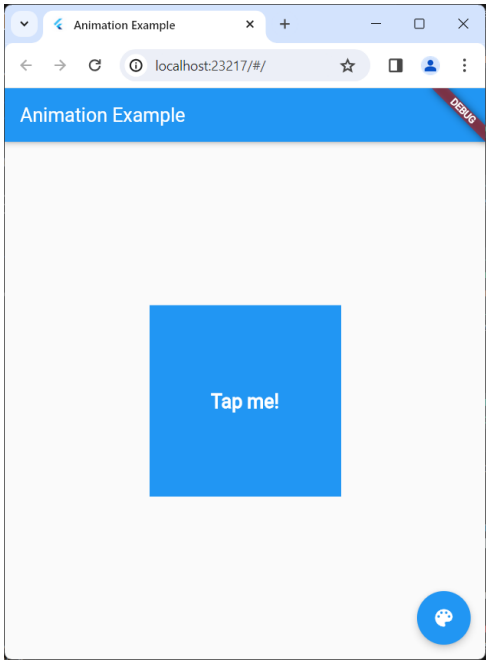
Ketika tombol floating action button ditekan, fungsi **_changeColor()** akan dipanggil, yang mengubah nilai warna latar belakang dan memicu pembaruan tampilan, sehingga memicu animasi perubahan warna.

Dengan menggunakan **AnimatedContainer** dan properti-propertinya, Anda dapat dengan mudah menambahkan animasi ke dalam aplikasi Anda, meningkatkan interaksi pengguna dan memberikan pengalaman yang lebih menarik dan dinamis.

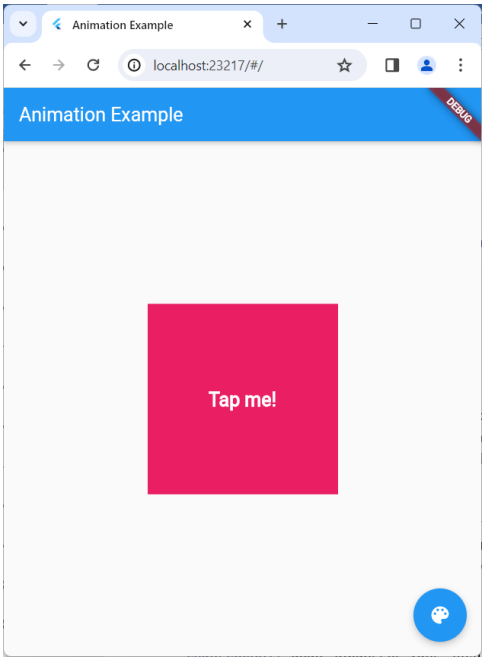
Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

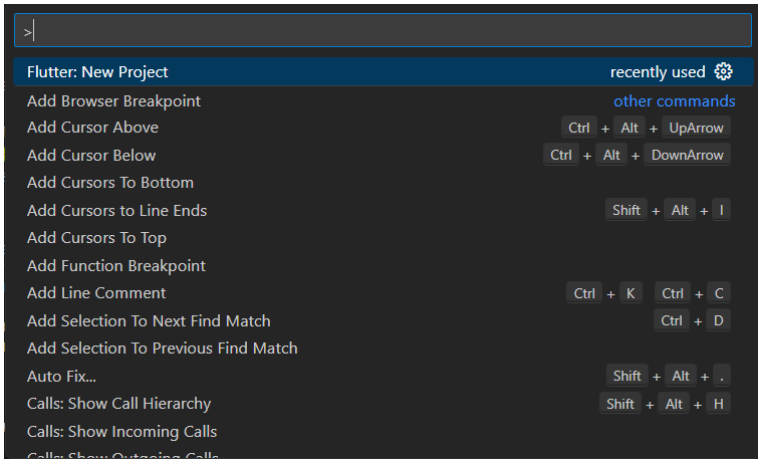


Ketika tombol ditekan, maka warnanya akan berubah.

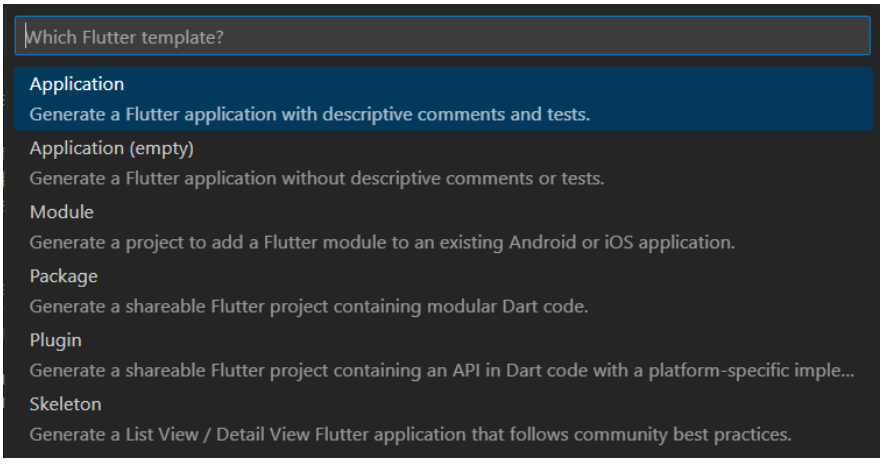


G. Contoh Koding: Membuat Tampilan Dinamis Untuk Aplikasi Flutter

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

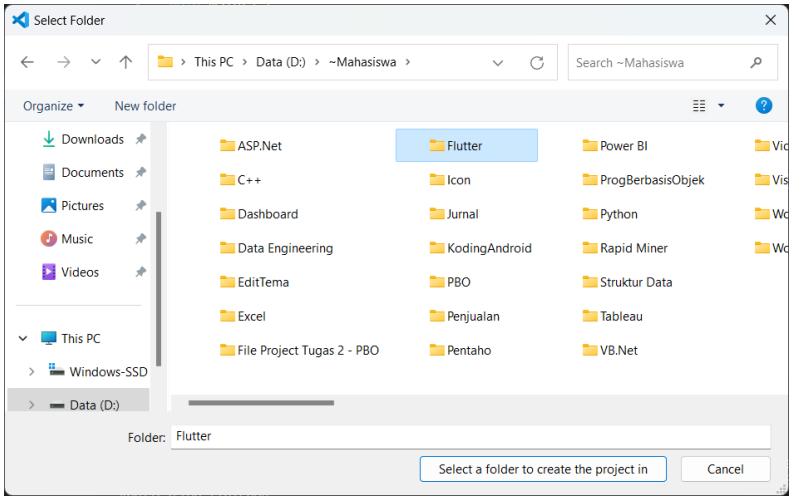


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **flutter_dinamis** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   title: 'Dynamic List Example',
   home: DynamicListExample(),
  );
 }
```

```
}

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

  @override
  DynamicListExampleState createState() =>
DynamicListExampleState();
}

class DynamicListExampleState extends State<DynamicListExample> {
  List<String> items = ['Apple', 'Banana', 'Orange'];
  TextEditingController textEditingController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Dynamic List Example'),
      ),
      body: Column(
        children: <Widget>[
          Expanded(
            child: ListView.builder(
              itemCount: items.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(items[index]),
                );
              },
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
```

```
          children: <Widget>[
            Expanded(
              child: TextField(
                controller: textEditingController,
                decoration: const InputDecoration(
                  labelText: 'Add New Item',
                ),
              ),
            ),
            IconButton(
              icon: const Icon(Icons.add),
              onPressed: () {
                setState(() {
                  items.add(textEditingController.text);
                  textEditingController.clear();
                });
              },
            ),
          ],
        ),
      ),
    ],
  ),
 );
 }
}
```

Dalam contoh di atas, kita membuat aplikasi yang menampilkan daftar item dinamis. Pengguna dapat menambahkan item baru ke dalam daftar dengan menggunakan **TextField** dan tombol tambah.

Kelas **DynamicListExampleState** memiliki sebuah list items yang berisi daftar item dan sebuah **TextEditingController** untuk mengontrol input dari **TextField**.

Di dalam metode **build()**, kita menggunakan **ListView.builder** untuk menampilkan daftar item dalam bentuk list. Setiap item dirender sebagai **ListTile**.

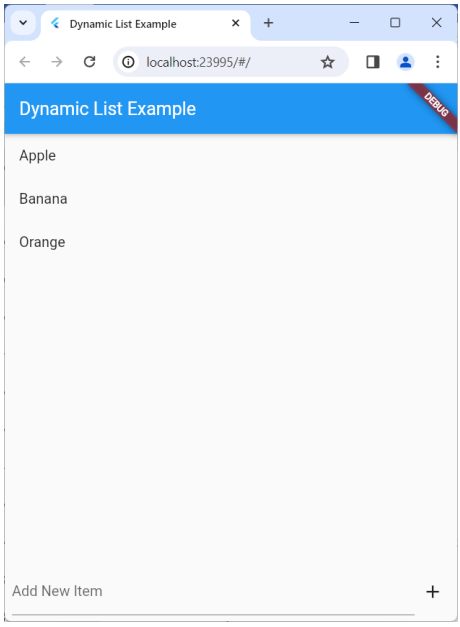
Pada bagian bawah tampilan, kita menempatkan **TextField** yang memungkinkan pengguna untuk memasukkan item baru. Ketika tombol tambah ditekan, kita menambahkan item baru ke dalam list menggunakan **setState()** untuk memperbarui tampilan.

Dengan demikian, pengguna dapat dengan mudah menambahkan item baru ke dalam daftar secara dinamis. Semoga contoh ini membantu Anda memahami cara membuat tampilan dinamis dengan Flutter.

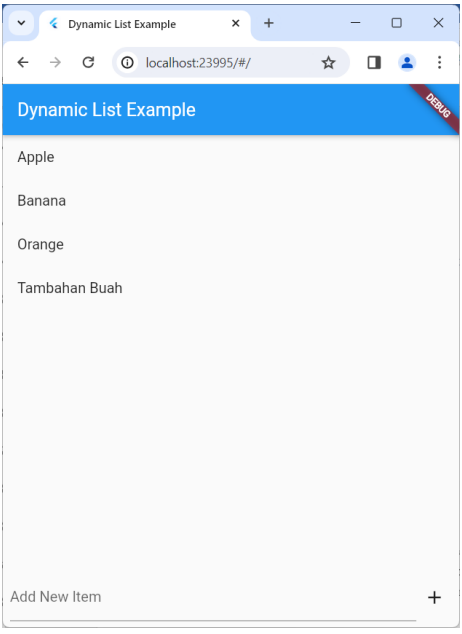
Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



Ketika kolom "Add New Item" diketikkan "Tambahan Buah" dan ditekan tombol "+", maka hasilnya akan seperti di bawah ini.



H. Evaluasi/Soal Latihan

Apa perbedaan antara stateless widget dan stateful widget dalam Flutter?

BAB 7
ROUTING DAN NAVIGASI

A. Tujuan Pembelajaran

Mengetahui konsep dasar routing dan navigasi dalam Flutter untuk mengelola perpindahan antar layar.

B. Apa Itu Routing Dan Navigasi?

Routing dan navigasi adalah konsep penting dalam pengembangan aplikasi Flutter yang berkaitan dengan pengelolaan perpindahan antar layar atau halaman dalam aplikasi. Dalam konteks Flutter, routing merujuk pada kemampuan untuk menentukan dan mengatur navigasi antara berbagai bagian dari aplikasi, sementara navigasi adalah proses perpindahan atau transisi antara layar yang berbeda dalam aplikasi.

Pada dasarnya, routing dan navigasi memungkinkan pengguna untuk berpindah antara berbagai layar dalam aplikasi, seperti halaman utama, detail item, formulir, atau layar pengaturan. Ini memungkinkan aplikasi untuk memiliki struktur yang terorganisir dan intuitif, serta memberikan pengalaman pengguna yang lebih baik.

Di Flutter, pengelolaan routing dan navigasi dilakukan menggunakan widget **Navigator**. **Navigator** adalah widget yang bertanggung jawab atas pengelolaan tumpukan layar atau halaman dalam aplikasi. Dengan **Navigator**, Anda dapat menambahkan, menghapus, atau mengganti layar dengan mudah.

Ada beberapa jenis navigasi yang umum digunakan dalam Flutter, termasuk navigasi push, pop, dan replace. Navigasi push digunakan untuk menambahkan layar baru ke dalam tumpukan navigasi, sementara navigasi pop digunakan untuk menghapus layar teratas dari tumpukan.

Navigasi replace digunakan untuk mengganti layar saat ini dengan layar baru tanpa menambahkan layar baru ke dalam tumpukan.

Routing dan navigasi memungkinkan pengembang untuk mengatur alur kerja aplikasi dan mengontrol bagaimana pengguna berinteraksi dengan aplikasi tersebut. Ini penting untuk menciptakan aplikasi yang mudah digunakan dan intuitif, serta memberikan pengalaman yang konsisten kepada pengguna.

Dalam aplikasi Flutter, Anda dapat menentukan rute atau rute navigasi menggunakan objek **Route**. Objek **Route** mewakili sebuah layar atau halaman dalam aplikasi, dan dapat berisi widget yang sesuai dengan konten yang ingin ditampilkan.

Konsep routing dan navigasi sangat penting dalam pengembangan aplikasi Flutter yang kompleks, terutama ketika Anda memiliki banyak layar atau alur kerja yang berbeda dalam aplikasi Anda. Dengan menggunakan routing dan navigasi yang baik, Anda dapat memperbaiki struktur aplikasi Anda dan memberikan pengalaman pengguna yang lebih baik secara keseluruhan.

Selain itu, routing dan navigasi juga memungkinkan Anda untuk melakukan navigasi dengan parameter, yang memungkinkan Anda untuk memindahkan data antara layar dalam aplikasi dengan mudah. Ini berguna ketika Anda perlu mengirim data dari satu layar ke layar lainnya, misalnya saat melakukan navigasi ke layar detail item dengan ID tertentu.

Dalam pengembangan aplikasi Flutter, Anda dapat mengimplementasikan routing dan navigasi menggunakan widget seperti **MaterialApp**, **Navigator**, dan **Route**. Flutter menyediakan berbagai metode dan widget yang memudahkan pengelolaan navigasi dan pengaturan rute dalam aplikasi Anda.

Penting untuk memahami konsep routing dan navigasi dalam Flutter agar Anda dapat merancang dan mengembangkan aplikasi yang lebih kompleks dan fungsional. Dengan menguasai konsep ini, Anda dapat meningkatkan pengalaman pengguna dalam aplikasi Flutter Anda dan membuatnya lebih efisien dan mudah digunakan.

C. Menggunakan Navigator Widget

Penggunaan widget Navigator adalah salah satu aspek penting dalam pengelolaan navigasi dan routing dalam aplikasi Flutter. Navigator adalah widget yang bertanggung jawab atas pengelolaan tumpukan layar atau halaman dalam aplikasi. Dengan Navigator, Anda dapat menambahkan, menghapus, atau mengganti layar dengan mudah, sehingga memberikan pengalaman navigasi yang baik kepada pengguna.

Navigator bekerja dengan menggunakan konsep tumpukan, di mana setiap layar baru yang ditambahkan akan ditumpuk di atas layar yang sudah ada. Ketika pengguna berpindah antar layar, Navigator akan menyesuaikan tumpukan layar sesuai dengan navigasi yang dilakukan, seperti menambah atau menghapus layar dari tumpukan.

Salah satu penggunaan umum dari Navigator adalah untuk melakukan navigasi antara layar-layar yang berbeda dalam aplikasi. Anda dapat menggunakan Navigator untuk menambahkan layar baru ke dalam tumpukan, menghapus layar yang ada, atau mengganti layar yang sedang ditampilkan dengan layar yang baru.

Navigator menggunakan objek **Route** untuk merepresentasikan setiap layar dalam aplikasi. Setiap **Route** memiliki widget yang bertanggung jawab atas tampilan konten dari layar tersebut. Anda dapat membuat objek **Route** baru untuk setiap layar dalam aplikasi dan menentukan bagaimana layar tersebut ditampilkan.

Dalam penggunaan Navigator, penting untuk memahami metode dan fungsi yang disediakan oleh widget ini. Anda dapat menggunakan metode seperti push, pop, dan replace untuk melakukan navigasi antara layar-layar yang berbeda. Metode push digunakan untuk menambahkan layar baru ke dalam tumpukan, sedangkan metode pop digunakan untuk menghapus layar teratas dari tumpukan.

Navigator juga menyediakan kemampuan untuk melakukan navigasi dengan parameter, yang memungkinkan Anda untuk mengirim data antara layar-layar dalam aplikasi. Dengan menggunakan parameter, Anda

dapat mengirim informasi atau data dari satu layar ke layar lainnya, misalnya saat melakukan navigasi ke layar detail item dengan ID tertentu.

Selain itu, Navigator juga mendukung animasi transisi antara layar-layar yang berbeda. Anda dapat menentukan animasi yang ingin digunakan saat melakukan navigasi, seperti animasi slide, fade, atau scale, untuk memberikan pengalaman navigasi yang lebih menarik kepada pengguna.

Navigator juga memungkinkan Anda untuk menangani peristiwa navigasi, seperti saat pengguna menekan tombol kembali di perangkat mereka. Anda dapat menentukan tindakan apa yang harus diambil saat peristiwa ini terjadi, misalnya dengan melakukan navigasi kembali ke layar sebelumnya atau menutup aplikasi.

Penting untuk diingat bahwa setiap aplikasi Flutter memiliki satu Navigator yang terpasang secara global. Ini berarti Anda hanya perlu menggunakan satu instance Navigator dalam aplikasi Anda, yang akan menangani semua navigasi dalam aplikasi.

Dengan memahami penggunaan Navigator widget dalam Flutter, Anda dapat merancang dan mengembangkan aplikasi yang memiliki navigasi yang baik dan pengalaman pengguna yang lebih baik secara keseluruhan. Ini memungkinkan Anda untuk membuat aplikasi yang lebih fungsional, intuitif, dan mudah digunakan oleh pengguna.

D. Mengirim Data Antar Halaman

Mengirim data antar halaman merupakan salah satu aspek penting dalam pengembangan aplikasi Flutter yang melibatkan navigasi dan komunikasi antar komponen. Dalam konteks Flutter, hal ini mengacu pada kemampuan untuk mentransfer informasi atau data dari satu halaman ke halaman lain dalam aplikasi.

Ada beberapa cara untuk mengirim data antar halaman dalam Flutter. Salah satu cara yang umum adalah dengan menggunakan parameter pada rute navigasi. Ketika Anda melakukan navigasi dari satu halaman ke halaman lain, Anda dapat menyertakan parameter yang berisi data yang ingin Anda kirimkan.

Misalnya, jika Anda ingin mengirimkan informasi tentang item yang dipilih dari halaman A ke halaman B, Anda dapat menyertakan informasi tersebut sebagai parameter saat melakukan navigasi dari halaman A ke halaman B. Hal ini memungkinkan halaman B untuk mengakses dan menggunakan informasi tersebut.

Selain menggunakan parameter pada rute navigasi, Anda juga dapat menggunakan singleton atau penyimpanan global untuk menyimpan data yang dapat diakses dari berbagai halaman dalam aplikasi. Dengan menggunakan pendekatan ini, Anda dapat menyimpan data di lokasi yang dapat diakses secara global dan mengaksesnya dari halaman mana pun dalam aplikasi.

Salah satu metode yang umum digunakan untuk menyimpan data secara global dalam Flutter adalah dengan menggunakan paket **provider** atau **GetX**. Dengan menggunakan paket ini, Anda dapat membuat model atau state management yang dapat diakses dari berbagai halaman dalam aplikasi, sehingga memungkinkan Anda untuk berbagi data dengan mudah antar halaman.

Selain menggunakan parameter pada rute navigasi dan penyimpanan global, Anda juga dapat menggunakan widget **ModalRoute** untuk mengakses data yang dikirimkan dari halaman sebelumnya. Widget **ModalRoute** memungkinkan Anda untuk mengakses data yang dikirimkan sebagai parameter saat inisialisasi halaman.

Penting untuk mempertimbangkan jenis data yang ingin Anda kirim antar halaman dalam aplikasi Anda. Jika data yang ingin Anda kirimkan sederhana, seperti string atau angka, maka penggunaan parameter pada rute navigasi mungkin sudah cukup. Namun, jika data yang ingin Anda kirimkan kompleks atau besar, Anda mungkin perlu menggunakan pendekatan yang lebih kompleks, seperti penyimpanan global atau state management.

Selain itu, penting juga untuk memperhatikan keamanan dan keandalan data yang dikirim antar halaman. Pastikan data yang dikirim tidak sensitif dan telah divalidasi dengan benar sebelum digunakan. Selain

itu, pastikan untuk menangani situasi di mana data yang diterima mungkin tidak sesuai dengan yang diharapkan.

Dalam pengembangan aplikasi Flutter, kemampuan untuk mengirim data antar halaman adalah keterampilan yang penting untuk dimiliki. Dengan memahami berbagai cara untuk melakukan ini, Anda dapat merancang dan mengembangkan aplikasi yang lebih fungsional dan efisien, serta memberikan pengalaman pengguna yang lebih baik secara keseluruhan.

E. Memodifikasi Tampilan Halaman

Salah satu cara untuk memodifikasi tampilan halaman adalah dengan menggunakan berbagai widget yang disediakan oleh Flutter. Flutter menyediakan berbagai macam widget yang dapat digunakan untuk membangun antarmuka pengguna yang kreatif dan menarik. Anda dapat memilih widget yang sesuai dengan kebutuhan tampilan halaman Anda, seperti **Container**, **Row**, **Column**, **Stack**, dan banyak lagi.

Selain menggunakan widget bawaan Flutter, Anda juga dapat membuat widget kustom atau mengubah widget yang sudah ada untuk mencapai tampilan yang diinginkan. Dengan membuat widget kustom, Anda dapat mengontrol dengan lebih detail tampilan dan perilaku widget sesuai dengan kebutuhan aplikasi Anda.

Penggunaan properti widget adalah cara lain untuk memodifikasi tampilan halaman dalam Flutter. Properti seperti **color**, **padding**, **margin**, **alignment**, **font**, dan sebagainya dapat digunakan untuk mengatur penampilan dan tata letak widget dengan lebih spesifik. Dengan mengubah nilai properti ini, Anda dapat mencapai tampilan yang sesuai dengan desain yang diinginkan.

Selain mengubah properti widget, Anda juga dapat menggunakan widget decorator seperti **BoxDecoration**, **Border**, **BorderRadius**, dan lain-lain untuk menambahkan efek visual atau dekorasi ke widget. Dengan menggunakan decorator, Anda dapat menambahkan bayangan, border,

gradient, dan lain-lain ke widget untuk meningkatkan estetika tampilan halaman Anda.

Flutter juga menyediakan widget animasi seperti **AnimatedContainer**, **AnimatedOpacity**, **AnimatedPositioned**, dan lain-lain yang memungkinkan Anda untuk mengubah tampilan halaman secara dinamis dengan animasi. Dengan menggunakan widget animasi, Anda dapat menambahkan efek visual yang menarik dan dinamis ke dalam tampilan halaman Anda, meningkatkan pengalaman pengguna.

Selain menggunakan widget dan properti, Anda juga dapat menggunakan media queries untuk memodifikasi tampilan halaman sesuai dengan ukuran layar perangkat pengguna. Media queries memungkinkan Anda untuk mendeteksi ukuran layar perangkat dan menyesuaikan tampilan halaman secara dinamis sesuai dengan ukuran layar yang tersedia.

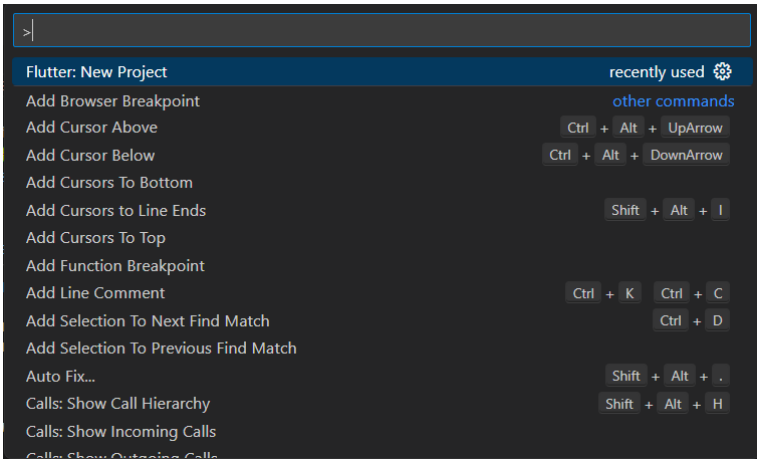
Penggunaan tema (theme) juga dapat mempengaruhi tampilan halaman secara keseluruhan dalam aplikasi Flutter. Dengan menggunakan tema, Anda dapat mengatur gaya dan tata letak standar untuk berbagai komponen UI dalam aplikasi Anda, seperti warna, font, padding, margin, dan lain-lain.

Selain itu, Flutter juga menyediakan kemampuan untuk menggunakan widget **LayoutBuilder** untuk membuat tata letak yang responsif. Dengan menggunakan **LayoutBuilder**, Anda dapat menyesuaikan tampilan halaman berdasarkan ukuran dan orientasi layar perangkat pengguna.

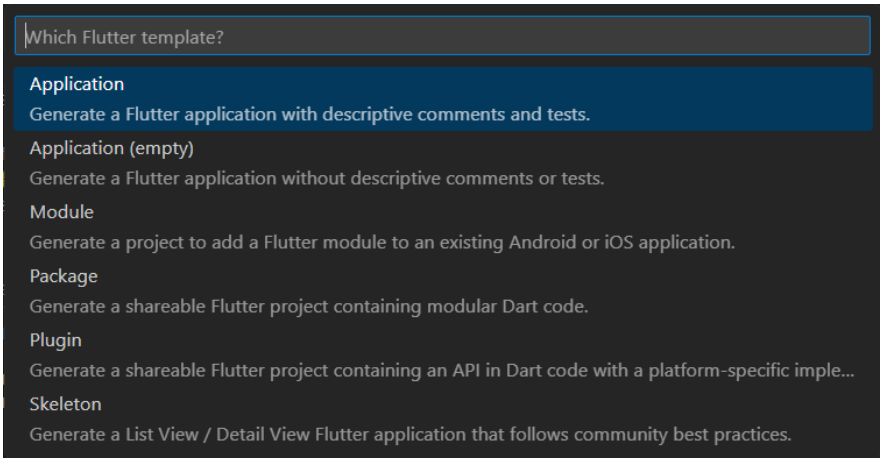
Terakhir, memodifikasi tampilan halaman juga melibatkan pemilihan dan pengaturan widget yang tepat untuk mencapai tampilan yang sesuai dengan desain aplikasi Anda. Anda perlu mempertimbangkan kebutuhan fungsional dan estetika aplikasi Anda serta preferensi pengguna dalam memodifikasi tampilan halaman.

F. Contoh Koding: Membuat Navigasi Antar Halaman

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

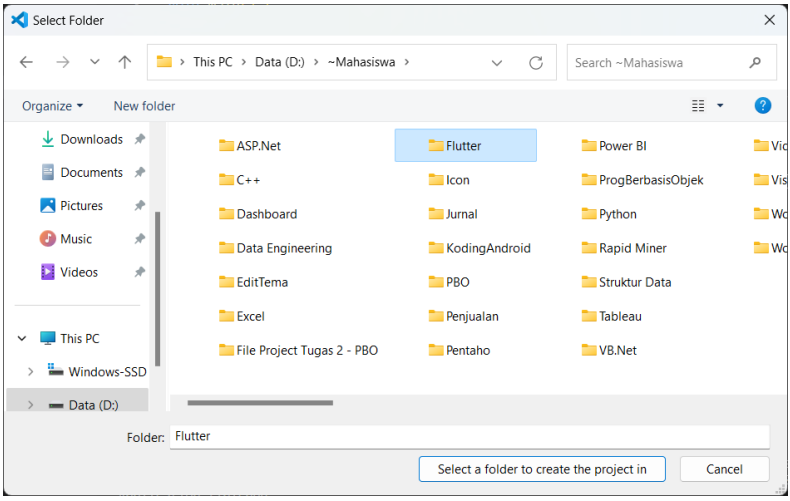


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **navigasi_sederhana** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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: 'Contoh Navigasi (Pindah Halaman)',
      initialRoute: '/',
      routes: {
        '/': (context) => const FirstPage(),
        '/second': (context) => const SecondPage(),
```

```
    },
   );
  }
}

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

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Halaman Pertama'),
   ),
   body: Center(
    child: ElevatedButton(
     onPressed: () {
      Navigator.pushNamed(context, '/second');
     },
     child: const Text('Pergi ke Halaman Kedua'),
    ),
   ),
  );
 }
}

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

