Modul Praktikum
Flutter

Bagian Dua: UI & Routing (part 2)

Yudi Wibisono (yudi@upi.edu)
Ilmu Komputer, Universitas Pendidikan Indonesia (cs.upi.edu)

versi: Beta; Feb 2024

FullScreen Dialog

Fullscreen dialog sesuai namanya, menampilkan dialog yang menutup semua layar.

Fullscreen dialog seperti halnya alertdialog, menggunakan showDialog()

Jadi code untuk alert dialog di atas dapat diganti bagian tampilanDialog(), builder yang tadinya menghasilkan alertdialog diganti dengan Dialog.fullscreen yang memiliki atribut child. Atribut ini berisi widget yang akan kita tampilkan dalam dialog. Karena sifatnya fullscreen, kita memiliki banyak tempat, tapi hati-hati dapat membingungkan user karena navigasi utama akan tertutup.

```
void tampilkanDialog(BuildContext context) {
 showDialog<void>(
   context: context,
   builder: (BuildContext context) => Dialog.fullscreen(
     child: Center(
         child: Column(
       mainAxisSize: MainAxisSize.min,
       children: [
         Padding(
             padding: EdgeInsets.all(20), child: Text("Dialog Fullscreen")),
         ElevatedButton(
           onPressed: () => Navigator.pop(context, 'Cancel'),
           child: const Text('Tutup'),
         ),
       ],
     )), //column cent
   ),
 );
  }
```

Saat button ditap, maka akan muncul dialog yang menutupi semua app.

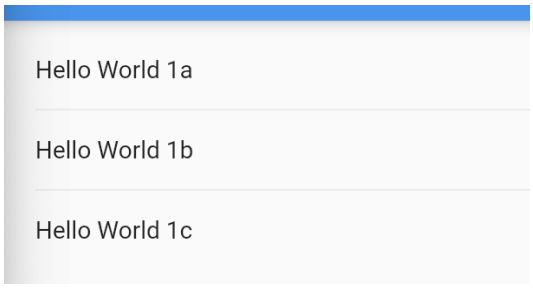
ListView

Listview meletakkan informasi dalam bentuk baris yang dapat di-scroll. Ini widget penting untuk aplikasi mobile karena dapat menampilkan informasi secara vertikal dan dapat di-scroll.

Sebagai contoh:

```
child: ListView(padding: const EdgeInsets.all(20), children: const [
            Text('Hello World 1a'),
            Divider(
              height: 30,
            ),
            Text('Hello World 1b'),
            Divider(
              height: 30,
            ),
            Text('Hello World 1c'),
          ]),
```

Hasilnya:



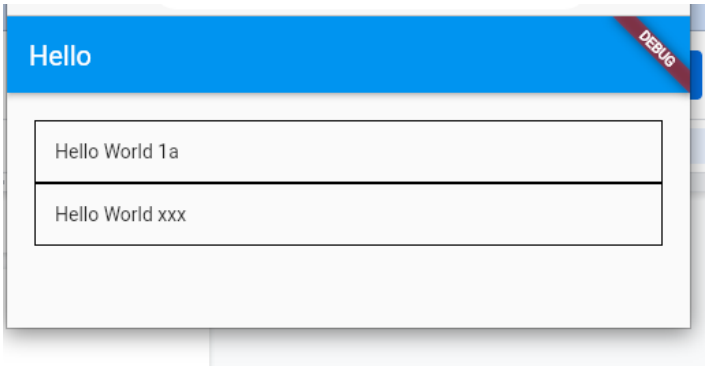
Code lengkap: https://pastebin.com/raw/4NMR2T0B

Contoh lain yang menggunakan widget Container di dalam ListView

```
body: Center(
          child: ListView (
            padding: EdgeInsets.all(20),
            children: [
                  Container(
                      decoration: BoxDecoration(border: Border.all()),
                      padding: EdgeInsets.all(14),
                      child: Text('Hello World 1a')),
                  Container(
                      decoration: BoxDecoration(border: Border.all()),
                      padding: EdgeInsets.all(14),
                      child: Text('Hello World xxx')),
```

```
                ]
            ),
        ),
```

Hasilnya:



Source: https://pastebin.com/raw/r3KU54UD

Selain container seperti contoh ini, dapat digunakan juga ListTile dan Card.

ListView Builder()

Jika kita akan menampilkan 100 baris data dinamik, cara sebelumnya tidak praktis lagi. Untuk jumlah row yang besar, dapat digunakan ListView Builder.

Berikut contoh penggunaan listview builder

Pertama siapkan isi dari ListView-nya yang disimpan dalam sebuah list.

```
@override
void initState() {
  super.initState();
  for (int i = 0; i < 20; i++) {
         data.add("Data ke $i ");
  }
}
```

Selanjutnya gunakan method ListView.builder. `Itemcount` berisi jumlah baris. Itembuilder berisi parameter index row yang akan "digambar".

```
child: ListView.builder(
          itemCount: data.length, //jumlah baris
          itemBuilder: (context, index) {
```