```
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text('Halaman Kedua'),
    ),
    body: Center(
     child: ElevatedButton(
      onPressed: () {
       Navigator.pop(context);
      },
      child: const Text('Kembali'),
     ),
    ),
   );
  }
}
```

Dalam contoh ini, kita memiliki dua halaman: **FirstPage** dan **SecondPage**. Kita menggunakan **MaterialApp** sebagai root widget aplikasi, dengan menentukan halaman awal (**initialRoute**) dan daftar rute (**routes**) yang akan diikuti.

Pada halaman pertama (**FirstPage**), kita memiliki tombol yang ketika ditekan akan memicu navigasi ke halaman kedua (**SecondPage**) menggunakan **Navigator.pushNamed**(context, '/second').

Pada halaman kedua (**SecondPage**), kita memiliki tombol "Go Back" yang akan memicu navigasi kembali ke halaman pertama menggunakan **Navigator.pop(context)**.

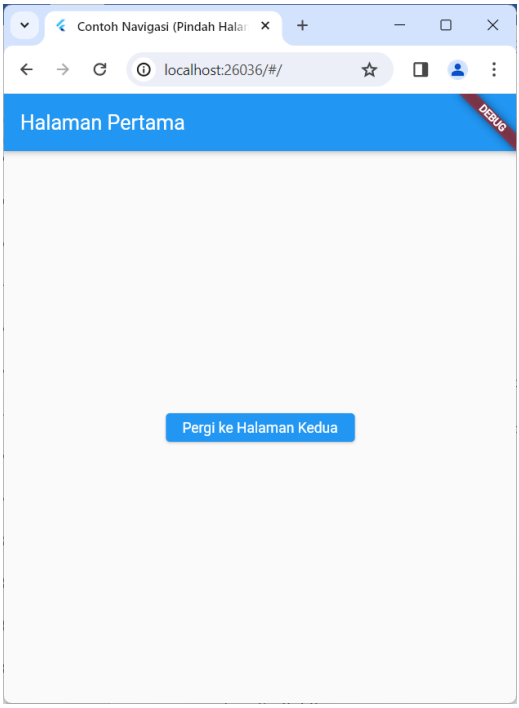
Dengan menggunakan rute dan Navigator, kita dapat dengan mudah membuat navigasi antar halaman dalam aplikasi Flutter.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah

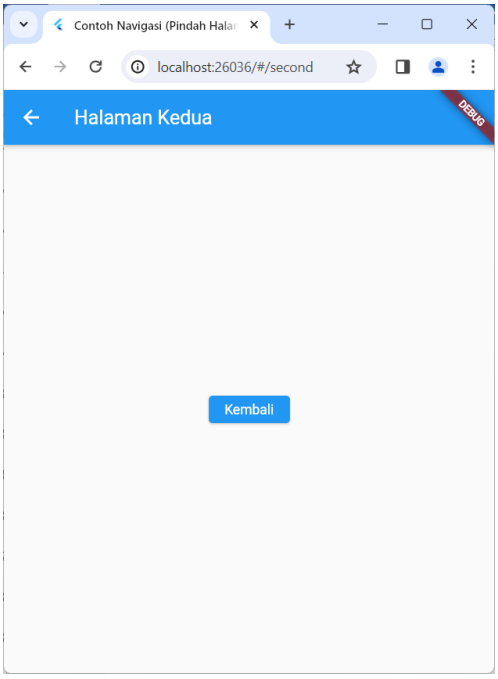
emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

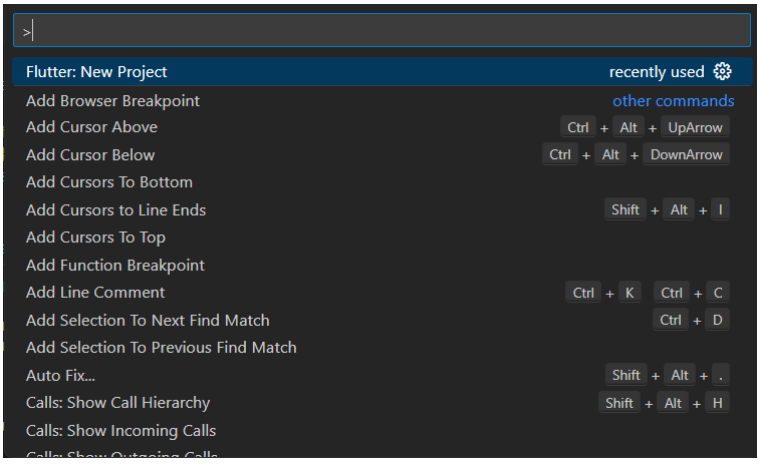


Ketika tombol “Pergi ke Halam Kedua” di klik, maka akan muncul tampilan berikut.

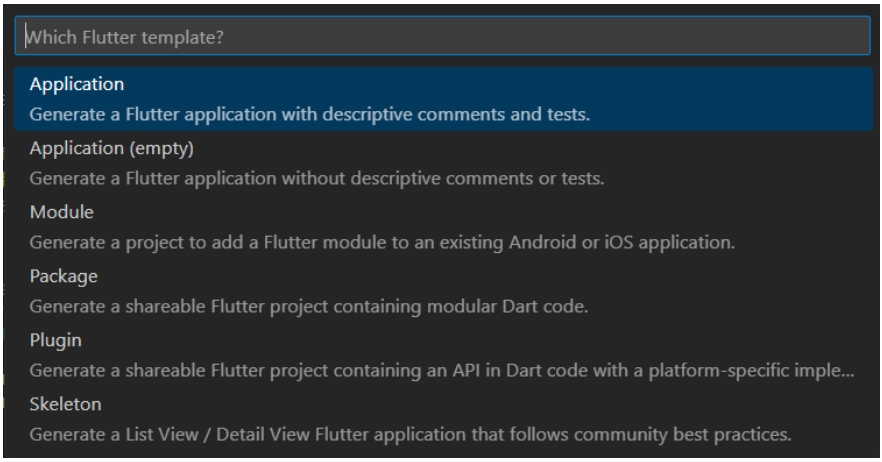


Contoh koding navigasi (lanjutan)

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

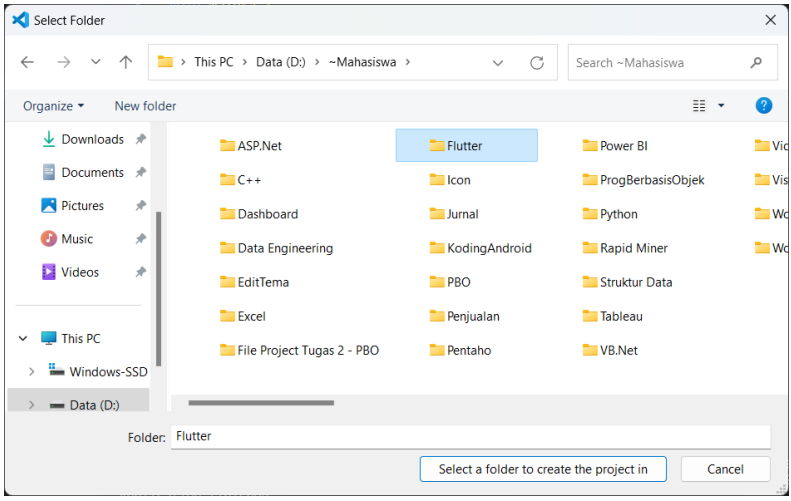


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **navigasi_lanjutan** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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: 'Contoh Navigasi',
      initialRoute: '/',
      routes: {
        '/': (context) => const HomePage(),
        '/pantai': (context) =>
            const MenuPage(menuTitle: 'Pantai', icon: Icons.beach_access),
        '/gunung': (context) =>
            const MenuPage(menuTitle: 'Gunung', icon: Icons.terrain),
        '/budaya': (context) =>
            const MenuPage(menuTitle: 'Budaya', icon: Icons.museum),
        '/kuliner': (context) =>
            const MenuPage(menuTitle: 'Kuliner', icon: Icons.restaurant),
      },
    );
  }
}
```

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

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Beranda'),
   ),
   body: GridView.count(
    crossAxisCount: 2,
    crossAxisSpacing: 20.0,
    mainAxisSpacing: 20.0,
    padding: const EdgeInsets.all(20.0),
    children: [
     ElevatedButton.icon(
      onPressed: () {
       Navigator.pushNamed(context, '/pantai');
      },
      icon: const Icon(Icons.beach_access),
      label: const Text('Pantai'),
     ),
     ElevatedButton.icon(
      onPressed: () {
       Navigator.pushNamed(context, '/gunung');
      },
      icon: const Icon(Icons.terrain),
      label: const Text('Gunung'),
     ),
     ElevatedButton.icon(
      onPressed: () {
       Navigator.pushNamed(context, '/budaya');
      },
```

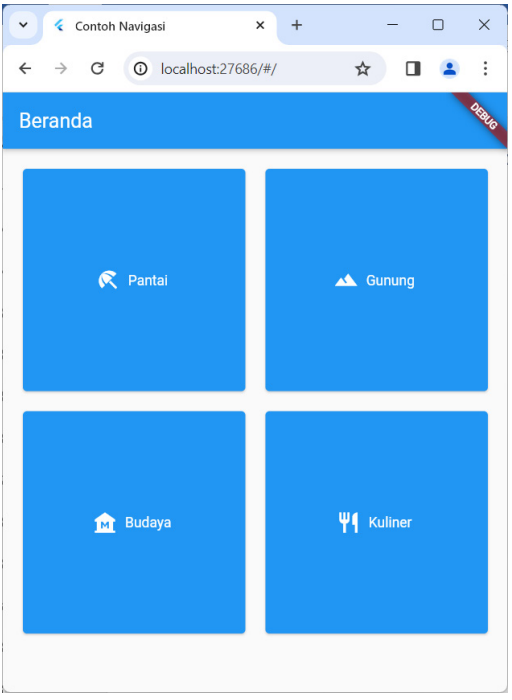
```
          icon: const Icon(Icons.museum),
          label: const Text('Budaya'),
        ),
        ElevatedButton.icon(
          onPressed: () {
            Navigator.pushNamed(context, '/kuliner');
          },
          icon: const Icon(Icons.restaurant),
          label: const Text('Kuliner'),
        ),
      ],
    ),
  );
 }
}

class MenuPage extends StatelessWidget {
 final String menuTitle;
 final IconData icon;

 const MenuPage({super.key, required this.menuTitle, required
this.icon});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text(menuTitle),
   ),
   body: Center(
    child: ElevatedButton.icon(
     onPressed: () {
      Navigator.pop(context);
     },
     icon: Icon(icon),
     label: const Text('Kembali'),
    ),
   ),
  );
 }
}
```

Dalam contoh di atas, kita menggunakan **GridView.count** untuk menampilkan menu dalam format 2 kolom dan 2 baris. Properti **crossAxisCount** digunakan untuk menentukan jumlah kolom (dalam hal ini 2), sedangkan **crossAxisSpacing** dan **mainAxisSpacing** digunakan untuk menentukan jarak antara masing-masing item dalam grid.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



G. Evaluasi/Soal Latihan

Jelaskan konsep dasar routing dan navigasi dalam pembuatan aplikasi Flutter!

BAB 8

MENGGUNAKAN PAKET DAN PLUGIN

A. Tujuan Pembelajaran

1. Memahami perbedaan antara paket dan plugin dalam konteks pengembangan aplikasi Flutter.
2. Menguasai cara menggunakan paket Flutter untuk menambahkan fungsionalitas tambahan ke aplikasi.
3. Belajar tentang plugin Flutter dan bagaimana cara mengintegrasikannya ke dalam proyek Flutter.
4. Mengetahui cara mencari, menginstal, dan mengelola paket dan plugin Flutter menggunakan pubspec.yaml dan Flutter pub command.

B. Menambahkan Paket Ke Proyek

Untuk menambahkan paket ke proyek Flutter, Anda perlu mengedit file pubspec.yaml proyek Anda. Berikut langkah-langkahnya:

1. Buka proyek Flutter Anda menggunakan editor kode favorit Anda.
2. Buka file pubspec.yaml.
3. Temukan bagian dependencies. Di sini Anda akan menambahkan paket-paket yang ingin Anda tambahkan ke proyek Anda. Paket-paket ini dituliskan di bawah dependencies dengan format nama_paket: versi.
4. Anda dapat menambahkan paket-paket dari pub.dev, yaitu repositori paket resmi Flutter. Cari paket yang ingin Anda tambahkan dan salin nama paket serta versi yang diinginkan.
5. Setelah menambahkan paket yang diinginkan, simpan file pubspec.yaml.
6. Kembali ke terminal atau command prompt, dan jalankan perintah flutter pub get. Perintah ini akan mengunduh dan menginstal paket-paket yang telah Anda tambahkan ke proyek Anda.

Setelah langkah-langkah di atas, paket yang Anda tambahkan sudah siap untuk digunakan dalam proyek Flutter Anda. Anda dapat mengimpor paket tersebut ke dalam file-file proyek Anda sesuai kebutuhan. Pastikan untuk mengecek dokumentasi paket tersebut untuk memahami cara penggunaannya.

C. Menggunakan Plugin Untuk Memperluas Fitur

Menggunakan plugin adalah cara yang sangat berguna untuk memperluas fitur dan fungsionalitas aplikasi Flutter Anda. Berikut langkah-langkah umum untuk menggunakan plugin dalam proyek Flutter Anda:

1. Pilih Plugin yang Diperlukan: Pertama, tentukan plugin yang sesuai dengan kebutuhan aplikasi Anda. Anda dapat mencari plugin di **pub.dev**, repositori resmi untuk plugin Flutter. Pilih plugin yang memiliki fitur yang Anda butuhkan dan perhatikan rating serta dokumentasi plugin tersebut.
2. Tambahkan Plugin ke File **pubspec.yaml**: Buka file **pubspec.yaml** dalam proyek Flutter Anda. Temukan bagian dependencies dan tambahkan nama plugin yang telah Anda pilih di bawahnya. Misalnya:

   ```
   dependencies:
    flutter:
     sdk: flutter
    nama_plugin: ^versi
   ```

   Ganti **nama_plugin** dengan nama plugin yang Anda pilih dan versi dengan versi yang sesuai.
3. Jalankan flutter **pub get**: Simpan file **pubspec.yaml** dan kembali ke **terminal** atau **command prompt**. Jalankan perintah flutter **pub get** untuk mengunduh dan menginstal plugin yang baru ditambahkan.
4. Impor dan Gunakan Plugin dalam Kode: Setelah plugin berhasil diinstal, Anda dapat mengimpor dan menggunakannya dalam kode Flutter Anda. Biasanya, dokumentasi plugin akan memberikan

petunjuk tentang cara mengimpor dan menggunakan plugin tersebut dalam aplikasi Anda.

5. Periksa Dokumentasi dan Contoh Kode: Pastikan untuk membaca dokumentasi plugin dan mencoba contoh kode yang disediakan. Dokumentasi biasanya akan memberikan informasi tentang cara menggunakan berbagai fitur plugin dan contoh kode yang dapat Anda gunakan sebagai referensi.
6. Uji dan Perbaiki Masalah: Setelah mengimplementasikan plugin dalam aplikasi Anda, uji fungsionalitasnya secara menyeluruh. Periksa apakah plugin berfungsi seperti yang diharapkan dan perbaiki masalah yang mungkin muncul selama pengujian.

Dengan mengikuti langkah-langkah di atas, Anda dapat dengan mudah menggunakan plugin untuk memperluas fitur dan fungsionalitas aplikasi Flutter Anda. Pastikan untuk memilih plugin yang andal dan terpercaya, serta membaca dokumentasi dengan cermat untuk memahami cara penggunaannya dengan baik.

D. Membuat Plugin Kustom

Untuk membuat plugin kustom (atau sering disebut sebagai "package" di lingkungan Flutter), Anda dapat mengikuti langkah-langkah berikut:

1. Buat Proyek Package Baru: Mulailah dengan membuat proyek Flutter baru dengan menggunakan perintah **flutter create — template=package nama_package**. Ganti **nama_package** dengan nama yang sesuai untuk package yang akan Anda buat.
2. Struktur Proyek Package: Setelah membuat proyek package, Anda akan memiliki struktur direktori yang mirip dengan proyek Flutter biasa. Di dalamnya, Anda akan menemukan **file lib/** yang akan berisi kode untuk package Anda.
3. Buat Kode Package: Mulailah dengan menentukan fitur dan fungsionalitas apa yang ingin Anda masukkan ke dalam package. Tulislah kode sesuai dengan kebutuhan Anda dalam direktori **lib/**.

Anda dapat membuat kelas, fungsi, atau komponen UI sesuai dengan keperluan Anda.

4. Dokumentasikan Package: Pastikan untuk mendokumentasikan package Anda dengan baik. Buatlah deskripsi yang jelas tentang apa yang dapat dilakukan oleh package, bagaimana cara menginstal dan menggunakannya, serta contoh kode penggunaan.
5. Publikasikan Package (Opsional): Jika Anda ingin membagikan package Anda dengan komunitas Flutter, Anda dapat mempublikasikannya di **pub.dev**. Untuk melakukan hal ini, Anda harus memiliki akun Google, dan Anda dapat menerbitkan package dengan menggunakan perintah flutter pub publish.
6. Gunakan Package: Setelah package Anda siap, Anda dapat menggunakannya dalam proyek Flutter lainnya dengan menambahkannya sebagai dependency di file **pubspec.yaml**. Gunakan alamat URL dari repository Git atau pub.dev untuk menambahkan package Anda.
7. Uji Package: Pastikan untuk menguji package Anda secara menyeluruh dalam berbagai skenario penggunaan. Anda dapat membuat unit test atau integrasi test untuk memastikan bahwa package berfungsi dengan baik.
8. Perbarui Package (Jika Diperlukan): Jika ada masalah atau jika Anda ingin menambahkan fitur baru ke dalam package Anda, Anda dapat memperbarui kode dan dokumentasi package tersebut. Jangan lupa untuk memperbarui versi package di file **pubspec.yaml** dan menerbitkan versi baru jika diperlukan.

E. Evaluasi/Soal Latihan

1. Apa perbedaan antara paket dan plugin dalam konteks pengembangan aplikasi Flutter?
2. Jelaskan langkah-langkah yang diperlukan untuk mencari, menginstal, dan mengelola paket dan plugin Flutter menggunakan pubspec.yaml dan Flutter pub command.

3. Bagaimana Anda akan menentukan apakah Anda perlu menggunakan paket atau plugin untuk memenuhi kebutuhan fungsionalitas tambahan dalam aplikasi Flutter?

BAB 9
MENGELOLA DATA

A. Tujuan Pembelajaran

1. Memahami konsep dasar pengelolaan data dalam aplikasi Flutter, termasuk state management dan penggunaan model data.
2. Mengetahui cara mengambil dan menyimpan data dari sumber eksternal seperti API atau database lokal.

B. Membuat Model Data

Model data dalam Flutter adalah representasi struktural dari informasi yang akan digunakan atau disimpan dalam aplikasi. Model data mendefinisikan jenis dan struktur data yang akan digunakan dalam aplikasi, serta bagaimana data tersebut akan diorganisasi dan diakses. Berikut adalah penjelasan lebih detail tentang pembuatan model data di Flutter:

1. Tujuan Model Data: Model data digunakan untuk mengatur dan merepresentasikan informasi yang diperlukan dalam aplikasi. Ini membantu dalam mengelola data dengan cara yang terstruktur dan mudah dimengerti.
2. Pemodelan Objek: Dalam Flutter, model data sering kali direpresentasikan sebagai objek atau kelas. Setiap kelas mewakili entitas tertentu dalam aplikasi, seperti pengguna, produk, pesanan, dan sebagainya.
3. Properti dan Metode: Setiap objek dalam model data memiliki properti yang mewakili atribut atau data yang terkait dengan objek tersebut. Properti-properti ini dapat berupa tipe data primitif seperti string, integer, boolean, atau tipe data kustom lainnya. Objek juga dapat memiliki metode yang digunakan untuk memanipulasi data atau memberikan fungsionalitas tambahan.

4. Encapsulation: Model data sering menggunakan konsep enkapsulasi untuk melindungi data dari akses langsung yang tidak diinginkan. Ini dilakukan dengan membuat properti objek menjadi pribadi (private) dan memberikan metode akses (getter) dan mutator (setter) yang sesuai.
5. Pembuatan Objek: Untuk membuat model data, Anda perlu membuat kelas yang mewakili jenis data yang ingin Anda atur. Misalnya, jika Anda ingin membuat model data untuk pengguna, Anda dapat membuat kelas User yang memiliki properti seperti id, username, dan email.
6. Inisialisasi Objek: Setelah membuat kelas, Anda dapat membuat objek dari kelas tersebut dengan menggunakan konstruktor. Konstruktor digunakan untuk menginisialisasi nilai properti objek saat objek dibuat.
7. Penggunaan Konstruktor: Konstruktor dapat memiliki parameter yang digunakan untuk memberikan nilai awal kepada properti objek saat objek dibuat. Ini memungkinkan Anda untuk membuat objek dengan nilai yang telah ditentukan.
8. Penggunaan Getter dan Setter: Untuk mengakses dan memanipulasi properti objek, Anda dapat menggunakan getter dan setter. Getter digunakan untuk mendapatkan nilai properti, sementara setter digunakan untuk mengatur nilai properti.
9. Serialisasi dan Deserialisasi: Model data sering digunakan dalam konteks penyimpanan data, baik dalam penyimpanan lokal maupun komunikasi dengan server. Serialisasi dan deserialisasi digunakan untuk mengubah objek menjadi format yang dapat disimpan atau dikirim, dan sebaliknya.
10. Validasi Data: Sebelum menyimpan atau menggunakan data dalam aplikasi, seringkali diperlukan validasi untuk memastikan bahwa data tersebut sesuai dengan format atau kriteria tertentu. Validasi dapat dilakukan di dalam model data menggunakan metode validasi yang sesuai.

11. Keterkaitan Antar Objek: Dalam beberapa kasus, objek dalam model data dapat saling terkait satu sama lain. Misalnya, dalam model data pesanan, objek pesanan dapat memiliki keterkaitan dengan objek produk dan pelanggan.
12. Pembaruan dan Penghapusan Objek: Model data juga harus mendukung operasi pembaruan dan penghapusan objek. Metode yang sesuai harus disediakan untuk memperbarui nilai properti objek atau menghapus objek dari struktur data.
13. Penggunaan Model Data dalam Aplikasi: Setelah model data dibuat, Anda dapat menggunakannya dalam aplikasi Anda untuk menyimpan, mengelola, dan menampilkan informasi kepada pengguna. Model data membantu dalam menjaga konsistensi dan struktur data dalam aplikasi.
14. Pemeliharaan dan Pengembangan: Model data merupakan bagian penting dari pengembangan aplikasi dan perlu dipelihara dan diperbarui sesuai dengan perubahan kebutuhan aplikasi atau persyaratan bisnis.

C. Menampilkan Data Dalam Widget

Menampilkan data dalam widget merupakan salah satu aspek penting dalam pengembangan aplikasi Flutter. Berikut adalah langkah-langkah umum untuk menampilkan data dalam widget:

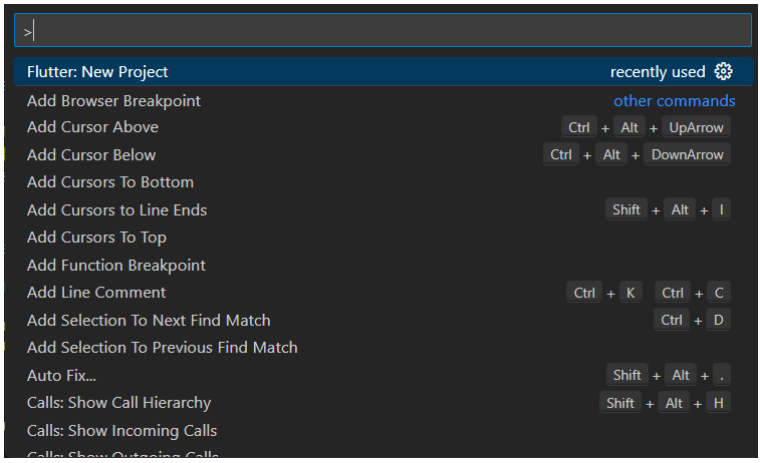
1. Persiapkan Data: Pertama, pastikan Anda memiliki data yang ingin ditampilkan dalam widget. Data ini dapat berasal dari berbagai sumber, seperti variabel lokal, panggilan API, atau database lokal.
2. Model Data: Jika data yang Anda miliki kompleks, Anda mungkin perlu membuat model data untuk mengatur dan merepresentasikan struktur data tersebut. Gunakan kelas atau objek untuk mewakili data dan propertinya.
3. Widget: Selanjutnya, tentukan widget yang akan digunakan untuk menampilkan data. Pilihlah widget yang sesuai dengan jenis data yang ingin Anda tampilkan. Misalnya, gunakan Text untuk menampilkan

teks, Image untuk menampilkan gambar, atau ListView untuk menampilkan daftar data.

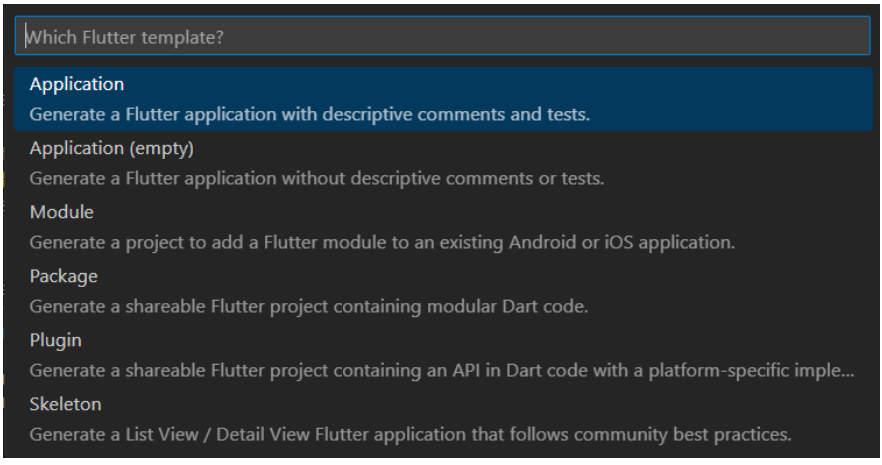
4. Gunakan Widget Builder: Jika data Anda berasal dari sumber eksternal seperti panggilan API, gunakan widget builder seperti FutureBuilder atau StreamBuilder untuk menangani pembaruan data secara dinamis.
5. Buat Tampilan Widget: Implementasikan widget yang akan menampilkan data sesuai dengan kebutuhan Anda. Sesuaikan tata letak, gaya, dan interaksi widget sesuai dengan desain aplikasi Anda.
6. Baca Data: Baca data yang Anda miliki dan gunakan untuk mengisi properti widget yang sesuai. Misalnya, atur teks pada widget Text menggunakan data yang Anda miliki.
7. Iterasi Melalui Data: Jika Anda memiliki daftar data atau data yang perlu diulang, gunakan widget seperti ListView.builder untuk mengiterasi melalui data dan membuat widget yang sesuai untuk setiap item data.
8. Manajemen State: Jika data Anda berubah secara dinamis, pastikan untuk mengelola state dengan benar agar widget dapat diperbarui secara otomatis saat data berubah. Gunakan state management solution seperti setState, Provider, Bloc, atau GetX sesuai dengan kompleksitas aplikasi Anda.
9. Uji Tampilan: Pastikan untuk menguji tampilan widget Anda di berbagai kondisi, seperti ketika data kosong, data masih dimuat, atau data berubah secara dinamis. Pastikan tampilan widget Anda responsif dan dapat menangani semua skenario yang mungkin terjadi.
10. Optimasi Performa: Terakhir, pastikan untuk mengoptimalkan performa aplikasi Anda dengan memperhatikan cara Anda mengambil, menyimpan, dan menampilkan data. Gunakan teknik caching, lazy loading, dan paginasi jika diperlukan untuk meningkatkan responsivitas aplikasi Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

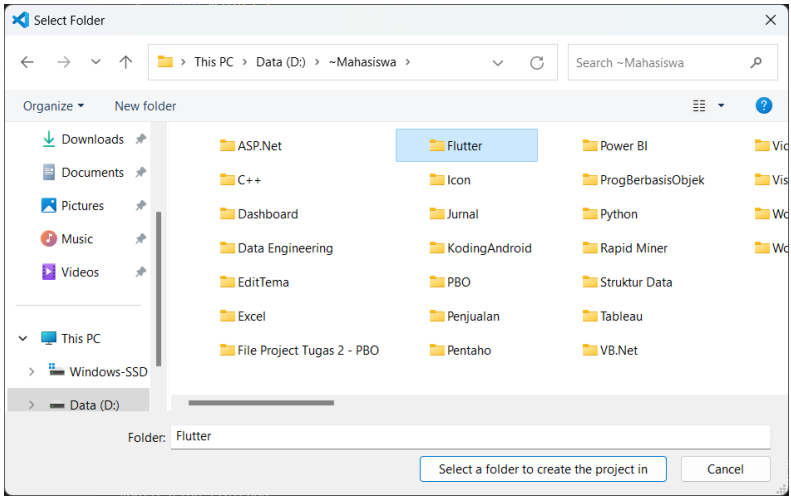


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **data_widget** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

class MyApp extends StatelessWidget {
 // Data yang akan ditampilkan dalam widget
 final List<String> dataList = ['Item 1', 'Item 2', 'Item 3', 'Item 4'];

 MyApp({super.key});

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

```
      appBar: AppBar(
        title: const Text('Data Display Example'),
      ),
      body: ListView.builder(
        itemCount: dataList.length,
        itemBuilder: (context, index) {
          // Untuk setiap item dalam data list, buat widget ListTile
          return ListTile(
            title: Text(dataList[index]), // Teks sesuai dengan data
            onTap: () {
              // Tindakan yang akan diambil saat item diklik
              debugPrint('Item ${index + 1} clicked');
            },
          );
        },
      ),
    ),
  );
 }
}
```

Dalam contoh di atas, kita menggunakan widget ListView.builder untuk menampilkan daftar data. Metode itemBuilder memungkinkan kita untuk membuat widget yang sesuai untuk setiap item dalam daftar data.

Kita menggunakan ListTile sebagai widget untuk menampilkan setiap item dalam daftar. Properti title dari ListTile diisi dengan teks dari data yang sesuai.

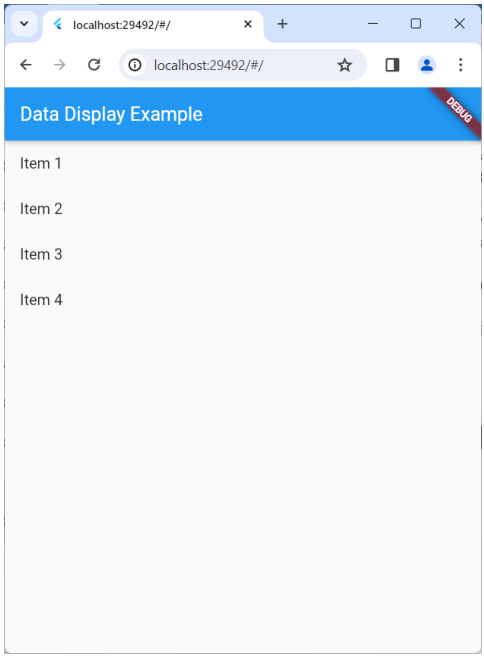
Ketika item dalam daftar diklik, sebuah pesan akan dicetak ke konsol yang menunjukkan item mana yang diklik. Ini adalah contoh sederhana dari cara menangani interaksi pengguna dalam aplikasi Anda.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah

emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



D. Memanipulasi Data

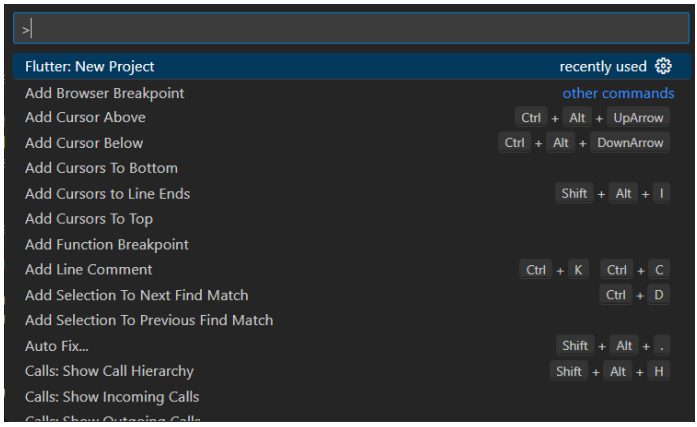
Memanipulasi data dalam Flutter adalah proses mengubah, mengelola, atau memanipulasi informasi yang disimpan dalam aplikasi. Ini melibatkan serangkaian tindakan seperti mengambil, menambah, menghapus, atau memperbarui data sesuai dengan kebutuhan aplikasi. Berikut adalah penjelasan lebih lanjut tentang memanipulasi data dalam konteks pengembangan aplikasi Flutter:

1. Mengambil Data: Langkah pertama dalam memanipulasi data adalah mengambil data dari sumbernya. Ini bisa berupa data yang disimpan secara lokal di aplikasi, data dari panggilan API eksternal, atau data dari database lokal atau cloud. Dengan menggunakan metode seperti http.get atau metode lainnya, Anda dapat mengambil data dari sumbernya.
2. Menyimpan Data: Setelah data diambil, seringkali perlu disimpan dalam aplikasi untuk digunakan nanti. Ini dapat dilakukan dengan menyimpan data dalam variabel, struktur data seperti daftar atau peta, atau penyimpanan lokal seperti Shared Preferences atau SQLite database.
3. Memperbarui Data: Data dalam aplikasi dapat berubah seiring waktu, dan seringkali perlu diperbarui atau dimodifikasi. Ini bisa berupa memperbarui nilai properti objek, menambah atau menghapus item dari daftar, atau memperbarui data dalam database. Anda dapat menggunakan metode seperti setState atau metode lainnya untuk memperbarui data dan memperbarui tampilan widget secara bersamaan.
4. Validasi Data: Sebelum memanipulasi data, penting untuk memvalidasi data untuk memastikan bahwa data tersebut sesuai dengan format atau kriteria tertentu. Ini melibatkan pemeriksaan apakah data yang dimasukkan pengguna sesuai dengan tipe data yang diharapkan, apakah data tersebut kosong, atau apakah data tersebut memenuhi syarat lainnya.
5. Menyortir Data: Terkadang Anda mungkin perlu menyortir data berdasarkan kriteria tertentu, seperti abjad, tanggal, atau nilai numerik. Anda dapat menggunakan metode seperti sort untuk menyortir data sesuai dengan kebutuhan Anda.
6. Filtering Data: Filtering data melibatkan menampilkan hanya data yang memenuhi kriteria tertentu. Ini bisa berupa menampilkan hanya item yang cocok dengan pencarian pengguna, menampilkan item yang sesuai dengan kategori tertentu, atau menampilkan data yang memenuhi syarat lainnya.

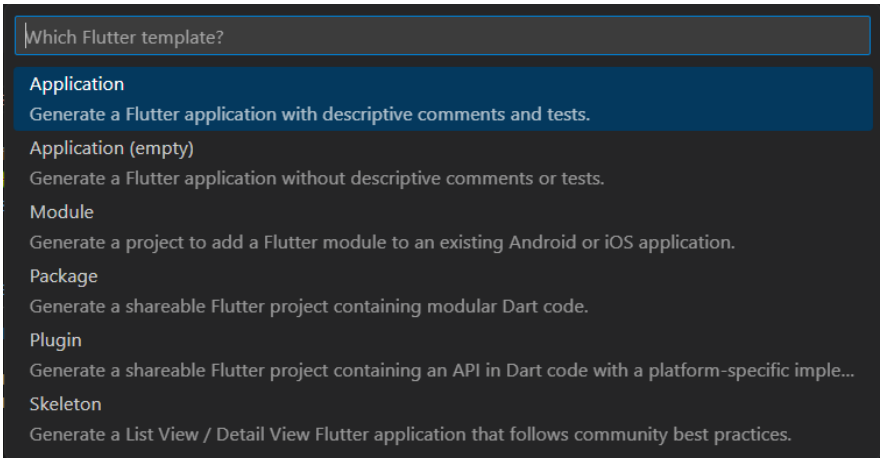
7. Menggabungkan Data: Kadang-kadang Anda perlu menggabungkan atau menggabungkan data dari sumber yang berbeda. Ini bisa berupa menggabungkan data dari beberapa panggilan API, menggabungkan data dari beberapa tabel database, atau menggabungkan data dari sumber lainnya.
8. Penghapusan Data: Ketika data tidak lagi diperlukan atau sudah kadaluarsa, penting untuk menghapusnya untuk membebaskan sumber daya dan ruang penyimpanan. Ini melibatkan menghapus data dari variabel, struktur data, atau penyimpanan lokal.
9. Pengelompokan Data: Terkadang Anda perlu mengelompokkan data ke dalam kategori atau kelompok tertentu. Ini bisa berupa mengelompokkan data berdasarkan kriteria tertentu, seperti tanggal, lokasi, atau kategori.
10. Optimasi Kinerja: Saat memanipulasi data, penting untuk memperhatikan kinerja aplikasi. Anda perlu memastikan bahwa operasi memanipulasi data tidak mempengaruhi kinerja aplikasi secara negatif, dan bahwa operasi tersebut dijalankan dengan efisien. Ini bisa melibatkan penggunaan teknik seperti memoization, pengoptimalan algoritma, atau penggunaan teknologi seperti isolates untuk menjalankan operasi secara asinkron.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

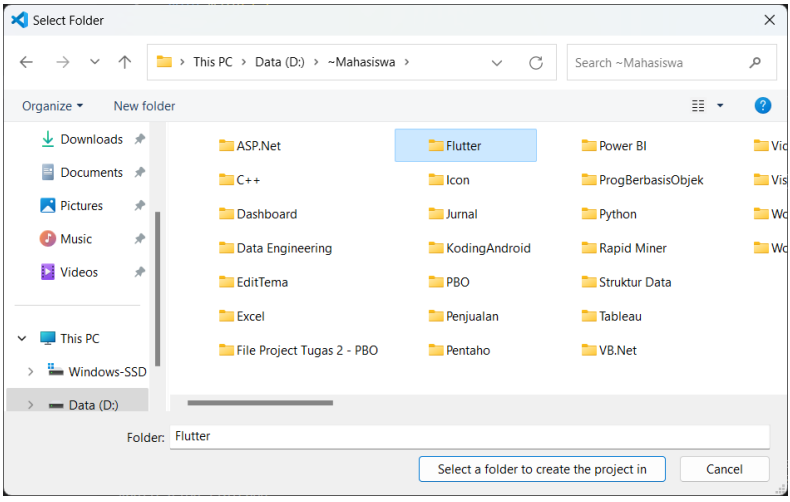


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **manipulasi_data** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

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

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

  @override
  // ignore: library_private_types_in_public_api
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  List<String> dataList = ['Item 1', 'Item 2', 'Item 3'];

// Method untuk menambahkan item baru ke dalam daftar
void addItem() {
  setState(() {
    dataList.add('New Item');
  });
}

// Method untuk memperbarui item di dalam daftar
void updateItem(int index) {
  setState(() {
    dataList[index] = 'Updated Item';
  });
}
```

```
// Method untuk menghapus item dari daftar
void deleteItem(int index) {
 setState(() {
  dataList.removeAt(index);
 });
}

@override
Widget build(BuildContext context) {
 return MaterialApp(
  home: Scaffold(
   appBar: AppBar(
    title: const Text('Contoh Data Manipulasi'),
   ),
   body: ListView.builder(
    itemCount: dataList.length,
    itemBuilder: (context, index) {
     return ListTile(
      title: Text(dataList[index]),
      onTap: () {
       // Memperbarui item yang diklik
       updateItem(index);
      },
      onLongPress: () {
       // Menghapus item yang ditekan lama
       deleteItem(index);
      },
     );
    },
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: () {
     // Menambahkan item baru
     addItem();
    },
    child: const Icon(Icons.add),
   ),
  ),
 );
 }
}
```

Dalam contoh di atas, kita menggunakan **ListView.builder** untuk menampilkan daftar data. Kita juga memiliki tiga method: addItem untuk menambahkan item baru, updateItem untuk memperbarui item yang diklik, dan deleteItem untuk menghapus item yang ditekan lama.

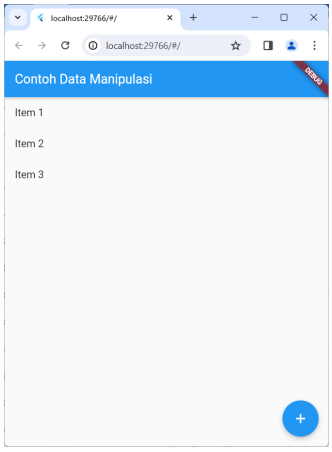
Setiap item dalam daftar ditampilkan sebagai **ListTile**, dan kita menetapkan aksi **onTap** untuk memperbarui item ketika diklik, dan aksi onLongPress untuk menghapus item ketika ditekan lama.

Tombol **FloatingActionButton** digunakan untuk menambahkan item baru ke dalam daftar. Saat tombol ini ditekan, method addItem dipanggil untuk menambahkan item baru ke dalam daftar, dan daftar akan diperbarui secara otomatis.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



E. Menggunakan API

Menggunakan API (Application Programming Interface) dalam Flutter memungkinkan aplikasi Anda berkomunikasi dengan server eksternal untuk mengambil atau mengirim data. Berikut adalah langkah-langkah umum untuk menggunakan API dalam Flutter:

1. Pilih API yang Dibutuhkan: Tentukan API yang akan Anda gunakan. Ini bisa menjadi API publik seperti RESTful API yang tersedia secara gratis, atau API berbayar yang mungkin diperlukan untuk mengakses fitur tertentu.
2. Pelajari Dokumentasi API: Baca dokumentasi API dengan seksama untuk memahami endpoint-endpoint yang tersedia, parameter-parameter yang dibutuhkan, dan format respons yang diharapkan. Ini penting untuk memahami cara menggunakan API dengan benar dalam aplikasi Anda.
3. Buat Permintaan HTTP: Dalam Flutter, Anda dapat menggunakan paket http atau paket lainnya untuk membuat permintaan HTTP ke server API. Gunakan metode seperti get, post, put, atau delete sesuai dengan operasi yang Anda perlukan.
4. Proses Respons: Setelah menerima respons dari server API, Anda perlu memproses respons tersebut sesuai dengan kebutuhan aplikasi Anda. Ini bisa berupa mengambil data yang relevan dari respons, menguraikan respons JSON, atau menangani respons kesalahan.
5. Gunakan Data dalam Aplikasi: Setelah data dari API diproses, Anda dapat menggunakannya dalam aplikasi Anda sesuai kebutuhan. Ini bisa berupa menampilkan data dalam widget, menyimpan data dalam penyimpanan lokal, atau melakukan operasi lainnya sesuai dengan logika aplikasi Anda.
6. Manajemen Koneksi: Penting untuk memperhatikan manajemen koneksi saat menggunakan API dalam aplikasi Flutter. Pastikan untuk menangani kasus-kasus seperti koneksi terputus, respons lambat, atau kesalahan jaringan lainnya dengan benar dalam kode Anda.

7. Optimasi Performa: Saat menggunakan API dalam aplikasi Flutter, perhatikan performa aplikasi Anda. Gunakan teknik caching, pengaturan waktu ulang, atau optimisasi lainnya untuk meningkatkan responsivitas dan kinerja aplikasi Anda.
8. Uji dan Debug: Sebelum menerapkan API dalam produksi, pastikan untuk menguji dan debug integrasi API dengan baik. Uji aplikasi Anda dalam berbagai kondisi dan pastikan bahwa API berfungsi dengan baik dan memberikan hasil yang diharapkan.
9. Pemeliharaan: Setelah mengimplementasikan API dalam aplikasi Anda, pastikan untuk memantau kinerja API secara berkala dan memperbarui integrasi Anda sesuai dengan perubahan API atau kebutuhan aplikasi Anda.
10. Pembaruan dan Pengembangan Lanjutan: Terus pantau perkembangan API yang Anda gunakan dan pertimbangkan untuk memperbarui atau meningkatkan integrasi API Anda sesuai dengan fitur baru atau perubahan kebutuhan aplikasi Anda.

F. HTTP Request

Hal penting yang wajib diketahui dalam mobile programming, terkhusus ketika menggunakan Flutter adalah HTTP Request. Sebab, hal ini menjadi kunci sekaligus jembatan untuk berinteraksi dengan backend agar dapat diolah lebih lanjut, termasuk memanipulasi database.

HTTP request di Flutter mengacu pada proses mengirim permintaan ke server web untuk mengambil atau mengirim data. Ini adalah bagian integral dari pengembangan aplikasi Flutter yang berkomunikasi dengan server backend untuk mendapatkan atau menyimpan informasi yang diperlukan.

1. Pengenalan HTTP Request: HTTP request adalah cara untuk berkomunikasi antara aplikasi Flutter dan server web melalui protokol HTTP. Permintaan ini dapat berupa permintaan untuk mengambil data (GET request), mengirim data (POST request), memperbarui data (PUT request), atau menghapus data (DELETE request).

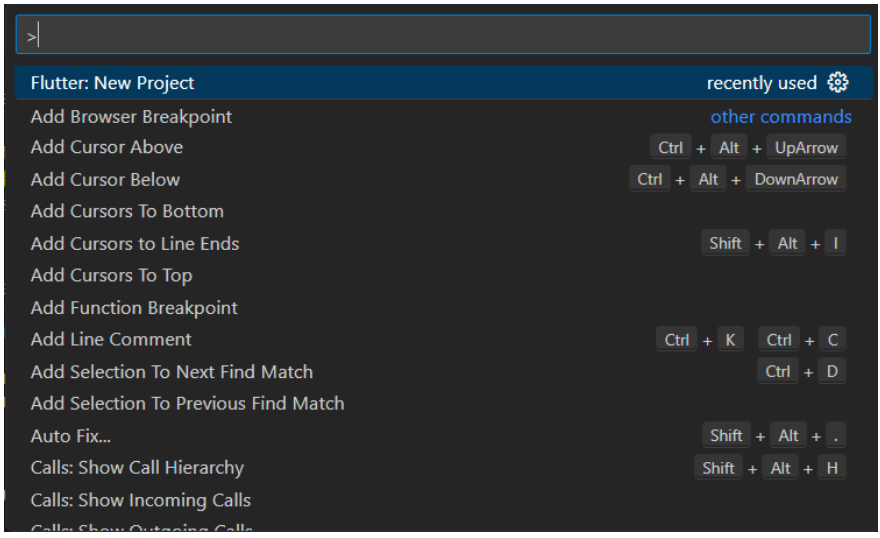
2. Package HTTP di Flutter: Untuk melakukan HTTP request dalam Flutter, pengembang dapat menggunakan package http, yang menyediakan kelas Client untuk membuat permintaan HTTP. Package ini memungkinkan pengembang untuk membuat permintaan ke server dan menangani respons yang diterima.
3. Membuat Permintaan: Untuk membuat permintaan HTTP dalam Flutter, pengembang dapat membuat instance dari kelas Client dari package http dan menggunakan metode seperti get(), post(), put(), atau delete() untuk mengirim permintaan ke server.
4. Menanggapi Respons: Setelah mengirim permintaan, server akan merespons dengan data yang diminta atau status yang sesuai dengan permintaan. Pengembang dapat menanggapi respons ini dengan menentukan fungsi callback untuk menangani respons yang diterima.
5. Menangani Respons: Respons yang diterima dari server dapat berupa berbagai format, seperti JSON, XML, atau teks mentah. Pengembang dapat menggunakan parsing untuk mengubah respons ke dalam format yang sesuai untuk digunakan dalam aplikasi Flutter.
6. Menangani Kesalahan: Dalam beberapa kasus, permintaan HTTP dapat gagal karena koneksi internet yang buruk atau kesalahan server. Pengembang perlu menangani kasus-kasus ini dengan menangkap dan menangani kesalahan yang mungkin terjadi selama proses pengiriman permintaan.
7. Menggunakan Future dan Async/Await: Karena HTTP request adalah operasi asinkronus, pengembang dapat menggunakan Future dan async/await untuk menangani respons dari server dengan cara yang efisien. Ini memungkinkan aplikasi tetap responsif selama proses pengiriman dan penerimaan data dari server.
8. Menangani Otorisasi: Dalam beberapa kasus, pengembang perlu menangani otorisasi untuk mengakses endpoint tertentu di server. Ini melibatkan penambahan header otorisasi ke dalam permintaan HTTP yang dikirim ke server.
9. Memperbarui Tampilan: Setelah menerima respons dari server, pengembang dapat menggunakan data yang diterima untuk

memperbarui tampilan dalam aplikasi Flutter. Misalnya, menampilkan daftar item yang diambil dari server atau menyimpan data yang diinput oleh pengguna ke server.

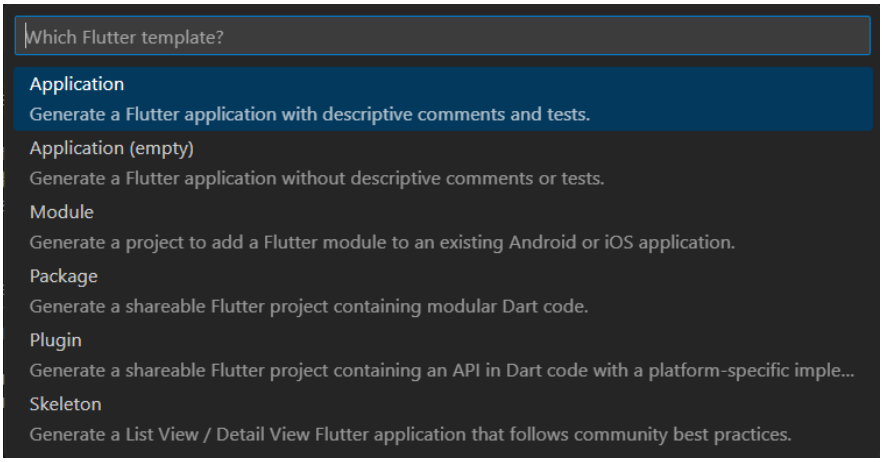
10. Keamanan dan Pengoptimalan: Penting bagi pengembang untuk memperhatikan keamanan dan pengoptimalan saat melakukan HTTP request dalam aplikasi Flutter. Ini termasuk menghindari mengekspos informasi sensitif, memvalidasi data yang diterima dari server, dan mengoptimalkan penggunaan bandwidth dan memori saat mengirim dan menerima data dari server.

G. Contoh koding: Mengambil data dari API dan menampilkannya di aplikasi Flutter

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

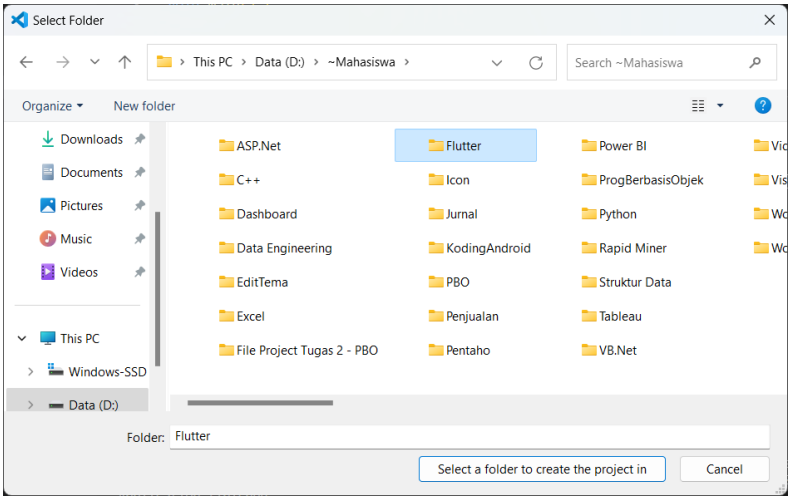


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **flutter_api** kemudian tekan **Enter**.

Silahkan buka file **pubspec.yaml** lalu paste script di bawah ini di bawah **cupertino_icons: ^1.0.2.** Lalu tekan **ctrl + S** untuk menyimpan perubahan.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
void main() {
 runApp(const MyApp());
}

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

 @override
 Widget build(BuildContext context) {
  return const MaterialApp(
   home: HomePage(),
  );
 }
}

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

 @override
 // ignore: library_private_types_in_public_api
 _HomePageState createState() => _HomePageState();
}
```

```
class _HomePageState extends State<HomePage> {
 List<dynamic> dataList = []; // List untuk menyimpan data dari API

 @override
 void initState() {
  super.initState();
  fetchData(); // Panggil method fetchData saat widget diinisialisasi
 }

 Future<void> fetchData() async {
  // Lakukan permintaan GET ke API
  final response =
    await
http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));

  // Cek jika respons berhasil (status code 200)
  if (response.statusCode == 200) {
   // Dekode data JSON
   final jsonData = jsonDecode(response.body);

   setState(() {
    // Perbarui dataList dengan data yang diterima dari API
    dataList = jsonData;
   });
  } else {
   // Tampilkan pesan kesalahan jika respons tidak berhasil
   debugPrint('Failed to load data: ${response.statusCode}');
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('API Data Example'),
   ),
```

```
    body: ListView.builder(
     itemCount: dataList.length,
     itemBuilder: (context, index) {
      return ListTile(
       title: Text(
          dataList[index]['title']), // Menampilkan judul dari setiap item
       subtitle: Text(
          dataList[index]['body']), // Menampilkan isi dari setiap item
      );
     },
    ),
   );
  }
}
```

Dalam contoh di atas, kita menggunakan API JSONPlaceholder (https://jsonplaceholder.typicode.com/posts) yang menyediakan data contoh tentang postingan. Kita mengambil data dari API menggunakan metode **http.get** dalam method **fetchData**.

Ketika data berhasil diterima, kita mendekode respons JSON dan memperbarui dataList dengan data yang diterima. Kemudian, kita menggunakan **ListView.builder** untuk menampilkan setiap item data dalam daftar menggunakan widget **ListTile**.

Pastikan untuk menambahkan **internet** permission ke dalam file **AndroidManifest.xml** jika Anda menggunakan Android. Dan juga pastikan untuk menambahkan **NSAppTransportSecurity** ke dalam file **Info.plist** jika Anda menggunakan iOS.

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

H. Evaluasi/Soal Latihan

1. Apa saja langkah-langkah yang perlu diambil untuk merancang struktur data yang efisien dan skalabel untuk aplikasi Flutter?
2. Di mana Anda dapat mencari sumber daya dan referensi tambahan untuk mempelajari lebih lanjut tentang pengelolaan data dalam aplikasi Flutter?

BAB 10
PENGUJIAN DAN DEBUGGING

A. Tujuan Pembelajaran

Mampu melakukan debugging sederhana pada aplikasi Flutter untuk mengidentifikasi dan memperbaiki kesalahan.

B. Memahami Jenis-Jenis Pengujian

Pengujian adalah bagian penting dari proses pengembangan perangkat lunak, termasuk dalam pengembangan aplikasi Flutter. Berbagai jenis pengujian digunakan untuk memastikan bahwa aplikasi berfungsi dengan baik, sesuai dengan tujuan dan kebutuhan pengguna. Berikut adalah penjelasan tentang beberapa jenis pengujian yang umum digunakan dalam pengembangan aplikasi Flutter:

1. Unit Testing: Unit testing adalah jenis pengujian yang fokus pada pengujian unit kecil dari kode Anda secara terisolasi. Dalam konteks Flutter, unit testing digunakan untuk menguji fungsi, metode, atau kelas individual tanpa ketergantungan pada bagian lain dari aplikasi. Unit testing membantu dalam memastikan bahwa setiap unit kode berperilaku sesuai dengan yang diharapkan.
2. Widget Testing: Widget testing adalah jenis pengujian yang fokus pada pengujian widget dalam aplikasi Flutter. Dalam widget testing, Anda dapat menguji interaksi antara widget, layanan, dan lapisan logika bisnis dalam aplikasi Anda. Ini membantu dalam memastikan bahwa tampilan dan perilaku widget sesuai dengan yang diinginkan.
3. Integration Testing: Integration testing adalah jenis pengujian yang fokus pada pengujian interaksi antara komponen atau bagian dari aplikasi secara bersamaan. Dalam konteks Flutter, integration testing dapat digunakan untuk menguji integrasi antara widget-widget dalam

tampilan, integrasi dengan layanan eksternal, atau integrasi antara lapisan bisnis dalam aplikasi.

4. Acceptance Testing: Acceptance testing adalah jenis pengujian yang fokus pada pengujian apakah aplikasi memenuhi persyaratan fungsional dan non-fungsional yang ditetapkan. Dalam konteks Flutter, acceptance testing dapat melibatkan simulasi interaksi pengguna yang lengkap, seperti navigasi antar layar, pengisian formulir, atau pengujian kinerja dan responsivitas aplikasi.
5. End-to-End Testing: End-to-end testing adalah jenis pengujian yang fokus pada pengujian aplikasi dari awal hingga akhir, mencakup semua komponen, integrasi, dan interaksi dalam aplikasi. Dalam konteks Flutter, end-to-end testing dapat digunakan untuk menguji aplikasi dari perspektif pengguna, mulai dari masuk ke aplikasi hingga menyelesaikan tugas-tugas tertentu.
6. Smoke Testing: Smoke testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa aplikasi dapat dijalankan secara dasar dan tidak ada kesalahan kritis yang terjadi. Dalam konteks Flutter, smoke testing dapat digunakan untuk memastikan bahwa aplikasi dapat dimulai dan berfungsi dengan baik tanpa adanya kesalahan yang fatal.
7. Regression Testing: Regression testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa perubahan baru dalam kode tidak mempengaruhi fungsi yang sudah ada dalam aplikasi. Dalam konteks Flutter, regression testing dapat digunakan untuk memastikan bahwa perubahan baru tidak menyebabkan regresi atau kerusakan pada fitur-fitur yang sudah ada.
8. Stress Testing: Stress testing adalah jenis pengujian yang dilakukan untuk mengevaluasi kinerja dan stabilitas aplikasi dalam kondisi beban yang ekstrem atau di luar batas normal. Dalam konteks Flutter, stress testing dapat digunakan untuk menguji batas-batas kinerja aplikasi, seperti jumlah pengguna simultan yang tinggi atau beban jaringan yang tinggi.
9. Usability Testing: Usability testing adalah jenis pengujian yang dilakukan untuk mengevaluasi seberapa mudah pengguna dapat

menggunakan aplikasi dan seberapa efektif aplikasi tersebut dalam memenuhi kebutuhan pengguna. Dalam konteks Flutter, usability testing dapat melibatkan pengujian interaksi pengguna dengan aplikasi, navigasi antarmuka pengguna, dan kemudahan penggunaan fitur-fitur aplikasi.

10. Accessibility Testing: Accessibility testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa aplikasi dapat diakses dan digunakan oleh pengguna dengan berbagai jenis kecacatan atau keterbatasan fisik. Dalam konteks Flutter, accessibility testing dapat melibatkan pengujian navigasi keyboard, kontras warna, dukungan untuk pembaca layar, dan fitur-fitur aksesibilitas lainnya.
11. Security Testing: Security testing adalah jenis pengujian yang dilakukan untuk mengidentifikasi dan mengatasi kerentanan keamanan dalam aplikasi. Dalam konteks Flutter, security testing dapat melibatkan pengujian untuk mendeteksi kerentanan seperti celah keamanan, injeksi SQL, atau serangan XSS.
12. Localization Testing: Localization testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa aplikasi dapat berfungsi dengan benar dalam berbagai bahasa dan wilayah. Dalam konteks Flutter, localization testing dapat melibatkan pengujian untuk memastikan bahwa teks dan konten aplikasi dapat ditampilkan dengan benar dalam berbagai bahasa dan skrip.
13. Cross-platform Testing: Cross-platform testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa aplikasi berfungsi dengan baik di berbagai platform atau perangkat. Dalam konteks Flutter, cross-platform testing dapat melibatkan pengujian aplikasi di berbagai perangkat Android, iOS, dan web.
14. Offline Testing: Offline testing adalah jenis pengujian yang dilakukan untuk memastikan bahwa aplikasi dapat berfungsi dengan baik dalam mode offline atau ketika koneksi internet tidak tersedia. Dalam konteks Flutter, offline testing dapat melibatkan pengujian untuk memastikan bahwa aplikasi dapat menyimpan dan memproses data secara lokal saat offline.

15. Concurrency Testing: Concurrency testing adalah jenis pengujian yang dilakukan untuk mengevaluasi seberapa baik aplikasi menangani konkurensi atau penggunaan bersama sumber daya di antara beberapa pengguna atau proses. Dalam konteks Flutter, concurrency testing dapat melibatkan pengujian untuk memastikan bahwa aplikasi dapat berperilaku dengan benar dalam situasi konkurensi seperti penggunaan thread atau isolates.

C. Menjalankan Tes Otomatis

Menjalankan tes otomatis adalah bagian integral dari siklus pengembangan perangkat lunak modern, termasuk dalam pengembangan aplikasi Flutter. Proses ini membantu pengembang memvalidasi dan memastikan bahwa aplikasi yang dibuat berfungsi sesuai dengan harapan, meminimalkan kesalahan, dan mempercepat siklus pengujian secara keseluruhan.

Saat menjalankan tes otomatis dalam Flutter, pengembang dapat memilih dari berbagai jenis tes, seperti unit testing, widget testing, dan integration testing. Setiap jenis tes memiliki cakupan dan tujuan yang berbeda, dan sering kali kombinasi dari beberapa jenis tes diperlukan untuk mencapai pengujian yang komprehensif.

Unit testing adalah jenis tes otomatis yang berfokus pada pengujian unit kecil dari kode, seperti fungsi, metode, atau kelas. Dalam konteks Flutter, unit testing dapat digunakan untuk memvalidasi logika bisnis aplikasi dan memastikan bahwa setiap unit kode berperilaku sesuai dengan yang diharapkan.

Widget testing adalah jenis tes otomatis yang berfokus pada pengujian widget dalam aplikasi Flutter. Dengan widget testing, pengembang dapat menguji interaksi antara widget, layanan, dan lapisan logika bisnis dalam aplikasi untuk memastikan bahwa tampilan dan perilaku aplikasi berfungsi dengan benar.

Integration testing adalah jenis tes otomatis yang berfokus pada pengujian integrasi antara komponen atau bagian dari aplikasi secara bersamaan. Dalam konteks Flutter, integration testing dapat digunakan

untuk menguji integrasi antara widget-widget dalam tampilan, integrasi dengan layanan eksternal, atau integrasi antara lapisan bisnis dalam aplikasi.

Menjalankan tes otomatis dalam Flutter biasanya dilakukan dengan menggunakan perangkat bantu seperti Flutter Test atau paket tes lainnya yang disertakan dalam ekosistem Flutter. Ini memungkinkan pengembang untuk menulis, menjalankan, dan menganalisis hasil tes secara efisien.

Tes otomatis dapat dijalankan secara lokal di mesin pengembangan atau dalam lingkungan CI/CD (Continuous Integration/Continuous Deployment) untuk otomatisasi pengujian berkelanjutan. Dengan menggunakan CI/CD, setiap perubahan kode dapat dipicu untuk menjalankan tes otomatis secara otomatis, memastikan bahwa setiap perubahan tidak mempengaruhi fungsionalitas aplikasi secara keseluruhan.

Selama proses pengembangan, pengembang dapat menggunakan alat bantu seperti Flutter DevTools untuk memonitor dan menganalisis hasil tes otomatis. Ini memungkinkan pengembang untuk melacak dan mengevaluasi kinerja pengujian serta memperbaiki masalah yang terdeteksi dengan cepat.

Selain itu, pengujian dapat dikonfigurasi untuk menjalankan tes dalam berbagai mode atau lingkungan, seperti mode debug atau mode rilis, serta menggunakan konfigurasi perangkat atau emulator yang berbeda. Ini membantu memastikan bahwa aplikasi dapat berfungsi secara konsisten di berbagai platform dan lingkungan.

Selama pengujian, pengembang juga dapat menggunakan alat bantu seperti mocking untuk mensimulasikan lingkungan atau dependensi eksternal yang kompleks. Ini memungkinkan pengujian untuk dilakukan secara terisolasi dan meminimalkan ketergantungan pada faktor eksternal yang tidak dapat diprediksi.

Menjalankan tes otomatis dalam Flutter merupakan bagian penting dari proses pengembangan aplikasi yang memastikan bahwa aplikasi berfungsi dengan baik, sesuai dengan tujuan dan harapan, dan meminimalkan risiko kesalahan yang dapat mempengaruhi pengalaman

pengguna akhir. Dengan mengadopsi pendekatan pengujian yang komprehensif dan otomatis, pengembang dapat meningkatkan kualitas dan kehandalan aplikasi Flutter yang dikembangkan.

D. Menggunakan Debugging

Menggunakan debugging adalah proses penting dalam pengembangan aplikasi Flutter yang membantu pengembang mengidentifikasi, memperbaiki, dan memecahkan masalah dalam kode mereka. Debugging memungkinkan pengembang untuk melacak jejak eksekusi kode, memeriksa nilai variabel, dan memahami perilaku aplikasi secara keseluruhan. Dalam Flutter, ada berbagai teknik dan alat yang dapat digunakan untuk debugging, mulai dari debugging di dalam lingkungan pengembangan hingga menggunakan alat bantu eksternal.

Saat menggunakan debugging dalam Flutter, pengembang dapat memanfaatkan fitur-fitur yang disediakan oleh lingkungan pengembangan terintegrasi (IDE) seperti Visual Studio Code atau Android Studio. IDE menyediakan alat bantu untuk menandai breakpoint, menjalankan aplikasi dalam mode debug, dan mengamati perilaku kode secara langsung selama eksekusi.

Salah satu teknik debugging yang umum digunakan dalam Flutter adalah penempatan breakpoint di dalam kode. Breakpoint adalah titik di mana eksekusi program dihentikan, dan pengembang dapat memeriksa nilai variabel, memeriksa tumpukan panggilan, dan memeriksa keadaan aplikasi pada saat tertentu. Dengan menempatkan breakpoint di tempat strategis, pengembang dapat mengidentifikasi titik masuk dan keluar dari fungsi, serta mengidentifikasi potensi masalah dalam alur eksekusi kode.

Selain breakpoint, pengembang juga dapat menggunakan fitur seperti "**log statements**" untuk memeriksa dan mencatat informasi selama eksekusi kode. Log statements adalah pernyataan yang dicetak ke konsol debug saat aplikasi berjalan, dan pengembang dapat menggunakan pernyataan ini untuk memeriksa nilai variabel, status alur program, dan informasi penting lainnya.

Dalam mode debug, pengembang dapat menggunakan fitur "hot reload" untuk melakukan perubahan dalam kode dan melihat perubahan tersebut secara langsung tanpa perlu memulai ulang aplikasi. Hot reload memungkinkan pengembang untuk menguji perubahan secara langsung dalam konteks aplikasi yang berjalan, mempercepat siklus pengembangan dan iterasi.

Selain alat bawaan dari IDE, pengembang juga dapat menggunakan alat bantu eksternal seperti Flutter DevTools untuk debugging. Flutter DevTools adalah suite alat yang menyediakan berbagai fitur untuk menganalisis, memantau, dan debugging aplikasi Flutter. Ini mencakup fitur seperti Inspect Widget, Timeline, dan Performance untuk membantu pengembang dalam menganalisis dan memperbaiki masalah dalam aplikasi.

Saat menggunakan debugging, pengembang juga dapat memanfaatkan teknik seperti "**step into**", "**step over**", dan "**step out**" untuk melacak alur eksekusi kode secara detail. Ini memungkinkan pengembang untuk menavigasi melalui kode dengan akurat dan mengidentifikasi titik masuk dan keluar dari fungsi dengan tepat.

Selain itu, pengembang juga dapat memanfaatkan fitur seperti "**watch expressions**" untuk memonitor nilai variabel tertentu selama eksekusi kode. Dengan menambahkan ekspresi pemantauan, pengembang dapat melacak nilai variabel secara real-time dan memeriksa perubahan nilainya selama eksekusi.

Saat menggunakan debugging dalam Flutter, penting untuk memahami alur eksekusi kode dan memahami cara kerja aplikasi secara keseluruhan. Pengembang perlu menggunakan kombinasi teknik dan alat debugging yang sesuai dengan kebutuhan proyek mereka, serta memiliki pemahaman yang kuat tentang bahasa pemrograman Dart dan konsep-konsep dasar dalam Flutter.

Selama proses debugging, pengembang juga dapat memanfaatkan komunitas dan sumber daya online untuk mencari solusi untuk masalah yang dihadapi. Forum seperti Stack Overflow, grup diskusi,

dan dokumentasi resmi Flutter dapat menjadi sumber informasi yang berharga untuk memecahkan masalah dan mengatasi hambatan dalam pengembangan aplikasi.

Menggunakan debugging dalam pengembangan aplikasi Flutter adalah langkah penting untuk mengidentifikasi, memperbaiki, dan memecahkan masalah dalam kode. Dengan memanfaatkan alat bantu seperti breakpoint, log statements, dan alat bantu eksternal seperti Flutter DevTools, pengembang dapat meningkatkan efisiensi dan produktivitas mereka dalam menyelesaikan masalah dalam aplikasi Flutter.

E. Evaluasi/Soal Latihan

1. Bagaimana cara melakukan debugging sederhana pada aplikasi Flutter menggunakan Visual Studio Code?
2. Mengapa penting untuk melakukan pengujian pada aplikasi Flutter, dan apa alat yang tersedia untuk melakukan pengujian?

BAB 11

MENERAPKAN DESAIN DAN TEMA PADA APLIKASI

A. Tujuan Pembelajaran

1. Memahami pentingnya desain dan tema dalam pengembangan aplikasi mobile menggunakan Flutter.
2. Menguasai konsep dasar tentang desain antarmuka pengguna (UI) dan bagaimana menerapkannya dalam Flutter.
3. Memahami konsep tema (theme) dalam Flutter dan bagaimana cara menerapkannya dalam aplikasi.
4. Mampu membuat tema kustom dan menyesuaikannya dengan merek atau gaya desain aplikasi.
5. Mengerti cara menggunakan material design dan widget-widget bawaan Flutter untuk menciptakan desain yang konsisten.
6. Mengetahui cara membuat tata letak yang baik dan efisien untuk meningkatkan pengalaman pengguna.
7. Menguasai penggunaan warna, tipografi, dan ikon dalam desain aplikasi Flutter.
8. Memahami konsep adaptabilitas antarmuka pengguna untuk berbagai perangkat dan orientasi layar.

B. Memahami Desain Dan Tema Pada Aplikasi Flutter

Desain dan tema mencakup berbagai aspek, mulai dari tata letak dan gaya visual hingga warna, tipografi, dan ikonografi. Dengan memahami prinsip-prinsip desain dan tema, Anda dapat menciptakan pengalaman pengguna yang konsisten dan memikat.

Salah satu aspek utama dari desain dan tema pada aplikasi Flutter adalah tata letak atau layout. Tata letak yang baik memastikan bahwa elemen-elemen antarmuka pengguna ditempatkan secara strategis untuk memaksimalkan keterbacaan dan navigasi. Anda dapat menggunakan widget Flutter seperti Row, Column, atau Stack untuk membuat tata letak yang responsif dan menarik.

Gaya visual atau visual style adalah bagian penting dari desain dan tema. Ini mencakup penggunaan warna, gradient, dan bayangan untuk memberikan estetika yang konsisten dan menarik pada aplikasi Anda. Anda dapat menggunakan kumpulan warna yang konsisten dan harmonis untuk menciptakan suasana yang sesuai dengan merek atau tujuan aplikasi Anda.

Warna adalah elemen penting dalam desain dan tema, dan memilih palet warna yang sesuai dapat memberikan identitas visual yang kuat pada aplikasi Anda. Pastikan untuk memilih warna yang sesuai dengan merek atau tema aplikasi Anda, dan gunakan dengan bijaksana untuk menyoroti elemen penting dan menarik perhatian pengguna.

Selain warna, tipografi juga berperan penting dalam desain dan tema. Pilihlah jenis huruf atau font yang sesuai dengan gaya dan tujuan aplikasi Anda, dan gunakan dengan konsisten untuk memastikan keterbacaan dan konsistensi visual.

Ikonografi adalah elemen desain lainnya yang penting dalam membantu pengguna memahami fungsi dan konten aplikasi. Pilihlah ikon yang sesuai dengan konteks dan tujuan aplikasi Anda, dan pastikan untuk menggunakan ikon dengan ukuran dan gaya yang konsisten.

Konsistensi adalah kunci dalam desain dan tema pada aplikasi Flutter. Pastikan bahwa elemen-elemen antarmuka pengguna seperti warna, tipografi, dan ikon digunakan secara konsisten di seluruh aplikasi Anda untuk menciptakan pengalaman pengguna yang terpadu dan mudah dipahami.

Perhatikan prinsip-prinsip desain seperti kesederhanaan, keterbacaan, dan kejelasan. Hindari penggunaan elemen dekoratif yang berlebihan atau

membingungkan, dan prioritaskan fungsionalitas dan kegunaan dalam desain aplikasi Anda.

Perhatikan pula konteks penggunaan aplikasi Anda. Sesuaikan desain dan tema dengan kebutuhan dan preferensi target pengguna Anda, serta faktor-faktor seperti platform, perangkat, dan situasi penggunaan.

Selain desain visual, Anda juga perlu memperhatikan interaksi pengguna atau user interaction dalam desain dan tema aplikasi Anda. Pastikan bahwa elemen-elemen antarmuka pengguna merespons dengan baik terhadap interaksi pengguna, seperti sentuhan, gerakan, atau input pengguna lainnya.

Gunakan prinsip-prinsip desain responsif untuk memastikan bahwa aplikasi Anda terlihat dan berfungsi dengan baik di berbagai ukuran layar dan orientasi perangkat. Pastikan untuk menguji aplikasi Anda pada berbagai perangkat dan resolusi layar untuk memastikan konsistensi desain.

Anda dapat menggunakan widget Flutter seperti Theme dan ThemeData untuk mengatur tema aplikasi secara global. Dengan menggunakan Theme, Anda dapat dengan mudah mengubah warna, tipografi, dan gaya visual lainnya di seluruh aplikasi Anda dengan cepat dan konsisten.

Pertimbangkan juga untuk menggunakan desain adaptif atau adaptive design dalam pengembangan aplikasi Flutter. Desain adaptif memungkinkan aplikasi Anda menyesuaikan diri dengan perangkat dan platform yang berbeda, memberikan pengalaman pengguna yang optimal di setiap konteks.

Anda dapat memanfaatkan komponen UI yang telah dibuat sebelumnya atau menggunakan paket widget dari pub.dev untuk mempercepat proses desain dan pengembangan. Hal ini dapat membantu Anda menghemat waktu dan upaya dalam membuat antarmuka pengguna yang menarik dan fungsional.

Jangan lupakan pentingnya evaluasi dan iterasi dalam desain dan tema aplikasi Anda. Teruslah mengumpulkan umpan balik dari pengguna

dan melakukan perubahan atau penyempurnaan yang diperlukan untuk meningkatkan pengalaman pengguna dan mencapai tujuan aplikasi Anda. Dengan terus berinovasi dan memperbaiki desain dan tema aplikasi Anda, Anda dapat memastikan bahwa aplikasi Anda tetap relevan dan menarik bagi pengguna Anda.

C. Menggunakan Material Design

Material Design adalah sistem desain yang dikembangkan oleh Google, yang menekankan pada pola desain yang bersih, konsisten, dan intuitif. Berikut adalah beberapa aspek yang perlu dipertimbangkan ketika menggunakan Material Design dalam aplikasi Flutter:

Widget Material: Flutter menyediakan kumpulan widget Material yang kaya, yang dirancang sesuai dengan prinsip-prinsip Material Design. Widget seperti AppBar, Button, Card, dan ListTile adalah beberapa contoh widget Material yang dapat Anda gunakan dalam aplikasi Anda.

Gaya Visual Material: Material Design menekankan pada penggunaan warna yang cerah, tipografi yang jelas, dan bayangan untuk memberikan kedalaman dan dimensi pada antarmuka pengguna. Gunakan palet warna Material dan jenis huruf yang direkomendasikan untuk menciptakan estetika yang konsisten dengan desain Material.

Efek Animasi: Material Design juga menekankan pada penggunaan efek animasi yang halus dan alami untuk meningkatkan interaksi pengguna. Gunakan animasi seperti transition, elevation, dan ripple effect untuk memberikan umpan balik visual yang menyenangkan dan responsif kepada pengguna.

Komponen UI Material: Material Design menyediakan berbagai komponen UI siap pakai seperti BottomNavigationBar, FloatingActionButton, dan Drawer, yang dapat Anda gunakan untuk membangun antarmuka pengguna yang lengkap dan konsisten dengan prinsip-prinsip Material Design.

Responsif dan Adaptif: Desain Material juga mendorong penggunaan prinsip desain responsif dan adaptif untuk memastikan bahwa aplikasi

Anda dapat menyesuaikan diri dengan berbagai ukuran layar dan orientasi perangkat dengan baik. Gunakan widget dan layout yang responsif untuk memastikan aplikasi Anda terlihat dan berfungsi dengan baik di berbagai perangkat.

Koherensi Visual: Penting untuk mempertahankan koherensi visual dalam aplikasi Anda dengan menggunakan elemen desain Material secara konsisten di seluruh aplikasi. Ini mencakup penggunaan warna, tipografi, ikon, dan animasi yang konsisten dengan pedoman Material Design.

Navigasi dan Navigasi: Material Design menyediakan pedoman yang jelas untuk navigasi dan navigasi dalam aplikasi. Gunakan komponen navigasi Material seperti BottomNavigationBar, TabBar, dan Drawer untuk membuat navigasi yang intuitif dan mudah digunakan bagi pengguna.

Feedback Pengguna: Material Design mendorong penggunaan elemen visual seperti indikator loading, stateful widgets, dan pesan umpan balik untuk memberikan pengalaman pengguna yang baik. Pastikan aplikasi Anda memberikan umpan balik yang jelas dan responsif kepada pengguna selama interaksi.

Adaptasi Platform: Material Design dirancang untuk berjalan dengan baik di berbagai platform, termasuk Android, iOS, dan web. Ketika menggunakan Material Design dalam aplikasi Flutter, pertimbangkan untuk mempertahankan konsistensi desain antara platform, sambil memanfaatkan keunggulan Flutter dalam membuat pengalaman yang kaya dan responsif.

Konteks dan Tujuan: Saat menerapkan Material Design dalam aplikasi Flutter, perhatikan konteks dan tujuan aplikasi Anda. Sesuaikan desain dengan kebutuhan dan preferensi pengguna Anda, serta konteks penggunaan aplikasi Anda untuk menciptakan pengalaman yang optimal.

D. Menerapkan Tema Kustom

Menerapkan tema kustom dalam pengembangan aplikasi Flutter adalah cara yang efektif untuk menciptakan tampilan yang unik dan konsisten sesuai dengan merek atau gaya desain yang ingin Anda terapkan. Ketika

Anda menerapkan tema kustom, Anda memiliki kendali penuh atas warna, tipografi, gaya tombol, dan elemen desain lainnya dalam aplikasi Anda.

Salah satu cara untuk menerapkan tema kustom adalah dengan menggunakan widget Theme dalam Flutter. Widget Theme memungkinkan Anda untuk menyesuaikan gaya visual aplikasi Anda secara global, yang akan berdampak pada setiap elemen antarmuka pengguna yang menggunakan tema tersebut. Anda dapat mengatur properti seperti warna primer, warna aksen, tipografi, dan banyak lagi.

Pertimbangkan untuk membuat kelas tema kustom sendiri yang memperluas ThemeData dan menyesuaikannya dengan kebutuhan aplikasi Anda. Ini memungkinkan Anda untuk mengatur tema kustom dengan lebih terperinci, termasuk properti-properti yang tidak disediakan oleh ThemeData bawaan.

Saat membuat tema kustom, pertimbangkan untuk menggunakan warna yang selaras dengan merek atau identitas visual aplikasi Anda. Anda dapat menentukan warna primer, warna aksen, warna latar belakang, dan warna teks yang sesuai dengan palet warna merek Anda.

Selain warna, penting juga untuk memilih jenis huruf atau font yang sesuai dengan gaya dan tujuan aplikasi Anda. Pilihlah jenis huruf yang mudah dibaca dan sesuai dengan identitas merek Anda. Anda dapat menyesuaikan jenis huruf, ukuran, dan gaya huruf untuk berbagai elemen antarmuka pengguna.

Pertimbangkan juga untuk menyesuaikan gaya tombol, input teks, dan elemen UI lainnya dengan tema kustom Anda. Misalnya, Anda dapat menentukan bentuk, bayangan, dan animasi yang sesuai dengan gaya desain yang ingin Anda terapkan.

Pertimbangkan untuk menggunakan modul dan kelas terpisah untuk mengelola tema kustom Anda. Ini memungkinkan Anda untuk memisahkan logika tema dari logika aplikasi dan membuatnya lebih mudah dikelola dan diubah di masa mendatang.

Saat menerapkan tema kustom, penting untuk menguji aplikasi Anda secara menyeluruh untuk memastikan konsistensi dan kualitas desain. Pastikan bahwa tema kustom Anda berfungsi dengan baik di berbagai perangkat dan kondisi penggunaan.

Pertimbangkan untuk membuat variasi tema kustom untuk mode gelap dan mode terang. Ini memungkinkan pengguna untuk memilih mode yang sesuai dengan preferensi mereka dan meningkatkan kenyamanan pengguna.

Anda dapat memanfaatkan animasi dan efek visual lainnya untuk meningkatkan tampilan dan nuansa tema kustom Anda. Gunakan animasi yang halus dan alami untuk memberikan pengalaman pengguna yang menarik dan responsif.

Jika memungkinkan, pertimbangkan untuk memanfaatkan fitur-fitur Flutter seperti theming dengan parameter untuk memberikan fleksibilitas tambahan dalam menerapkan tema kustom. Ini memungkinkan pengguna atau pengembang untuk menyesuaikan tema sesuai dengan preferensi mereka sendiri.

Pastikan untuk dokumentasikan tema kustom Anda dengan baik agar mudah dipahami dan diakses oleh pengembang lain dalam tim Anda. Ini termasuk menyertakan panduan desain, referensi warna, dan instruksi penggunaan tema kustom.

Jadilah terbuka terhadap umpan balik dari pengguna dan rekan tim Anda saat menerapkan tema kustom. Berikan kesempatan kepada mereka untuk memberikan masukan dan saran untuk meningkatkan atau menyempurnakan tema kustom Anda.

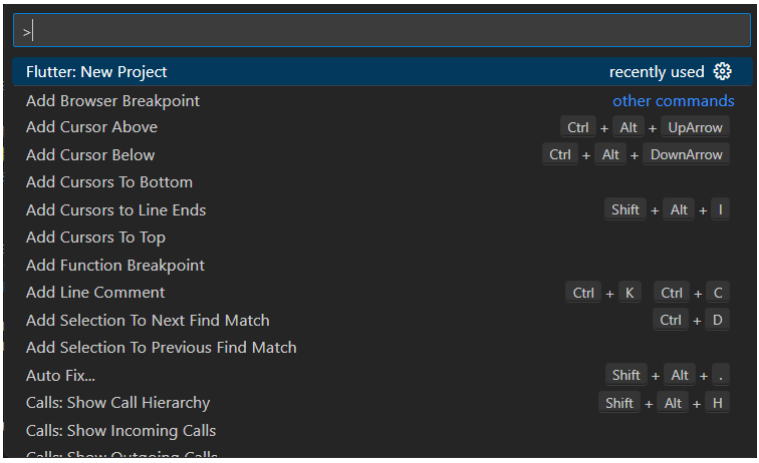
Tetap terhubung dengan perkembangan desain dan teknologi terbaru untuk terus meningkatkan tema kustom Anda. Terus belajar dan bereksperimen dengan berbagai teknik dan strategi desain untuk menciptakan pengalaman pengguna yang unik dan memikat.

Dengan menerapkan tema kustom dengan bijaksana dan terencana, Anda dapat menciptakan antarmuka pengguna yang menarik, konsisten, dan memikat dalam aplikasi Flutter Anda. Tema kustom yang baik

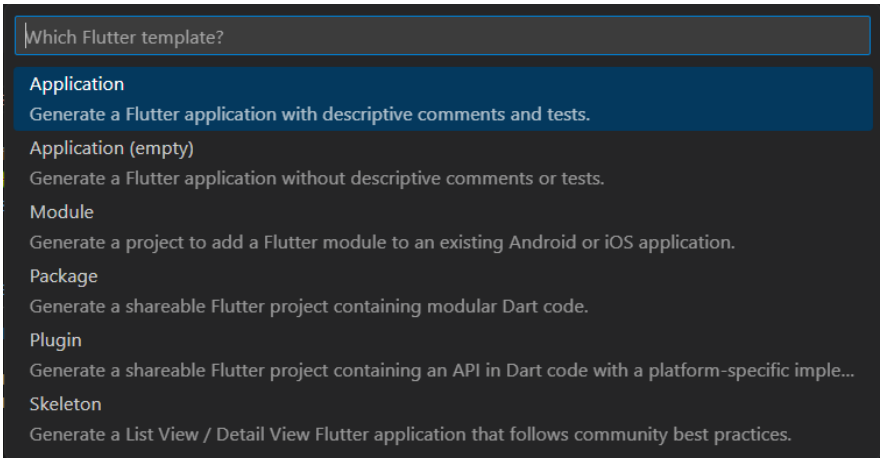
tidak hanya meningkatkan penampilan aplikasi Anda, tetapi juga dapat meningkatkan keterlibatan pengguna dan citra merek Anda secara keseluruhan.

E. Contoh Koding: Menerapkan Tema Kustom Pada Aplikasi Flutter

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

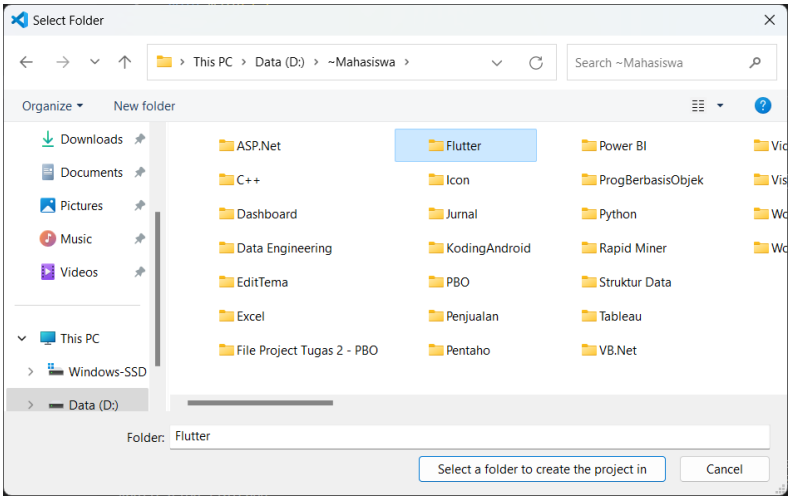


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **tema_kustom** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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: 'Custom Theme Example',
      theme: customTheme(), // Menggunakan tema kustom yang telah
dibuat
      home: const HomePage(),
```

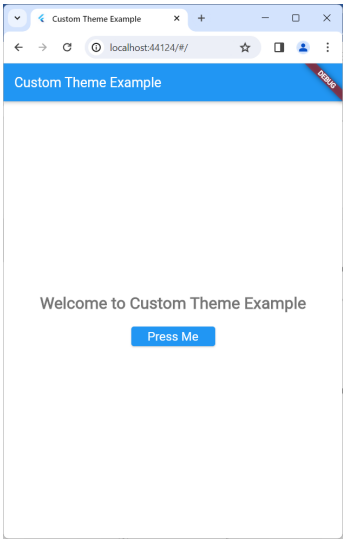
```
    );
  }
}

// Fungsi untuk membuat tema kustom
ThemeData customTheme() {
  return ThemeData(
    primaryColor: Colors.blue, // Warna aksen
    scaffoldBackgroundColor: Colors.white, // Warna latar belakang
    textTheme: const TextTheme(
      displayLarge: TextStyle(
          fontSize: 24, fontWeight: FontWeight.bold), // Gaya teks untuk judul
      bodyLarge: TextStyle(
          fontSize: 16, color: Colors.black87), // Gaya teks untuk konten
    ),
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.blue, // Warna tombol diangkat
        textStyle: const TextStyle(fontSize: 18), // Gaya teks tombol diangkat
        padding: const EdgeInsets.symmetric(
            vertical: 12, horizontal: 24), // Padding tombol diangkat
      ),
    ),
    colorScheme:          ColorScheme.fromSwatch().copyWith(secondary:
Colors.orange),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
      title: const Text('Custom Theme Example'),
    ),
    body: Center(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
       Text(
        'Welcome to Custom Theme Example',
        style: Theme.of(context)
          .textTheme
          .displayLarge, // Menggunakan gaya teks judul dari tema kustom
       ),
       const SizedBox(height: 20),
       ElevatedButton(
        // Menggunakan tombol diangkat dengan gaya dari tema kustom
        onPressed: () {
         // Handle button tap
        },
        child: const Text('Press Me'),
       ),
      ],
     ),
    ),
   );
  }
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



F. Evaluasi/Soal Latihan

1. Bagaimana cara membuat tema kustom dalam Flutter, dan apa yang harus dipertimbangkan saat membuatnya?
2. Apa itu material design, dan bagaimana cara mengimplementasikannya dalam Flutter?
3. Bagaimana cara mengatur warna, tipografi, dan ikon dalam desain aplikasi Flutter?
4. Jelaskan pentingnya tata letak yang baik dalam meningkatkan pengalaman pengguna (UX) aplikasi.

BAB 12

MENYIAPKAN APLIKASI UNTUK PRODUKSI

A. Tujuan Pembelajaran

1. Memahami langkah-langkah yang diperlukan untuk mempersiapkan aplikasi Flutter untuk produksi.
2. Menguasai proses pengoptimalan aplikasi Flutter untuk meningkatkan kinerja dan pengalaman pengguna.
3. Memahami konsep build dan deployment dalam konteks aplikasi Flutter.

B. Mengoptimalkan Aplikasi

Mengoptimalkan aplikasi Flutter adalah proses penting dalam pengembangan untuk memastikan aplikasi berjalan dengan lancar, efisien, dan responsif. Dengan melakukan optimisasi yang tepat, Anda dapat meningkatkan kinerja aplikasi, mengurangi penggunaan sumber daya, dan meningkatkan pengalaman pengguna secara keseluruhan.

Salah satu langkah pertama dalam mengoptimalkan aplikasi Flutter adalah dengan melakukan profil atau memeriksa kinerja aplikasi Anda menggunakan alat bawaan seperti Flutter DevTools atau alat pihak ketiga seperti Firebase Performance Monitoring. Dengan memahami bagaimana aplikasi Anda berperilaku dan mengidentifikasi area yang memerlukan perbaikan, Anda dapat membuat strategi optimasi yang efektif.

Optimalkan tata letak UI aplikasi Anda dengan menggunakan widget Flutter yang efisien dan menghindari penggunaan widget yang berlebihan atau kompleks. Menggunakan widget seperti ListView.builder daripada ListView statis dapat membantu mengurangi overhead memori dan meningkatkan kinerja aplikasi Anda, terutama untuk daftar besar.

Pertimbangkan untuk menggunakan state management yang efisien seperti Provider, Riverpod, atau Bloc untuk mengelola keadaan aplikasi Anda dengan baik. Mengoptimalkan penggunaan keadaan dan meminimalkan pembaruan yang tidak perlu dapat membantu mengurangi konsumsi sumber daya dan meningkatkan responsifitas aplikasi Anda.

Gunakan widget Flutter yang dioptimalkan untuk kinerja seperti ListView.builder, GridView.builder, atau SingleChildScrollView dengan viewport caching untuk mengurangi beban memori saat menampilkan daftar atau konten bergulir.

Pertimbangkan untuk menggunakan caching dan pre-fetching untuk mengurangi waktu muat dan meningkatkan responsifitas aplikasi Anda, terutama saat mengambil data dari internet. Flutter memiliki paket caching HTTP seperti Dio atau HTTP yang dapat membantu meningkatkan kinerja aplikasi Anda.

Optimalkan penggunaan animasi dan efek visual dalam aplikasi Anda dengan membatasi jumlah animasi kompleks atau berlebihan. Menggunakan animasi yang ringan dan responsif dapat membantu menjaga kinerja aplikasi Anda tetap lancar dan responsif.

Lakukan code splitting atau lazy loading untuk memisahkan kode aplikasi Anda menjadi bagian-bagian yang lebih kecil dan hanya memuat kode yang diperlukan saat diperlukan. Ini dapat membantu mengurangi waktu muat awal dan meningkatkan waktu respons aplikasi Anda.

Pastikan untuk menghapus atau memperbaiki segala jenis memory leaks atau sumber daya yang tidak terpakai dalam aplikasi Anda. Memory leaks dapat mengakibatkan penurunan kinerja dan meningkatkan penggunaan memori aplikasi Anda secara keseluruhan.

Selalu gunakan versi Flutter yang terbaru dan perbarui paket dan dependensi Anda secara teratur untuk memanfaatkan peningkatan kinerja dan perbaikan bug yang diperbarui.

Pertimbangkan untuk mengoptimalkan gambar dan media yang digunakan dalam aplikasi Anda dengan mengompresi atau menyesuaikan resolusi dan formatnya sesuai kebutuhan.

Gunakan widget Flutter seperti FutureBuilder atau StreamBuilder untuk mengambil dan menampilkan data secara asinkron tanpa menghalangi antarmuka pengguna.

Hindari penggunaan operasi komputasi berat atau panjang di thread utama aplikasi Anda. Gunakan isolates atau memanfaatkan fitur-fitur async dan await untuk menjalankan operasi komputasi yang intensif secara asinkron dan di luar thread utama.

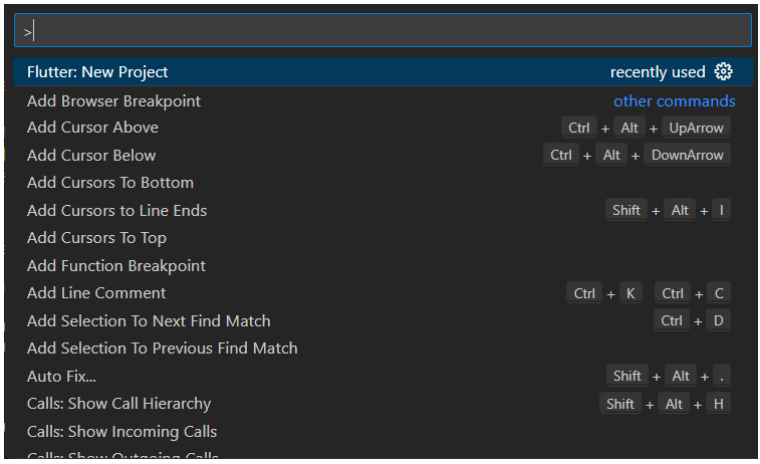
Gunakan widget Flutter seperti ListView.separated daripada ListView.builder jika Anda perlu menampilkan daftar dengan elemen-elemen yang dipisahkan, untuk menghindari perhitungan indeks yang berlebihan.

Optimalkan penggunaan memori dengan melakukan garbage collection secara berkala dan memantau penggunaan memori aplikasi Anda menggunakan alat bawaan Flutter DevTools.

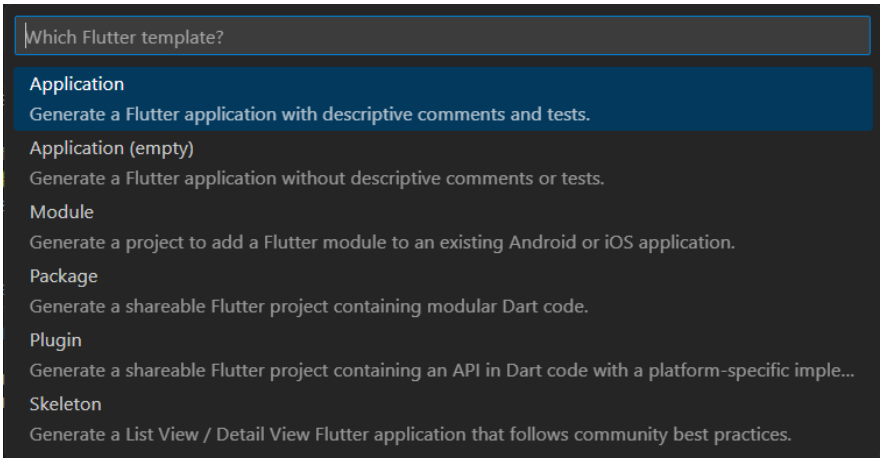
Terakhir, lakukan pengujian dan pemantauan kinerja secara teratur untuk memastikan bahwa perubahan yang Anda buat menghasilkan peningkatan kinerja yang diinginkan dan tidak memperkenalkan masalah kinerja baru dalam aplikasi Anda.

Contoh koding:

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

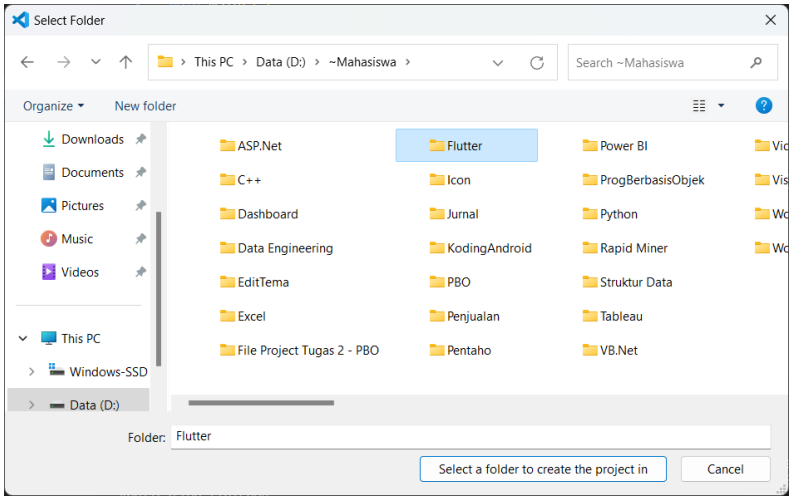


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **aplikasi_optimal** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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: 'Flutter Optimization Example',
   home: HomePage(),
  );
 }
}

class HomePage extends StatelessWidget {
 final List<String> items = List.generate(100, (index) => 'Item $index');

 HomePage({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Optimization Example'),
   ),
   body: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
     Expanded(
      child: ListView.builder(
       itemCount: items.length,
```

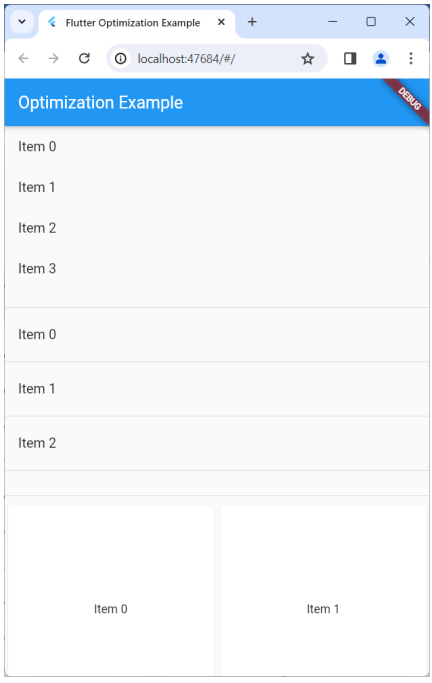
```
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(items[index]),
          );
        },
      ),
    ),
    const Divider(),
    Expanded(
      child: ListView.separated(
        itemCount: items.length,
        separatorBuilder: (context, index) => const Divider(),
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(items[index]),
          );
        },
      ),
    ),
    const Divider(),
    Expanded(
      child: GridView.builder(
        gridDelegate:                                    const
SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
        ),
        itemCount: items.length,
        itemBuilder: (context, index) {
          return Card(
            child: Center(
              child: Text(items[index]),
            ),
          );
        },
      ),
```

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

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



C. Menyiapkan Aplikasi Untuk Rilis

Menyiapkan aplikasi untuk rilis adalah tahap penting dalam pengembangan aplikasi Flutter sebelum aplikasi dapat dinikmati oleh pengguna akhir. Proses ini melibatkan serangkaian langkah yang dirancang untuk memastikan bahwa aplikasi berfungsi dengan baik, memiliki kualitas yang tinggi, dan siap untuk didistribusikan ke platform yang dituju. Berikut adalah penjelasan terperinci tentang langkah-langkah yang terlibat dalam menyiapkan aplikasi untuk rilis di Flutter:

Pertama-tama, pastikan aplikasi telah melewati tahap pengembangan dan pengujian dengan baik. Semua fitur harus berfungsi seperti yang diharapkan dan tidak ada bug yang signifikan yang ditemukan.

Selanjutnya, optimalkan kinerja aplikasi dengan mengidentifikasi dan memperbaiki masalah kinerja seperti waktu muat yang lambat, penggunaan memori yang tinggi, atau masalah kinerja lainnya.

Pastikan bahwa aplikasi memiliki manajemen keadaan yang baik dan tidak ada kebocoran memori yang signifikan. Manajemen keadaan yang efisien penting untuk memastikan aplikasi berjalan dengan lancar dan responsif.

Lakukan pengujian menyeluruh untuk memastikan bahwa aplikasi berfungsi dengan baik di berbagai perangkat dan sistem operasi yang berbeda. Hal ini termasuk pengujian pada perangkat Android dan iOS serta berbagai ukuran layar dan orientasi.

Pastikan aplikasi menggunakan tema dan desain yang konsisten serta sesuai dengan pedoman desain platform yang dituju, seperti Material Design untuk Android dan Human Interface Guidelines untuk iOS.

Periksa kembali kode aplikasi untuk memastikan bahwa tidak ada kode yang tidak digunakan atau tidak perlu yang disertakan dalam rilis akhir. Ini dapat membantu mengurangi ukuran file aplikasi dan meningkatkan kinerja.

Optimalkan gambar dan media yang digunakan dalam aplikasi dengan mengompresi atau menyesuaikan resolusi dan formatnya sesuai

kebutuhan. Ini dapat membantu mengurangi ukuran file aplikasi dan waktu unduh.

Pastikan bahwa semua teks dalam aplikasi diterjemahkan dengan baik dan bebas dari kesalahan tata bahasa atau ejaan. Kualitas terjemahan yang baik penting untuk pengalaman pengguna yang positif.

Uji aplikasi secara menyeluruh untuk memastikan bahwa fitur-fitur penting seperti login, pembayaran, atau interaksi dengan API berfungsi dengan benar. Periksa juga bahwa aplikasi berperilaku seperti yang diharapkan dalam skenario penggunaan yang berbeda.

Pastikan bahwa aplikasi mematuhi persyaratan hukum dan kebijakan privasi yang berlaku, termasuk persyaratan GDPR (General Data Protection Regulation) dan kebijakan privasi platform yang digunakan.

Lakukan pengujian keamanan untuk memastikan bahwa aplikasi tidak rentan terhadap serangan atau pelanggaran keamanan yang mungkin membahayakan data pengguna.

Periksa kembali semua izin yang diminta oleh aplikasi dan pastikan bahwa aplikasi hanya meminta izin yang diperlukan untuk berfungsi. Ini akan membantu meningkatkan tingkat kepercayaan pengguna terhadap aplikasi.

Buat dokumentasi rilis yang jelas dan komprehensif yang mencakup informasi tentang fitur baru, perbaikan bug, dan persyaratan sistem untuk menjalankan aplikasi. Dokumentasi rilis ini akan membantu pengguna dan tim dukungan jika mereka mengalami masalah.

Siapkan materi pemasaran seperti tangkapan layar, video promosi, atau deskripsi aplikasi yang menarik. Materi pemasaran yang baik dapat membantu menarik perhatian pengguna potensial dan meningkatkan unduhan aplikasi.

Terakhir, siapkan proses distribusi untuk rilis aplikasi ke toko aplikasi, seperti Google Play Store untuk Android dan App Store untuk iOS. Pastikan bahwa semua persyaratan dan langkah-langkah yang diperlukan telah dipenuhi sebelum mengirimkan aplikasi untuk tinjauan.

D. Menerbitkan Aplikasi

Menerbitkan aplikasi Flutter adalah langkah penting dalam proses pengembangan aplikasi yang memungkinkan aplikasi Anda menjadi tersedia untuk diunduh dan digunakan oleh pengguna. Proses ini melibatkan serangkaian langkah yang harus diikuti dengan cermat untuk memastikan bahwa aplikasi Anda dapat ditemukan, diunduh, dan digunakan dengan lancar oleh pengguna yang dituju. Berikut adalah penjelasan terperinci tentang langkah-langkah yang terlibat dalam menerbitkan aplikasi Flutter:

Pertama, pastikan aplikasi Anda telah selesai dikembangkan dan diuji secara menyeluruh. Semua fitur harus berfungsi dengan baik, dan tidak ada bug atau masalah kinerja yang signifikan yang ditemukan.

Selanjutnya, siapkan aplikasi Anda untuk distribusi dengan membangun versi rilis yang dioptimalkan untuk performa dan ukuran. Ini melibatkan penggunaan perintah flutter build dengan argumen —release untuk menghasilkan paket aplikasi yang siap untuk didistribusikan.

Setelah itu, buat ikon aplikasi yang menarik dan relevan untuk digunakan sebagai identitas visual aplikasi Anda. Pastikan ikon ini sesuai dengan pedoman desain platform yang dituju, seperti Material Design untuk Android dan Human Interface Guidelines untuk iOS.

Selanjutnya, buatlah deskripsi aplikasi yang informatif dan menarik untuk digunakan dalam toko aplikasi. Deskripsi ini harus menjelaskan fitur-fitur utama aplikasi Anda dan manfaatnya bagi pengguna potensial.

Pastikan bahwa aplikasi Anda mematuhi semua persyaratan dan pedoman toko aplikasi yang berlaku, termasuk persyaratan konten, persyaratan teknis, dan persyaratan privasi. Ini termasuk memeriksa bahwa aplikasi Anda tidak melanggar hak cipta atau hak kekayaan intelektual orang lain.

Selanjutnya, persiapkan tangkapan layar (screenshots) dan video promosi yang menarik untuk digunakan dalam halaman toko aplikasi

Anda. Materi pemasaran ini dapat membantu menarik perhatian pengguna potensial dan meningkatkan tingkat konversi unduhan.

Jika Anda merencanakan peluncuran beta, siapkan proses distribusi beta melalui layanan beta tester yang tersedia di toko aplikasi. Ini memungkinkan Anda untuk mengumpulkan umpan balik dari pengguna beta sebelum peluncuran resmi.

Selanjutnya, tentukan harga aplikasi Anda (jika berlaku) dan siapkan proses pembayaran menggunakan gateway pembayaran yang sesuai dengan platform toko aplikasi yang Anda gunakan.

Jika Anda merencanakan peluncuran di toko aplikasi Google Play Store, persiapkan materi promosi tambahan seperti grafik promosi dan deskripsi fitur-fitur aplikasi Anda untuk digunakan dalam kampanye pemasaran Google Play.

Jika Anda merencanakan peluncuran di toko aplikasi Apple App Store, pastikan bahwa aplikasi Anda memenuhi semua persyaratan Apple dan telah melewati proses tinjauan yang ketat sebelum disetujui untuk peluncuran.

Saat aplikasi Anda siap untuk didistribusikan, buat akun pengembang di toko aplikasi yang Anda tuju (seperti Google Play Console atau Apple Developer Account) dan ikuti proses pendaftaran dan konfigurasi yang diperlukan.

Kemudian, siapkan paket aplikasi Anda untuk diunggah ke toko aplikasi. Ini melibatkan penambahan deskripsi, tangkapan layar, dan informasi lain yang diperlukan dalam halaman aplikasi Anda.

Setelah aplikasi Anda diunggah, tunggu proses tinjauan yang mungkin diperlukan oleh toko aplikasi sebelum aplikasi Anda tersedia untuk diunduh oleh pengguna.

Terakhir, setelah aplikasi Anda tersedia untuk diunduh, pastikan untuk memantau kinerja dan umpan balik dari pengguna, dan lakukan pembaruan reguler untuk meningkatkan dan memperbaiki aplikasi Anda seiring waktu.

E. Evaluasi/Soal Latihan

1. Jelaskan langkah-langkah yang diperlukan untuk mengoptimalkan kinerja aplikasi Flutter sebelum diluncurkan ke produksi.
2. Apa yang dimaksud dengan proses build dan deployment dalam pengembangan aplikasi Flutter?

BAB 13

TIPS DAN TRIK PENGEMBANGAN FLUTTER

A. Tujuan Pembelajaran

1. Menguasai teknik-teknik yang dapat membantu mengatasi masalah umum dalam pengembangan aplikasi Flutter.
2. Memahami konsep-konsep lanjutan dalam Flutter yang dapat membantu meningkatkan kualitas dan kinerja aplikasi.
3. Menguasai teknik-debugging lanjutan untuk mengidentifikasi dan memperbaiki bug yang kompleks dalam aplikasi Flutter.
4. Mempelajari tips dan trik untuk meningkatkan pengalaman pengguna dan membuat aplikasi Flutter lebih menarik dan interaktif.

B. Mengetahui Tips Dan Trik Pengembangan Flutter

Mengetahui tips dan trik pengembangan Flutter dapat membantu meningkatkan produktivitas dan kualitas aplikasi yang Anda bangun. Berikut adalah beberapa tips dan trik yang berguna dalam pengembangan Flutter:

1. Memahami Widget: Widget adalah elemen dasar dalam pembangunan aplikasi Flutter. Penting untuk memahami berbagai jenis widget yang tersedia dan bagaimana cara menggunakannya secara efektif dalam membangun antarmuka pengguna yang responsif dan menarik.
2. Menggunakan State Management: State management adalah konsep penting dalam pengembangan aplikasi Flutter, terutama saat mengelola keadaan dinamis aplikasi. Ada berbagai pendekatan state management yang tersedia, seperti Provider, Bloc, Redux, dan lain-lain. Pilihlah yang paling sesuai dengan kebutuhan proyek Anda.
3. Memanfaatkan Widget Library: Flutter menyediakan kaya koleksi widget dalam widget library. Manfaatkan widget library ini untuk mempercepat pengembangan aplikasi Anda. Anda juga dapat mencari

paket widget tambahan dari pub.dev untuk menambah fungsionalitas aplikasi Anda.

4. Menggunakan Hot Reload: Hot reload adalah fitur yang sangat berguna dalam pengembangan Flutter yang memungkinkan Anda melihat perubahan kode secara langsung dalam aplikasi yang sedang berjalan. Manfaatkan fitur ini untuk mempercepat iterasi pengembangan Anda.
5. Menyusun Kode Dengan Baik: Praktik penyusunan kode yang baik sangat penting dalam pengembangan Flutter. Gunakan konsep seperti pembagian kode menjadi modul-modul terpisah, penerapan pola desain, dan dokumentasi kode untuk membuat kode lebih mudah dipelihara dan dikelola.
6. Menggunakan DevTools: Flutter DevTools adalah kumpulan alat pengembangan yang kuat yang membantu Anda menganalisis, memantau, dan debugging aplikasi Flutter Anda dengan lebih baik. Pelajari cara menggunakan DevTools untuk meningkatkan efisiensi pengembangan Anda.
7. Menggunakan Animasi: Animasi dapat meningkatkan pengalaman pengguna aplikasi Flutter. Pelajari cara menggunakan widget animasi Flutter dan manfaatkan animasi untuk membuat aplikasi Anda lebih menarik dan interaktif.
8. Tes Otomatis: Menulis tes otomatis adalah praktik terbaik dalam pengembangan perangkat lunak yang membantu memastikan kualitas dan keandalan aplikasi Anda. Pelajari cara menulis dan menjalankan tes otomatis dalam Flutter untuk memastikan aplikasi Anda bebas dari bug.
9. Mengoptimalkan Kinerja: Ketahui cara mengoptimalkan kinerja aplikasi Flutter Anda. Gunakan teknik seperti lazy loading, caching, dan pengoptimalan UI untuk meningkatkan responsivitas dan efisiensi aplikasi Anda.
10. Belajar dari Komunitas: Komunitas Flutter sangat aktif dan ramah. Gunakan sumber daya seperti forum, grup diskusi, tutorial, dan dokumentasi resmi untuk memperdalam pemahaman Anda tentang Flutter dan memperoleh wawasan baru dari pengembang lain.

11. Berlangganan Berita dan Update: Flutter terus berkembang dengan cepat, dan ada banyak perbaikan dan fitur baru yang diperkenalkan dalam setiap rilis. Berlangganan buletin Flutter atau mengikuti akun media sosial resmi Flutter untuk tetap terinformasi tentang pembaruan terbaru dan praktik terbaik dalam pengembangan Flutter.

C. Meningkatkan Kinerja Aplikasi

Meningkatkan kinerja aplikasi merupakan aspek krusial dalam pengembangan perangkat lunak yang dapat mempengaruhi pengalaman pengguna akhir secara signifikan. Dalam konteks Flutter, ada beberapa strategi dan praktik yang dapat diterapkan untuk meningkatkan kinerja aplikasi Anda.

Pertama, penting untuk memahami bahwa kinerja aplikasi Flutter dipengaruhi oleh berbagai faktor, termasuk kompleksitas UI, logika bisnis, penggunaan sumber daya perangkat, dan kecepatan jaringan. Oleh karena itu, meningkatkan kinerja aplikasi melibatkan peningkatan dalam berbagai aspek ini.

Salah satu langkah pertama dalam meningkatkan kinerja aplikasi adalah dengan mengoptimalkan UI. Gunakan widget yang ringan dan sederhana, hindari widget yang kompleks atau berat yang dapat memperlambat pembuatan UI. Selain itu, hindari penggunaan terlalu banyak widget yang tidak perlu atau redundan, karena hal ini dapat meningkatkan beban rendering.

Perhatikan penggunaan state management dalam aplikasi Anda. Pilihlah metode state management yang tepat sesuai dengan kebutuhan aplikasi Anda, dan hindari over-engineering atau penggunaan state management yang berlebihan yang dapat mempengaruhi kinerja aplikasi.

Optimalkan penggunaan memori dalam aplikasi Anda dengan mengelola memori secara efisien. Hindari alokasi memori yang berlebihan atau memori bocor yang dapat menyebabkan aplikasi menjadi lambat atau tidak responsif. Gunakan alat bantu seperti Flutter DevTools untuk

memantau penggunaan memori aplikasi Anda dan mengidentifikasi masalah yang mungkin timbul.

Pertimbangkan untuk menggunakan teknik caching untuk mengurangi waktu pengambilan data dari server atau penyimpanan lokal. Dengan menyimpan data yang sering digunakan dalam cache, Anda dapat mengurangi beban jaringan dan meningkatkan responsivitas aplikasi secara keseluruhan.

Optimalkan tampilan aplikasi Anda dengan menghindari penggunaan terlalu banyak efek animasi yang berat atau kompleks. Meskipun animasi dapat meningkatkan interaktivitas dan keindahan aplikasi, penggunaan berlebihan dapat mempengaruhi kinerja, terutama pada perangkat dengan spesifikasi rendah.

Pastikan aplikasi Anda diuji secara menyeluruh pada berbagai perangkat dan platform untuk memastikan kinerja yang konsisten. Pengujian lintas-platform dapat membantu mengidentifikasi masalah kinerja yang mungkin terjadi pada perangkat dengan spesifikasi yang berbeda.

Pertimbangkan untuk menggunakan teknik seperti code splitting untuk membagi kode aplikasi menjadi bagian-bagian yang lebih kecil dan memuatnya secara dinamis saat diperlukan. Ini dapat membantu mengurangi waktu pemuatan aplikasi dan meningkatkan responsivitas.

Optimalkan penggunaan jaringan dalam aplikasi Anda dengan mengimplementasikan teknik seperti prefetching atau preloading untuk memuat data sebelumnya atau selama pengguna sedang tidak aktif. Hal ini dapat membantu mengurangi waktu tunggu dan meningkatkan responsivitas aplikasi.

Perhatikan ukuran dan format file gambar dan media yang digunakan dalam aplikasi Anda. Gunakan kompresi gambar dan format file yang ringan seperti WebP untuk mengurangi beban pengunduhan dan mempercepat waktu pemuatan.

Gunakan teknik pemrograman asinkronus untuk menghindari blocking UI thread dan memastikan responsivitas aplikasi. Gunakan

fitur-fitur seperti isolates untuk menjalankan tugas-tugas berat secara terpisah dari UI thread dan mencegah aplikasi menjadi tidak responsif.

Pastikan untuk memperbarui SDK Flutter dan paket dependensi Anda secara teratur untuk memanfaatkan perbaikan bug dan peningkatan kinerja yang diperkenalkan dalam versi terbaru.

Selalu pantau dan analisis kinerja aplikasi Anda menggunakan alat bantu seperti Flutter DevTools atau Firebase Performance Monitoring. Ini akan membantu Anda mengidentifikasi area-area yang membutuhkan perbaikan dan mengimplementasikan perbaikan yang diperlukan untuk meningkatkan kinerja.

Pastikan untuk memperbarui dan mengoptimalkan kode aplikasi Anda secara berkala dengan menghapus kode yang tidak perlu atau tidak digunakan. Hindari akumulasi kode yang tidak terpakai yang dapat memperlambat kinerja aplikasi secara keseluruhan.

Terakhir, teruslah belajar dan menjelajahi teknik-teknik pengembangan baru yang dapat membantu meningkatkan kinerja aplikasi Anda. Ikuti pembaruan dan tren dalam industri pengembangan perangkat lunak untuk tetap mendapatkan wawasan terbaru tentang cara meningkatkan kinerja aplikasi Anda dalam lingkungan Flutter.

D. Memperbaiki Bug

Memperbaiki bug adalah bagian penting dari proses pengembangan perangkat lunak di mana pengembang mengidentifikasi, menganalisis, dan memperbaiki masalah yang terjadi dalam aplikasi. Dalam konteks pengembangan aplikasi Flutter, memperbaiki bug melibatkan beberapa langkah yang harus diikuti untuk memastikan bahwa masalah tersebut diperbaiki dengan benar dan tidak muncul lagi di masa mendatang.

Langkah pertama dalam memperbaiki bug adalah mengidentifikasi dan mereproduksi masalah tersebut. Pengembang perlu memahami secara jelas bagaimana bug terjadi, apa yang menyebabkannya, dan langkah-langkah yang diperlukan untuk mereproduksi bug tersebut. Ini dapat

melibatkan pengujian aplikasi secara intensif di berbagai skenario dan kondisi.

Setelah bug berhasil direproduksi, langkah selanjutnya adalah menganalisis kode untuk menemukan penyebab akar dari masalah tersebut. Pengembang perlu memeriksa logika bisnis, penggunaan widget, interaksi dengan state, atau panggilan ke layanan eksternal yang mungkin menyebabkan bug tersebut.

Setelah penyebab bug ditemukan, pengembang dapat mulai memperbaikinya dengan melakukan perubahan kode yang diperlukan. Perbaikan dapat melibatkan perubahan kecil seperti perbaikan sintaksis atau perubahan besar seperti restrukturisasi logika aplikasi.

Selama melakukan perbaikan bug, penting untuk memastikan bahwa perubahan yang diperlukan tidak mempengaruhi fungsi lain dalam aplikasi. Pengembang harus melakukan pengujian menyeluruh untuk memastikan bahwa perbaikan bug tidak memperkenalkan bug baru atau memengaruhi kinerja aplikasi secara keseluruhan.

Setelah perubahan kode selesai, pengembang perlu melakukan pengujian ulang untuk memastikan bahwa bug telah diperbaiki dengan benar. Ini melibatkan pengujian aplikasi di berbagai skenario dan kondisi untuk memverifikasi bahwa masalah tidak muncul lagi setelah perbaikan dilakukan.

Jika bug tersebut terkait dengan dependensi pihak ketiga atau paket Flutter tertentu, pengembang perlu memperbarui atau mengganti dependensi tersebut dengan versi yang lebih baru atau alternatif yang lebih stabil.

Selain itu, penting untuk membuat catatan tentang bug yang ditemukan dan perbaikan yang dilakukan dalam sistem pelacakan bug atau alat manajemen proyek yang sesuai. Ini membantu memelihara catatan historis tentang masalah yang telah ditemui dan langkah-langkah yang telah diambil untuk memperbaikinya.

Pengembang juga dapat memanfaatkan alat bantu seperti debugger dan logcat untuk memeriksa dan menganalisis perilaku aplikasi saat terjadi

bug. Ini membantu dalam memahami alur eksekusi kode dan menemukan titik di mana masalah terjadi.

Selain itu, berkolaborasi dengan tim pengembangan lainnya atau mencari bantuan dari komunitas Flutter dapat menjadi sumber informasi yang berharga dalam memperbaiki bug yang kompleks atau sulit diidentifikasi.

Setelah bug diperbaiki dan diuji ulang, penting untuk merilis pembaruan aplikasi kepada pengguna yang terdampak oleh bug tersebut. Memberikan informasi tentang perbaikan bug dalam catatan rilis aplikasi dapat meningkatkan transparansi dan kepercayaan pengguna terhadap aplikasi Anda.

Setelah rilis pembaruan, penting untuk terus memantau aplikasi untuk memastikan bahwa bug tersebut tidak muncul kembali atau tidak menimbulkan dampak negatif lainnya. Hal ini dapat melibatkan pemantauan log aplikasi dan umpan balik pengguna.

Selain memperbaiki bug yang ada, pengembang juga dapat mencegah munculnya bug baru dengan melakukan kode review, pengujian unit, widget testing, dan integration testing secara teratur.

Pengembang juga dapat menggunakan alat bantu otomatisasi seperti Continuous Integration (CI) dan Continuous Deployment (CD) untuk memastikan bahwa perubahan kode yang dilakukan tidak mempengaruhi kualitas dan kinerja aplikasi secara keseluruhan.

Memperbaiki bug dalam aplikasi Flutter melibatkan serangkaian langkah yang sistematis dan hati-hati, mulai dari identifikasi dan analisis masalah hingga perbaikan dan pengujian ulang. Dengan menggunakan pendekatan yang tepat dan alat yang sesuai, pengembang dapat memastikan bahwa aplikasi mereka berfungsi dengan baik dan bebas dari masalah yang mengganggu.

E. Menerapkan Praktik Pengembangan Terbaik

Menerapkan praktik pengembangan terbaik dalam pengembangan aplikasi Flutter sangat penting untuk memastikan bahwa aplikasi yang dibangun memiliki kualitas yang tinggi, mudah dipelihara, dan dapat diperluas di masa mendatang. Berikut adalah beberapa praktik pengembangan terbaik yang dapat diterapkan dalam pengembangan aplikasi Flutter:

Pertama, penting untuk merancang aplikasi dengan pola desain yang baik. Ini termasuk penggunaan arsitektur yang tepat seperti Model-View-Controller (MVC), Model-View-ViewModel (MVVM), atau BloC (Business Logic Component) untuk memisahkan logika bisnis dari tampilan pengguna dan memudahkan pemeliharaan kode.

Gunakan widget Flutter dengan bijaksana dan konsisten. Hindari penggunaan widget yang kompleks atau berat yang dapat memperlambat pembuatan UI, dan pastikan untuk menggunakan widget yang tepat untuk tugas tertentu seperti ListView untuk menampilkan daftar item.

Pertimbangkan untuk menggunakan widget yang dioptimalkan untuk performa seperti ListView.builder untuk daftar besar atau GridView untuk tata letak grid. Hal ini dapat membantu meningkatkan responsivitas aplikasi dan mengurangi beban memori.

Gunakan widget StatelessWidget ketika memungkinkan untuk menghindari overhead yang terkait dengan pembuatan dan pembaruan widget. Hanya gunakan StatefulWidget ketika diperlukan untuk mengelola keadaan dinamis dalam aplikasi.

Manfaatkan State Management untuk mengelola keadaan aplikasi dengan efisien. Pilihlah metode state management yang sesuai dengan kebutuhan proyek Anda seperti Provider, Bloc, Redux, atau GetX dan gunakan dengan konsisten dalam seluruh aplikasi.

Pastikan untuk menguji aplikasi Anda secara menyeluruh dengan pengujian unit, widget testing, dan integration testing. Ini membantu memastikan bahwa aplikasi berfungsi dengan baik dalam berbagai skenario dan kondisi.

Gunakan kode yang bersih dan mudah dimengerti dengan memberikan nama variabel dan fungsi yang deskriptif, memecah kode menjadi modul-modul yang terpisah, dan menghindari kode yang berlebihan atau tidak perlu.

Selanjutnya, terapkan prinsip-prinsip pemrograman yang solid seperti Single Responsibility Principle (SRP), Don't Repeat Yourself (DRY), dan Keep It Simple, Stupid (KISS) untuk membuat kode lebih mudah dipelihara dan diperluas di masa mendatang.

Pastikan untuk melakukan kontrol versi kode menggunakan sistem kontrol versi seperti Git dan menggunakan alur kerja yang tepat seperti Git Flow untuk mengelola pengembangan dan rilis kode secara efisien.

Selanjutnya, gunakan alat bantu seperti linting dan analisis statis kode untuk mengidentifikasi dan memperbaiki potensi masalah dalam kode Anda seperti bug, kode yang tidak efisien, atau pelanggaran aturan gaya.

Berlangganan buletin Flutter, mengikuti forum, dan bergabung dengan komunitas pengembang Flutter untuk tetap terinformasi tentang pembaruan terbaru, praktik terbaik, dan tips dan trik dalam pengembangan aplikasi Flutter.

Selanjutnya, pelajari dan manfaatkan alat bantu seperti Flutter DevTools untuk menganalisis, memantau, dan debug aplikasi Anda dengan lebih baik. Ini termasuk fitur seperti Inspect Widget, Performance, dan Memory untuk memperbaiki masalah dan meningkatkan kinerja aplikasi.

Selalu berkolaborasi dengan tim pengembangan lainnya dan terbuka terhadap umpan balik dari rekan tim Anda. Diskusikan ide, solusi, dan tantangan bersama untuk mencapai hasil yang lebih baik.

Terakhir, jadilah proaktif dalam memperbarui dan meningkatkan keterampilan Anda dalam pengembangan Flutter dengan terus belajar melalui tutorial, kursus, dan sumber daya online lainnya. Tingkatkan pemahaman Anda tentang konsep-konsep dasar dan lanjutan dalam Flutter untuk menjadi pengembang yang lebih baik.

Dengan menerapkan praktik pengembangan terbaik ini secara konsisten, Anda dapat membangun aplikasi Flutter yang berkualitas tinggi, mudah dipelihara, dan memberikan pengalaman pengguna yang luar biasa.

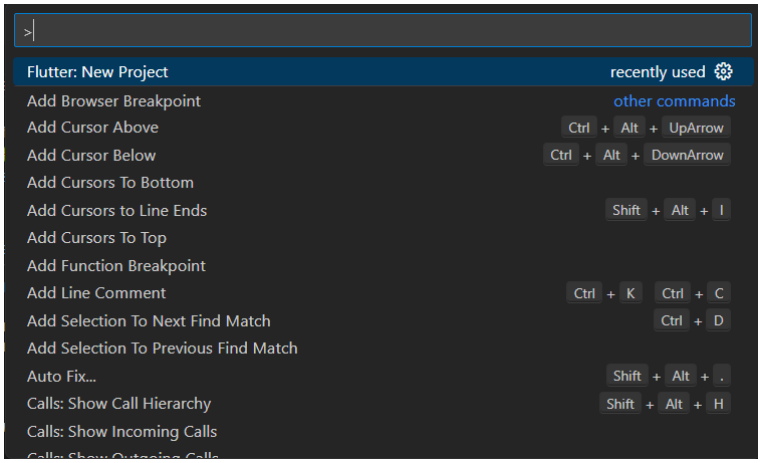
F. Evaluasi/Soal Latihan

1. Jelaskan beberapa tips untuk meningkatkan produktivitas dalam pengembangan aplikasi Flutter.
2. Bagaimana cara mengatasi masalah umum seperti kinerja lambat atau kegagalan kompilasi dalam pengembangan aplikasi Flutter?
3. Bagaimana Anda akan menggunakan tips dan trik yang Anda pelajari dalam bab ini untuk meningkatkan aplikasi Flutter yang sedang Anda kembangkan?

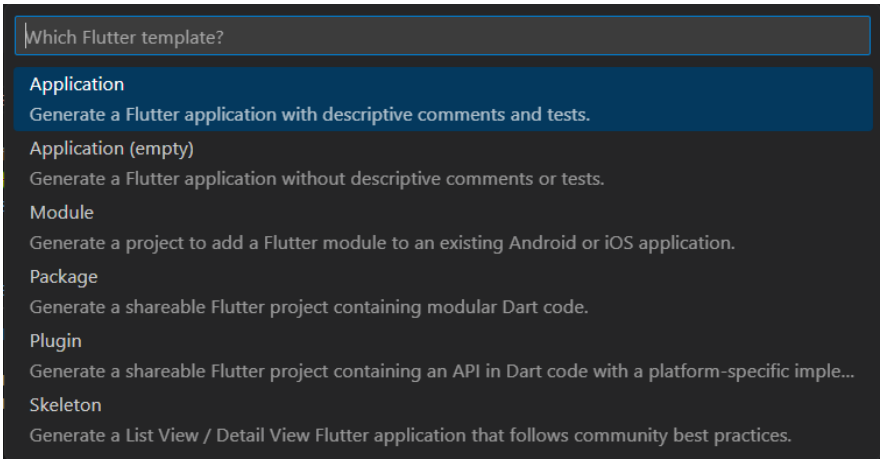
BAB 14
CONTOH KODING

A. Dashboard

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

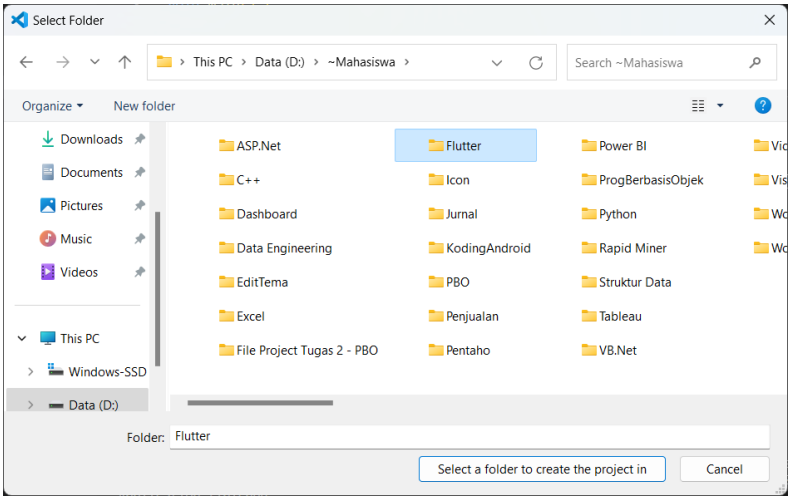


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **dashboard** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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: '92202075 - Dashboard',
   home: Scaffold(
    appBar: AppBar(
     title: const Text('Arie Gunawan - Dashboard'),
```

```
),
body: GridView.count(
 crossAxisCount: 2,
 children: <Widget>[
  Card(
   color: Colors.blue,
   child: InkWell(
    onTap: () {},
    splashColor: Colors.blue.withAlpha(30),
    child: Center(
     child: Column(
      mainAxisSize: MainAxisSize.min,
      children: const <Widget>[
       Icon(
        Icons.shopping_cart,
        size: 70,
        color: Colors.white,
       ),
       Text(
        'Belanja',
        style: TextStyle(color: Colors.white, fontSize: 20),
```

```
          )
        ],
      ),
    ),
  ),
),
Card(
  color: Colors.green,
  child: InkWell(
    onTap: () {},
    splashColor: Colors.green.withAlpha(30),
    child: Center(
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: const <Widget>[
          Icon(
            Icons.payment,
            size: 70,
            color: Colors.white,
          ),
          Text(
            'Pembayaran',
            style: TextStyle(color: Colors.white, fontSize: 20),
          )
        ],
      ),
    ),
  ),
),
Card(
  color: Colors.yellow,
  child: InkWell(
    onTap: () {},
    splashColor: Colors.yellow.withAlpha(30),
    child: Center(
```

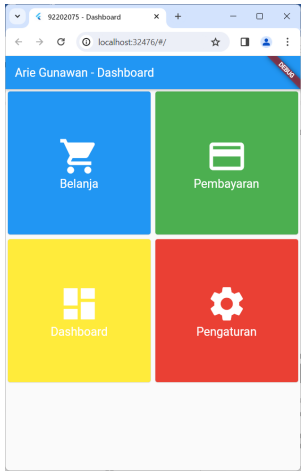
```
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: const <Widget>[
          Icon(
            Icons.dashboard,
            size: 70,
            color: Colors.white,
          ),
          Text(
            'Dashboard',
            style: TextStyle(color: Colors.white, fontSize: 20),
          )
        ],
      ),
    ),
  ),
),
Card(
  color: Colors.red,
  child: InkWell(
    onTap: () {},
    splashColor: Colors.red.withAlpha(30),
    child: Center(
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: const <Widget>[
          Icon(
            Icons.settings,
            size: 70,
            color: Colors.white,
          ),
          Text(
            'Pengaturan',
            style: TextStyle(color: Colors.white, fontSize: 20),
          )
```

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

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

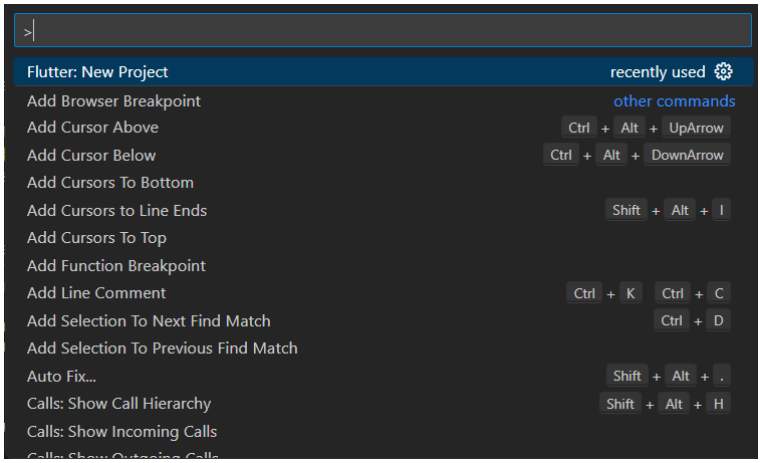
```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

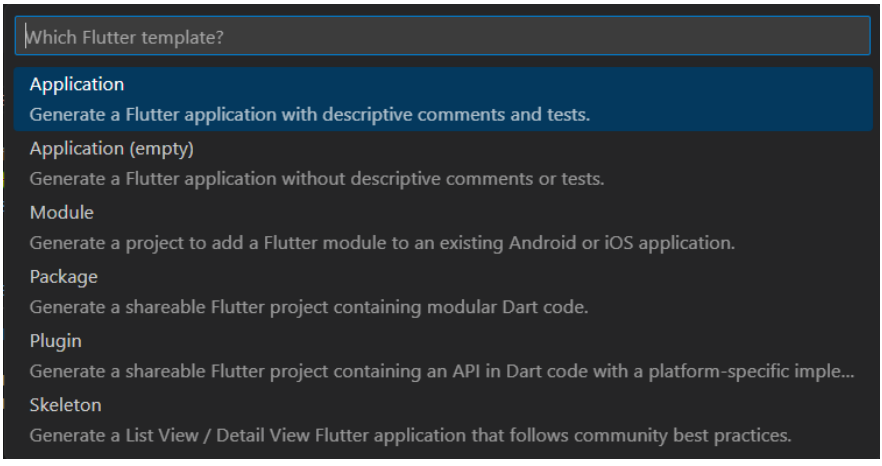


B. Menu Navigasi Sederhana

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

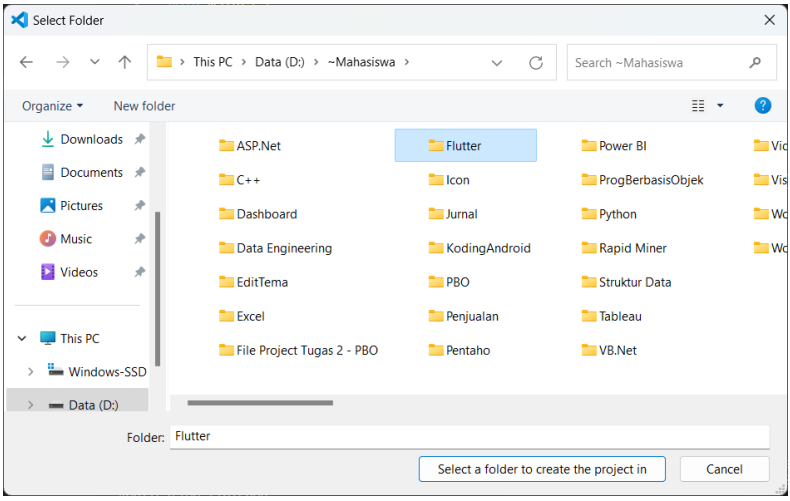


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **menu_navigasi_sederhana** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

void main() => runApp(MyApp());

// ignore: use_key_in_widget_constructors
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: '92202075 - Menu Navigasi',
      home: HomePage(),
      routes: {
        '/about': (context) => AboutPage(),
        '/contact': (context) => ContactPage(),
        '/gallery': (context) => GalleryPage(),
      },
```

```
    );
  }
}

// ignore: use_key_in_widget_constructors
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Arie Gunawan - Menu Navigasi'),
      ),
      drawer: Drawer(
        child: ListView(
          padding: EdgeInsets.zero,
          children: <Widget>[
            const DrawerHeader(
              decoration: BoxDecoration(
                color: Colors.blue,
              ),
              child: Text(
                'Menu',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 24,
                ),
              ),
            ),
            ListTile(
              leading: const Icon(Icons.home),
              title: const Text('Beranda'),
              onTap: () {
                Navigator.pop(context);
              },
            ),
```

```
            ListTile(
              leading: const Icon(Icons.info),
              title: const Text('Tentang Kami'),
              onTap: () {
                Navigator.pop(context);
                Navigator.pushNamed(context, '/about');
              },
            ),
            ListTile(
              leading: const Icon(Icons.contact_phone),
              title: const Text('Hubungi Kami'),
              onTap: () {
                Navigator.pop(context);
                Navigator.pushNamed(context, '/contact');
              },
            ),
            ListTile(
              leading: const Icon(Icons.image),
              title: const Text('Galeri'),
              onTap: () {
                Navigator.pop(context);
                Navigator.pushNamed(context, '/gallery');
              },
            ),
          ],
        ),
      ),
      body: const Center(
        child: Text('Ini adalah halaman Beranda'),
      ),
    );
  }
}

// ignore: use_key_in_widget_constructors
```

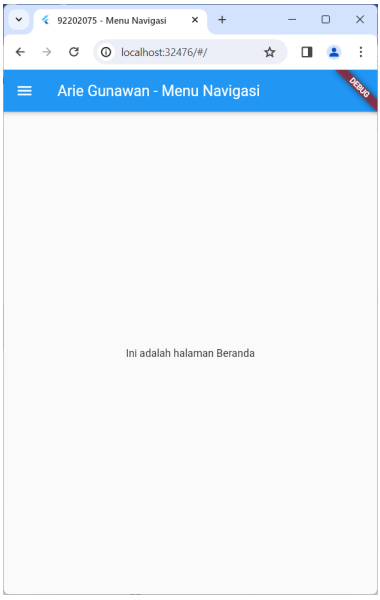
```
class AboutPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Tentang Kami'),
      ),
      body: const Center(
        child: Text('Ini adalah halaman Tentang Kami'),
      ),
    );
  }
}

// ignore: use_key_in_widget_constructors
class ContactPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Hubungi Kami'),
      ),
      body: const Center(
        child: Text('Ini adalah halaman Hubungi Kami'),
      ),
    );
  }
}

// ignore: use_key_in_widget_constructors
class GalleryPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

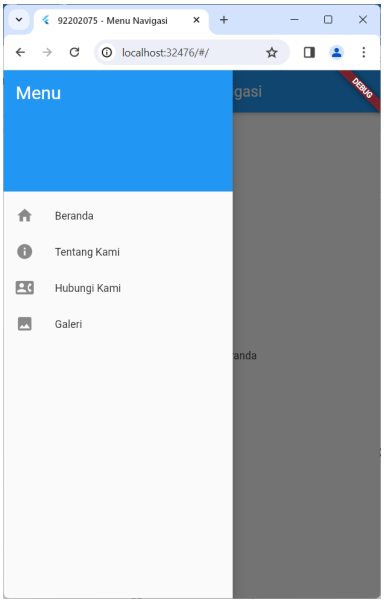
```
      title: const Text('Galeri'),
    ),
    body: const Center(
      child: Text('Ini adalah halaman Galeri'),
    ),
  );
 }
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

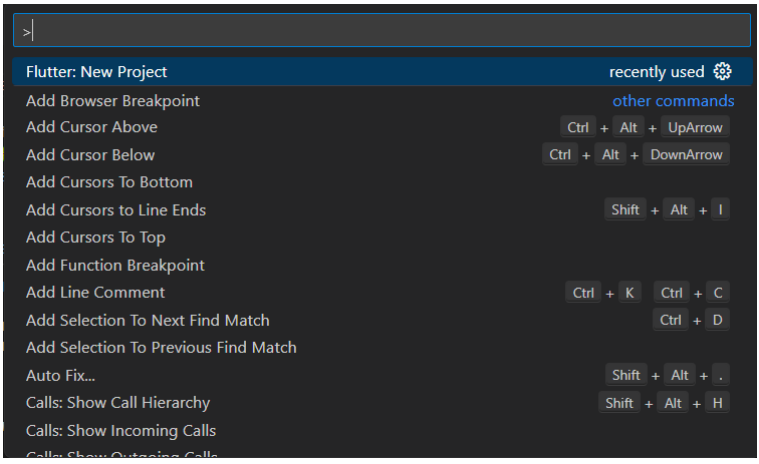
Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



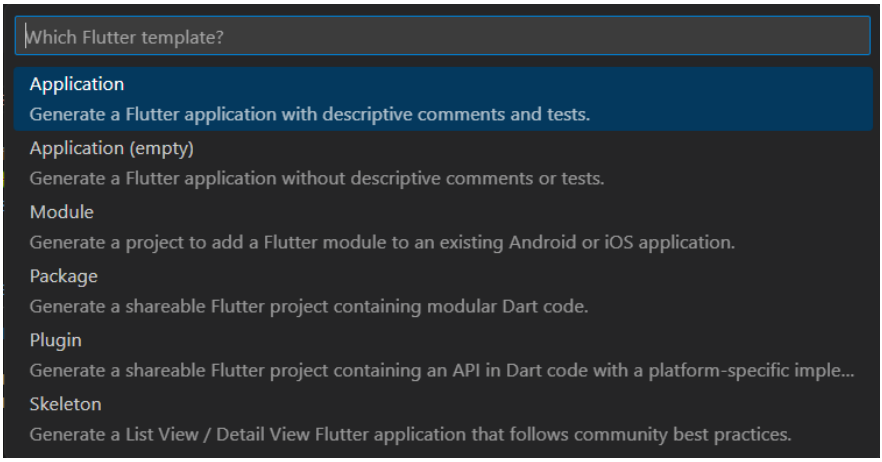


C. Form Input

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

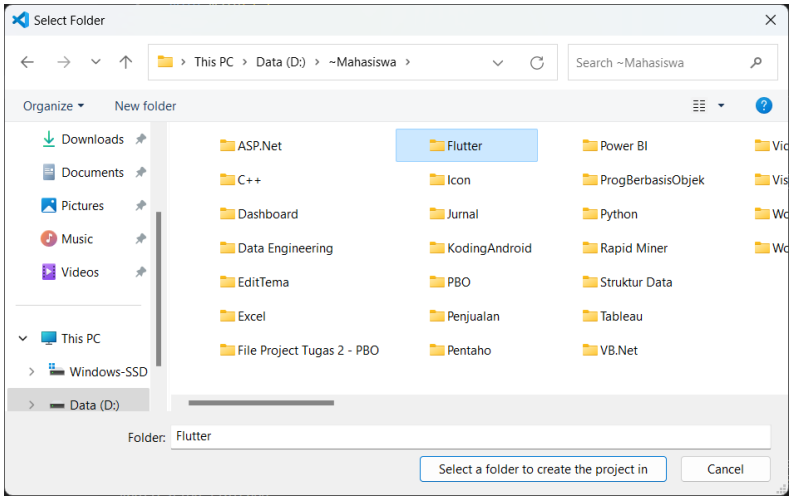


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **form_input** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

void main() {
  runApp(const MaterialApp(
    title: "92202075 - Form Input Lanjutan",
    home: BelajarForm(),
  ));
}

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

  @override
  // ignore: library_private_types_in_public_api
  _BelajarFormState createState() => _BelajarFormState();
}
class _BelajarFormState extends State<BelajarForm> {
 final _formKey = GlobalKey<FormState>();

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: const Text("Arie Gunawan - Form Input"),
     ),
     body: Form(
       key: _formKey,
       child: Container(
         padding: const EdgeInsets.all(20.0),
         child: Column(
           children: [
             // tambahkan komponen seperti input field disini
             TextFormField(
               decoration: const InputDecoration(
```

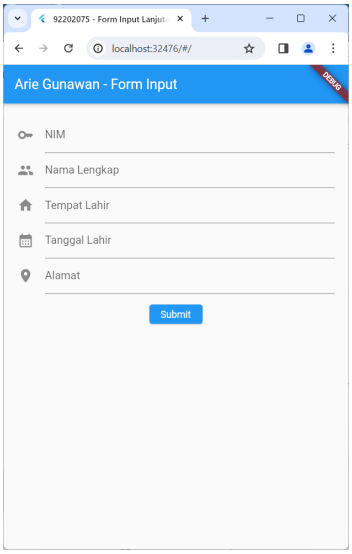
```
      hintText: "Masukan NIM Anda",
      labelText: "NIM",
      icon: Icon(Icons.key)),
  ),
  TextFormField(
   decoration: const InputDecoration(
      hintText: "Masukan Nama Lengkap Anda",
      labelText: "Nama Lengkap",
      icon: Icon(Icons.people)),
  ),
  TextFormField(
   decoration: const InputDecoration(
      hintText: "Masukan Tempat Lahir Anda",
      labelText: "Tempat Lahir",
      icon: Icon(Icons.home)),
  ),
   TextFormField(
    decoration: const InputDecoration(
       hintText: "Masukan Tanggal Lahir Anda",
       labelText: "Tanggal Lahir",
       icon: Icon(Icons.calendar_month)),
   ),
   TextFormField(
    decoration: const InputDecoration(
       hintText: "Masukan Alamat Anda",
       labelText: "Alamat",
       icon: Icon(Icons.room)),
   ),
   Padding(
    padding: const EdgeInsets.symmetric(vertical: 16.0),
    child: ElevatedButton(
     onPressed: () {
```

```
            // Validate will return true if the form is valid, or false if
            // the form is invalid.
            if (_formKey.currentState!.validate()) {
              // Process data.
            }
          },
          child: const Text('Submit'),
        ),
      ),
    ],
  ),
 ),
),
);
}
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

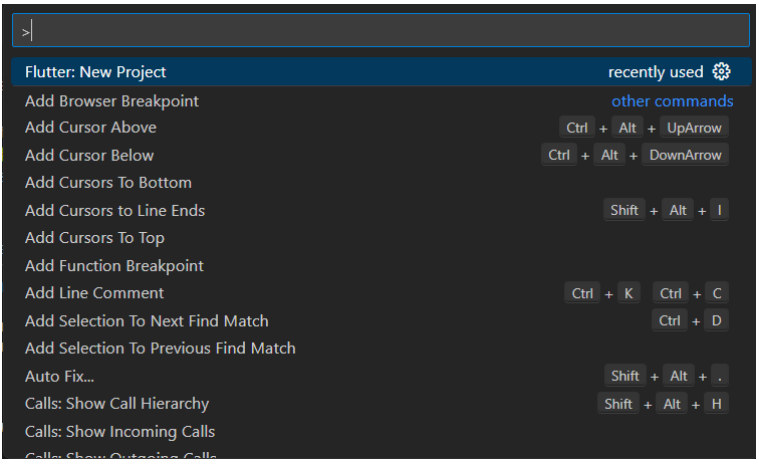
```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

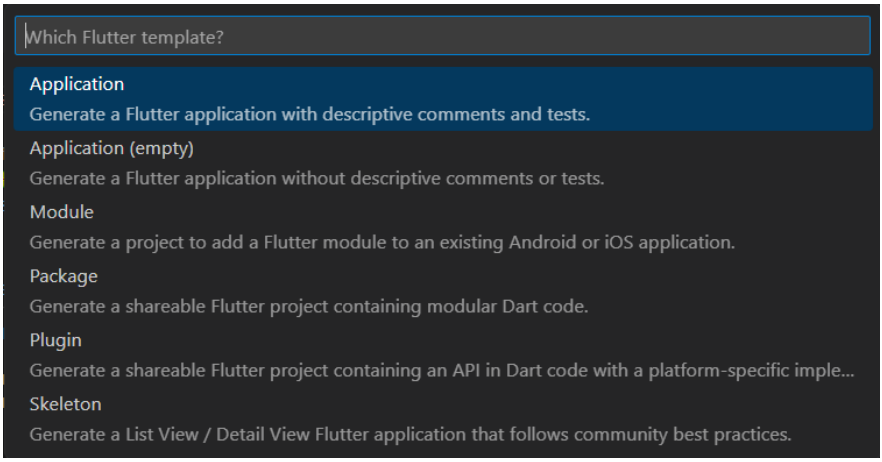


D. Bottom Navigation Bar

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

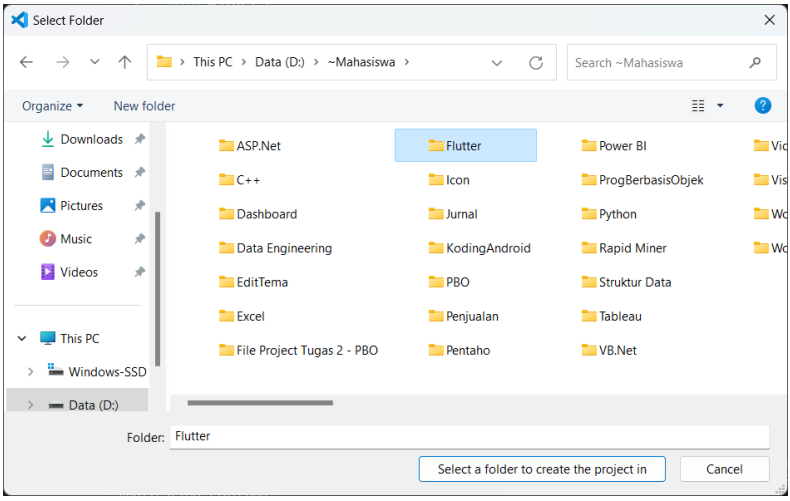


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **bottom_navigation_bar** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

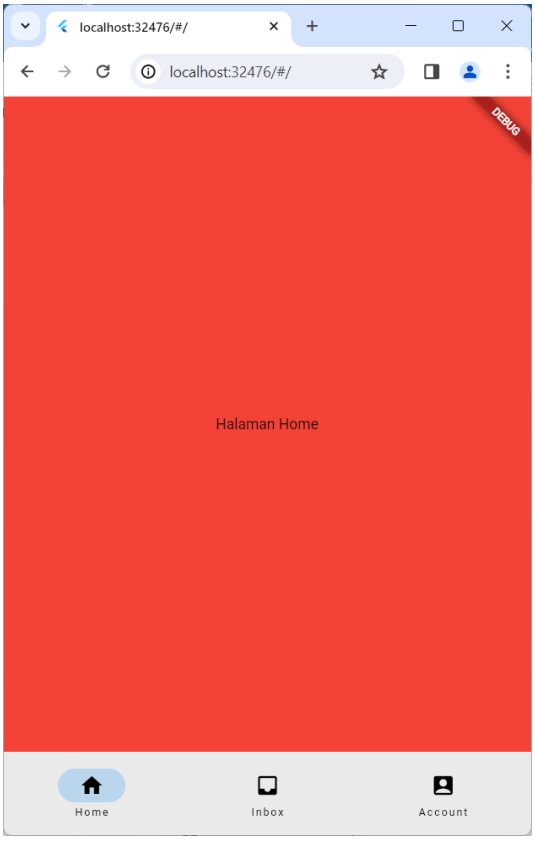
```
import 'package:flutter/material.dart';

void main() => runApp(const ExampleApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(home: NavigationExample());
  }
}

class NavigationExample extends StatefulWidget {
  const NavigationExample({super.key});
  @override
  State<NavigationExample>                createState()                =>
_NavigationExampleState();
}

class _NavigationExampleState extends State<NavigationExample> {
  int currentPageIndex = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      bottomNavigationBar: NavigationBar(
        onDestinationSelected: (int index) {
          setState(() {
            currentPageIndex = index;
          });
        },
```

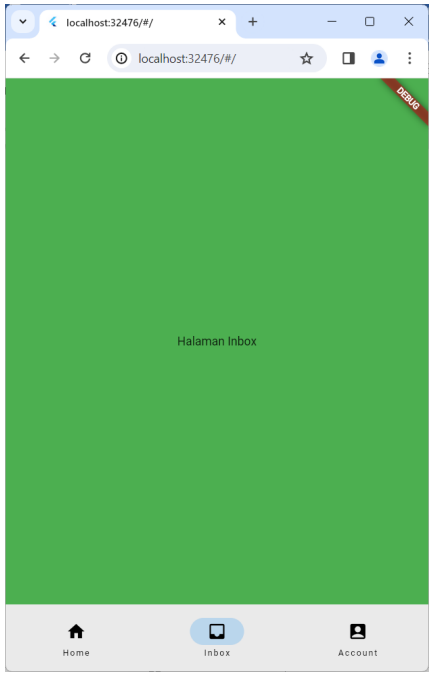
```
      selectedIndex: currentPageIndex,
      destinations: const <Widget>[
        NavigationDestination(
          icon: Icon(Icons.home),
          label: 'Home',
        ),
        NavigationDestination(
          icon: Icon(Icons.inbox),
          label: 'Inbox',
        ),
        NavigationDestination(
          selectedIcon: Icon(Icons.bookmark),
          icon: Icon(Icons.account_box),
          label: 'Account',
        ),
      ],
    ),
    body: <Widget>[
      Container(
        color: Colors.red,
        alignment: Alignment.center,
        child: const Text('Halaman Home'),
      ),
      Container(
        color: Colors.green,
        alignment: Alignment.center,
        child: const Text('Halaman Inbox'),
      ),
      Container(
        color: Colors.blue,
        alignment: Alignment.center,
        child: const Text('Halaman Account'),
      ),
    ][currentPageIndex],
  );
 }
}
```

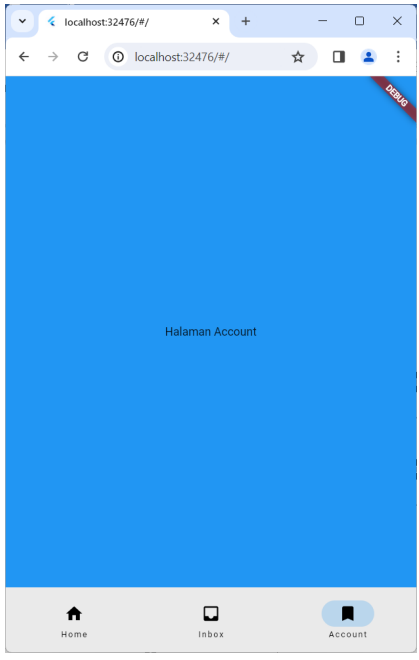
Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

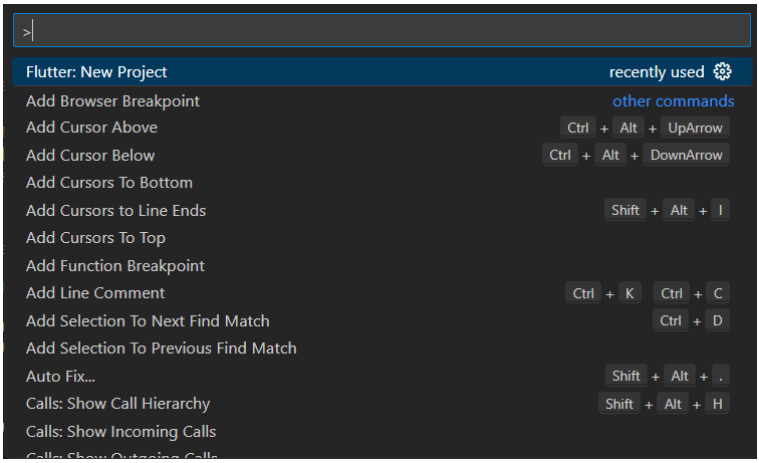



localhost:32476/#/
localhost:32476/#/
DEBUG
Halaman Inbox
Home
Inbox
Account

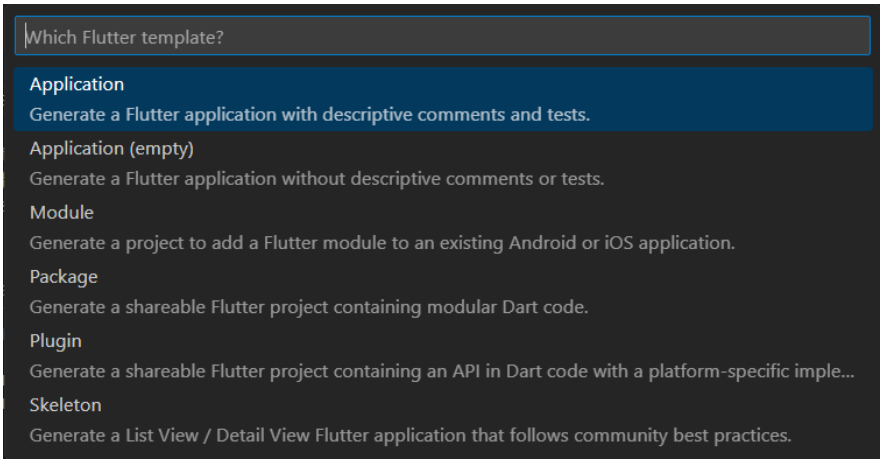

localhost:32476/#/
localhost:32476/#/
DEBUG
Halaman Account
Home
Inbox
Account

E. ListView Nested

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

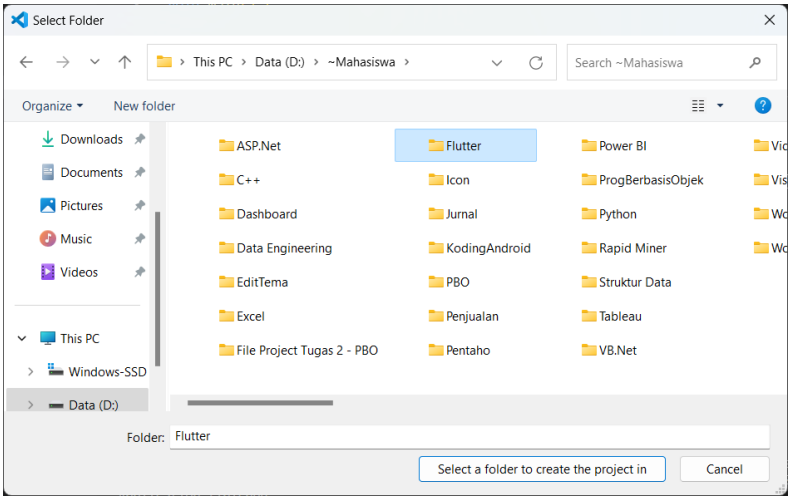


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **list_view_nested** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

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

void main() {
  runApp(const MaterialApp(
    //menghilangkan label debug di kanan atas
    debugShowCheckedModeBanner: false,
    title: "92202075 - List View Nested",
    home: MyApp(),
  ));
}

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  // ignore: library_private_types_in_public_api
```

```
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        // membuat title app bar berada di tengah atas
        title: const Center(child: Text("Arie Gunawan - List View Nested")),
      ),
      //bungkus semua widget di dengan listview terlebih dahulu
      //widget di dalam listview seperti padding, singlechildscrollview, dan
listview
      body: ListView(
        children: [
          const Padding(
            padding: EdgeInsets.only(top: 20, left: 15),
            child: Text("Headline News"),
          ),
          //singlechildscrollview berfungsi untuk membuat widget dapat
discroll
          //fungsinya hampir sama dengan listview
          SingleChildScrollView(
            //untuk merubah arah scroll menjadi ke kanan
            scrollDirection: Axis.horizontal,
            child: Row(
              children: [
                Padding(
                  padding: const EdgeInsets.only(top: 15, left: 15),
                  child: Container(
                    width: 200,
                    height: 150,
                    decoration: const BoxDecoration(
                        color: Colors.grey,
```

```
                image: DecorationImage(
                  image: NetworkImage(

"https://cdn.pixabay.com/photo/2016/10/19/08/57/mountains-
1752433__340.jpg"),
                    fit: BoxFit.cover)),
              ),
            ),
            Padding(
              padding: const EdgeInsets.only(top: 15, left: 15),
              child: Container(
                width: 200,
                height: 150,
                decoration: const BoxDecoration(
                  color: Colors.grey,
                  image: DecorationImage(
                    image: NetworkImage(

"https://cdn.pixabay.com/photo/2016/08/27/14/38/mountains-
1624284__340.jpg"),
                    fit: BoxFit.cover)),
              ),
            ),
          ],
        ),
      ),
      const Padding(
        padding: EdgeInsets.only(top: 15, left: 15),
        child: Text("List All News"),
      ),

      //membuat listview di dalam listview
      //perhatikan beberapa properti di dalamnya untuk menghidnari
error
      ListView(
```

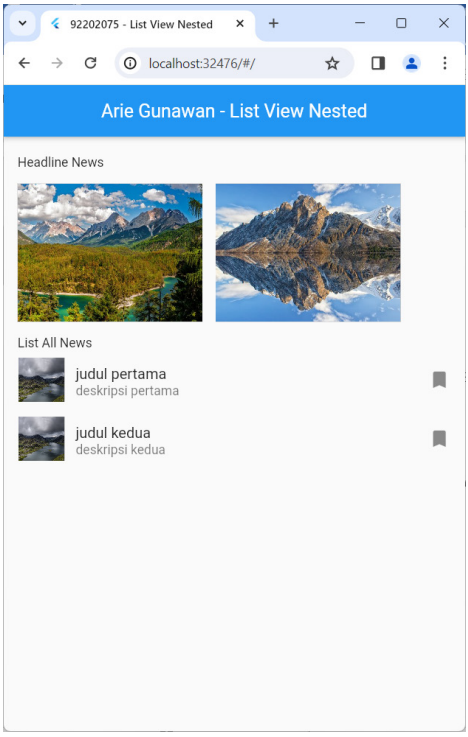
```
        //Jika tidak menyetel properti shrinkWrap, ListView akan sebesar
induknya.
        //Jika menyetelnya ke true, maka besarnya akan menyesuaikan
dengan ukuran content di dalamnya
        shrinkWrap: true,

        //membuat widget tidak dapat discroll sendiri
        //scrolling mengikuti parent
        physics: const NeverScrollableScrollPhysics(),

        children: [
          ListTile(
            leading: Container(
              height: 50,
              width: 50,
              decoration: const BoxDecoration(
                color: Colors.grey,
                image: DecorationImage(
                  image: NetworkImage(

"https://cdn.pixabay.com/photo/2016/11/23/18/29/cloudy-
1854241__340.jpg"),
                  fit: BoxFit.cover)),
            ),
            title: const Text("judul pertama"),
            subtitle: const Text("deskripsi pertama"),
            trailing: const Icon(Icons.bookmark)),
          ListTile(
            leading: Container(
              height: 50,
              width: 50,
              decoration: const BoxDecoration(
                color: Colors.grey,
                image: DecorationImage(
                  image: NetworkImage(
```

```
"https://cdn.pixabay.com/photo/2016/11/23/18/29/cloudy-
1854241__340.jpg"),
                    fit: BoxFit.cover)),
            ),
            title: const Text("judul kedua"),
            subtitle: const Text("deskripsi kedua"),
            trailing: const Icon(Icons.bookmark))
        ],
      ),
    ],
   ),
  );
 }
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

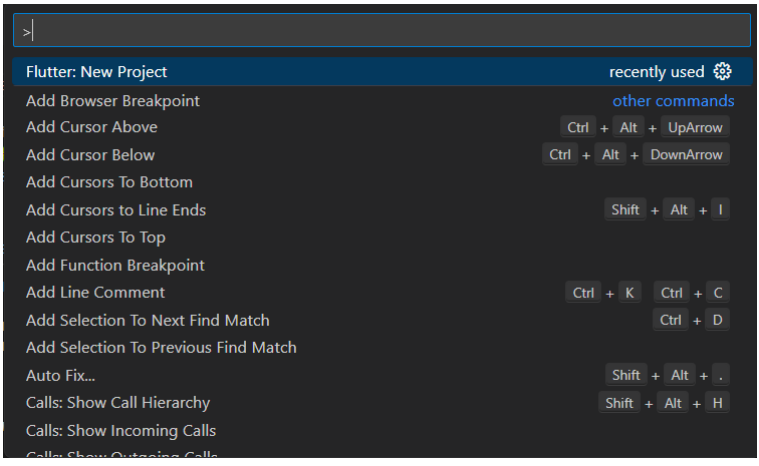
```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

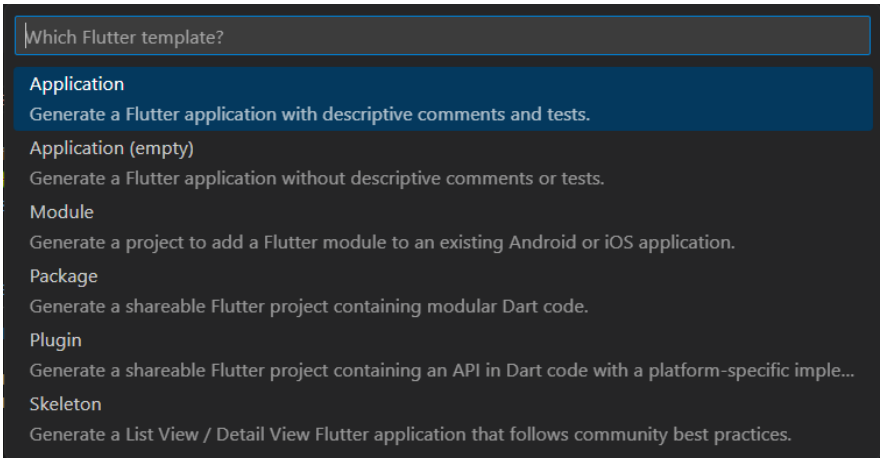


F. Catatan Keuangan Pribadi

Buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

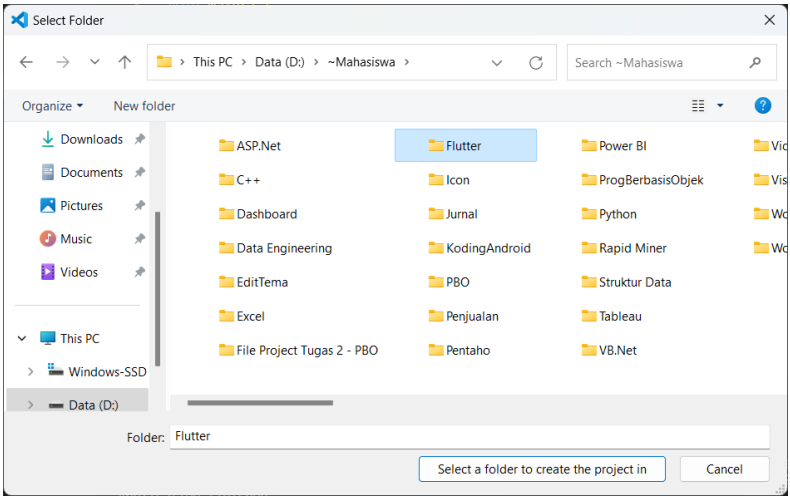


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **catatan_keuangan** kemudian tekan **Enter**.

Buka file **lib/main.dart**, hapus semua code yang ada karena kita akan memulainya dari awal. Kemudian masukkan code berikut:

```
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(
   debugShowCheckedModeBanner: false,
   title: "92202075 - Catatan Keuangan Pribadi",
   home: Scaffold(
    // ignore: avoid_unnecessary_containers
    body: Container(
     //MENGGUNAKAN COLUMN KARENA ADA DUA OBJEK YANG INGIN DIPISAHKAN
     //PERTAMA ADALAH HEADER BESERTA SUMMARYNYA DAN YANG KEDUA ADALAH HISTORY
      child: Column(
       children: <Widget>[
        //KITA GUNAKAN STACK UNTUK MENGATUR POSITION HEADER SERTA CONTAINER UNTUK SUMMARY
        Stack(
         children: <Widget>[backgroundHeader(), summaryCash()],
        ),

        //BAGIAN INI AKAN MENAMPILKAN HISTORY PENGELUARAN
        cardDetail('Makan Siang', 'Beli Makan Di Warung Padang', '15.000',
          'out'),
        cardDetail('Bonus', 'Dapat Hibah Penelitian', '30.000.000', 'in'),
        cardDetail('Beli Baju', 'Baju Kemeja Kantor', '250.000', 'out'),
```

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

 Widget cardDetail(title, description, price, type) {
  //BUAT CARD
  return Card(
   //DENGAN MARGIN YANG DISESUAIKAN
   margin: const EdgeInsets.only(top: 15, left: 15, right: 15),
   //DENGAN KETEBALAN AGAR MEMBENTUK SHADOW SENILAI 8
   elevation: 8,
   //CHILD DARI CARD MENGGUNAKAN LISTTILE AGAR LEBIH MUDAH
 MENGATUR AREANYA
   //KARENA SECARA DEFAULT LISTTILE TELAH TERBAGI MENJADI 3
BAGIAN
   //POSISI KIRI (LEADING), TENGAH (TITLE), BAWAH TENGAH
(SUBTITLE) DAN KANAN(TRAILING)
   //SEHINGGA KITA HANYA TINGGAL MEMASUKKAN TEKS YANG
SESUAI
   child: ListTile(
    leading: Icon(
     type == 'out'
        ? Icons.subdirectory_arrow_left
        : Icons.subdirectory_arrow_right,
     color: type == 'out' ? Colors.redAccent : Colors.lightGreen,
    ),
    title: Text(
     title,
     style: const TextStyle(fontWeight: FontWeight.bold),
    ),
```

```
    subtitle: Text(description),
    trailing: Text(
     // ignore: prefer_interpolation_to_compose_strings
     "Rp " + price,
     style: TextStyle(
        color: type == 'out' ? Colors.redAccent : Colors.lightGreen),
    ),
   ),
  );
}

Widget summaryCash() {
 //CONTAINER KEDUA INI BERWARNA PUTIH, KITA SET POSITIONEDNYA DENGAN MENENTUKAN VALUE DARI TOP DAN LEFT AGAR BERADA DITENGAH, DISESUAIKAN SAJA
 return Positioned(
  top: 180,
  left: 20,
  //CONTAINER KEDUA INI KITA BUAT LEBIH KECIL DENGAN MENENTUKAN WIDTH DAN HEIGHNYA TERBATAS
  child: Container(
   width: 370,
   height: 140,
   //SAMA HALNYA DENGAN CONTAINER SEBELUMNYA, WARNANYA DI-SET DAN BORDERRADIUSNYA KALI INI BERBEDA KITA SET KE-4 SISINYA AGAR MELENGKUNG
   decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(30),
   ),
   //CHILD DARI CONTAINER INI DI-SET PADDINGNYA AGAR TERDAPAT JARAK DARI ATAS
   child: Padding(
    padding: const EdgeInsets.only(top: 30.0),
    //KARENA ADA DUA BAGIAN YANG BERBARIS DARI KIRI KE KANAN MAKA KITA GUNAKAN ROW()
```

```
      child: Row(
       //MAIN ALIGMENTNYA DI-SET SPACEAROUND AGAR KEDUA
OBJEKNYA ADA JARAK YANG SESUAI
       mainAxisAlignment: MainAxisAlignment.spaceAround,
       crossAxisAlignment: CrossAxisAlignment.start,
       children: <Widget>[
        //MASING-MASING OBJECT MENGGUNAKAN COLUMN LAGI ADA
DUA BUAH TEKS YANG INGIN DITAMPILKAN SECARA VERTICAL
        Column(
         children: const <Widget>[
          Text("Penghasilan"),
          Divider(),
          Text(
           "Rp 30.000.000",
           style: TextStyle(fontSize: 25, fontWeight: FontWeight.bold),
          ),
         ],
        ),
        Column(
         children: const <Widget>[
          Text("Pengeluaran"),
          Divider(),
          Text(
           "Rp 265.000",
           style: TextStyle(fontSize: 25, fontWeight: FontWeight.bold),
          ),
         ],
        ),
       ],
      ),
     ),
    ),
   );
}
```

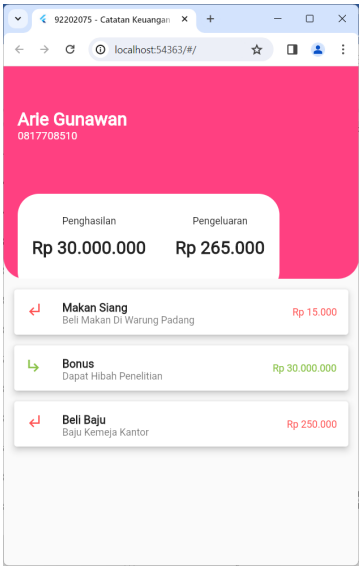
```
Widget backgroundHeader() {
 //KITA BUAT CONTAINER DENGAN TINGGI SEBESAR 300, DAN LEBAR SEJAUH YANG BISA DIJANGKAU
 //BOXDECORATIONNYA KITA SET WARNANYA PINKACCENT DAN PADA BAGIAN BAWAH KIRI-KANAN DIBUAT LENGKUNGAN
 return Container(
  height: 300,
  width: double.infinity,
  decoration: const BoxDecoration(
   color: Colors.pinkAccent,
   borderRadius: BorderRadius.only(
    bottomLeft: Radius.circular(30),
    bottomRight: Radius.circular(30),
   ),
  ),
  //ADAPUN CHILD DARI CONTAINER DIATAS KITA ATUR POSISINYA MENGGUNAKAN PADDING
  child: Padding(
   //PADDINGNYA DI-SET HANYA UNTUK TOP DAN LEFT
   padding: const EdgeInsets.only(top: 60, left: 20),
   //KARENA KITA AKAN MENAMPILKAN LEBIH DARI 1 OBJEK YANG BERUSUSUN KEBAWAH
   //MAKA KITA GUNAKAN COLUMN
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: const <Widget>[
     //BAGIAN INI NORMAL, HANYA MENAMPILKAN TEXT DENGAN STYLE MASING-MASING
      Text(
       "Arie Gunawan",
       style: TextStyle(
          fontSize: 25, color: Colors.white, fontWeight: FontWeight.bold),
      ),
      Text(
       "0817708510",
       style: TextStyle(
```

```
            fontSize: 15,
            color: Colors.white,
          ),
        ),
      ],
    ),
  ),
 );
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

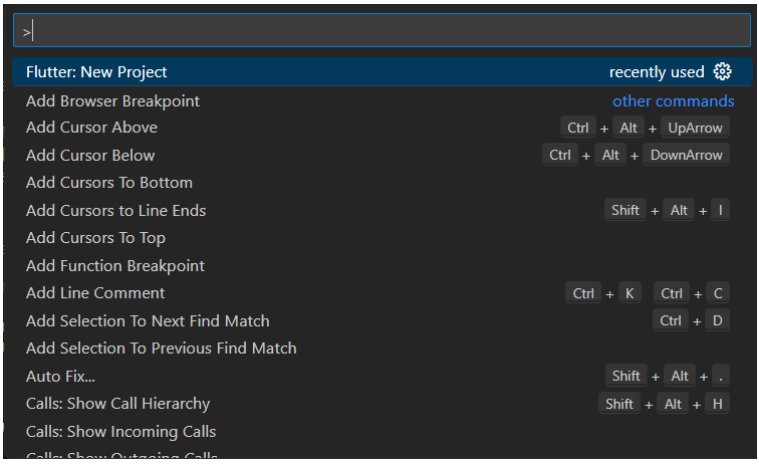
```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.

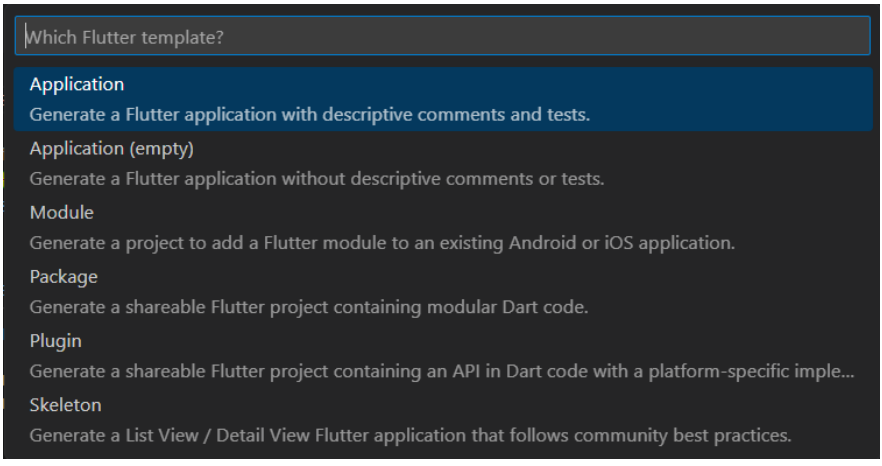


G. CRUD Data Pegawai

Bagian pertama yang akan kita kerjakan adalah fitur untuk menampilkan seluruh data pegawai, sebelum memulainya, buat project baru menggunakan **Command Pallete**. Pilih menu **View** dan klik **Command Pallete** sehingga muncul tampilan berikut.

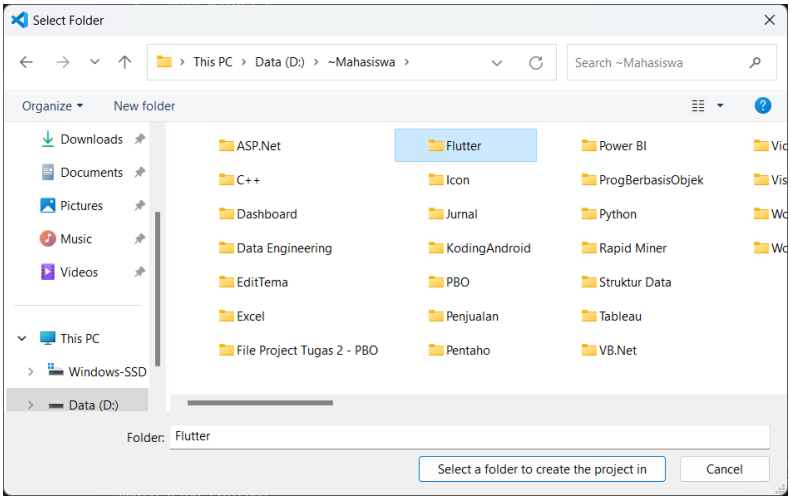


Kemudian ketikkan flutter, pilih **Flutter New Project**, maka akan muncul tampilan berikut.



Pilih menu **Application**.

Pilih folder yang diinginkan, kemudian klik tombol **Select a folder to create the project in**.



Beri nama projectnya, dengan mengetik **crud_data_pegawai** kemudian tekan **Enter**.

Sebagaimana kita ketahui, file pertama yang akan di-load Flutter adalah **main.dart**, jadi buka file **lib/main.dart** dan modifikasi menjadi.

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Employee(),
    );
  }
}
```

Untuk meng-handle tampilan dari data pegawai, dimana kita akan menampilkan beberapa informasi, diantaranya, nama pegawa, gaji dan

umur, maka buat file baru bernama **employee.dart** di dalam folder **lib/pages** dan tambahkan struktur berikut

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

class Employee extends StatelessWidget {
 //DUMMY DATA YANG AKAN DITAMPILKAN SEBELUM MELAKUK
HIT KE API
 //ADAPUN FORMAT DATANYA MENGIKUTI STRUKTU YANG SUD
DITETAPKAN PADA EMPLOYEEMODEL
 final data = [
  EmployeeModel(
   id: "1",
   employeeName: "Deny Hidayatullah",
   employeeSalary: "320800",
   employeeAge: "54",
   profileImage: "",
  ),
  EmployeeModel(
   id: "2",
   employeeName: "Arie Gunawan",
   employeeSalary: "40000",
   employeeAge: "45",
   profileImage: "",
  ),
  EmployeeModel(
   id: "3",
   employeeName: "Sari Ningsih",
   employeeSalary: "120000",
```

```
    employeeAge: "60",
    profileImage: "",
   ),
   EmployeeModel(
    id: "4",
    employeeName: "Dhieka A. Lantana",
    employeeSalary: "28000",
    employeeAge: "37",
    profileImage: "",
   ),
  ];

  Employee({super.key});

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text('Karyawan CRUD'),
    ),
    //ADAPUN UNTUK LOOPING DATA PEGAWAI, KITA GUNAKAN
LISTVIEW BUILDER
    //KARENA WIDGET INI SUDAH DILENGKAPI DENGAN FITUR
SCROLLING
    body: ListView.builder(
     itemCount: data.length, //MENGHITUNG JUMLAH DATA YANG AKAN
DITAMPILKAN
     //LOOPING DATA
     itemBuilder: (context, i) {
      //KEMUDIAN TAMPILKAN DATA PEGAWAI BERDASARKAN INDEX
YANG DISIMPAN DI DALAM VARIABLE I
      return Card(
       elevation: 8,
       child: ListTile(
        title: Text(
```

```
            data[i].employeeName,
            style:
                const TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
          ),
          subtitle: Text('Umur: ${data[i].employeeAge}'),
          trailing: Text("\$${data[i].employeeSalary}"),
        ),
      );
    },
  ),
 );
 }
}
```

Adapun untuk formatting struktur data yang diinginkan, buat file employee_model.dart di dalam folder lib/models dan tambahkan code berikut

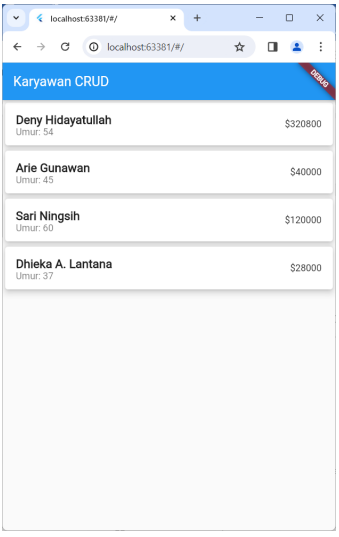
```
class EmployeeModel {
  String id;
  String employeeName;
  String employeeSalary;
  String employeeAge;
  String profileImage;

  //BUAT CONSTRUCTOR AGAR KETIKA CLASS INI DILOAD, MAKA DATA YANG DIMINTA HARUS DIPASSING SESUAI TIPE DATA YANG DITETAPKAN
  EmployeeModel(
     {required this.id,
     required this.employeeName,
     required this.employeeSalary,
     required this.employeeAge,
     required this.profileImage});
```

```
 //FUNGSI INI UNTUK MENGUBAH FORMAT DATA DARI JSON KE FORMAT YANG SESUAI DENGAN EMPLOYEE MODEL
 factory EmployeeModel.fromJson(Map<String, dynamic> json) => EmployeeModel(
    id: json['id'],
    employeeName: json['employee_name'],
    employeeSalary: json['employee_salary'],
    employeeAge: json['employee_age'],
    profileImage: json['profile_image']);
}
```

Untuk menjalankan koding, menggunakan terminal. Klik **New Terminal**. Kemudian ketikkan **flutter run**, tekan **Enter**. Pilih salah emulator yang akan digunakan untuk menjalankan project. Disini penulis menggunakan Chrome, berarti ketik angka 2.

```
PS D:\~Mahasiswa\Flutter\aplikasipertama> flutter run
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.1413]
Chrome (web)      • chrome  • web-javascript • Google Chrome 111.0.5563.147
Edge (web)        • edge    • web-javascript • Microsoft Edge 111.0.1661.62
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Jika tidak ada masalah, maka hasilnya akan tampil seperti berikut.



DAFTAR PUSTAKA

Form class–widgets library–Dart API. (2024). In *api.flutter.dev*. https://api.flutter.dev/flutter/widgets/Form-class.html

Sahretech. (2024). In *Sahretech*. https://www.sahretech.com/search/label/Flutter?updated-max=2022-02-07T06%3A58%3A00-08%3A00&max-results=6#PageNo=5

wpOceans. (2020). Kategori Flutter–Daeng Web. In *Kategori Flutter–Daeng Web*. https://daengweb.id/kategori/flutter

BIOGRAFI PENULIS

Arie Gunawan, S.Kom., M.M.S.I

Lahir di kota Jakarta pada tanggal 10 April 1978. Penulis adalah Dosen di Fakultas Teknologi Komunikasi dan Informatika Program Studi Sistem Informasi Universitas Nasional. Menamatkan pendidikan program Sarjana (S1) di Universitas Gunadarma Jakarta prodi Sistem Informasi dan menyelesaikan program Pasca Sarjana (S2) di Universitas Gunadarma prodi Sistem Informasi. Kemudian penulis menyelesaikan kuliah S3 di Universitas Pendidikan Bandung, Jawa Barat mengambil konsentrasi Sistem Informasi Manajemen. Sebelum menjadi dosen pada tahun 2019, penulis sebelumnya adalah seorang praktisi yang bekerja di beberapa perusahaan, yaitu:

- PT Nusantara Surya Sakti sebagai MIS Dept Head
- PT Karunia Abadi Mandiri Persada sebagai IT Manager
- PT Semesta Finance sebagai IT Supervisor

Penulis memiliki keahlian dalam bidang:

- Database: MS SQL Server 2005/2008/2014, MySQL
- Programming: Visual Basic 6.0, Visual Basic.Net, ASP.Net VB, Java NetBeans, Android Studio, C++, Python, HTML 5, Bootstrap, CSS, AJAX Toolkit
- ETL Tools: SSIS (SQL Server Integration Services), Pentaho, Talend, DTS (Data Transformation System)
- BI Tools: Kyubit, Tableau, PowerBI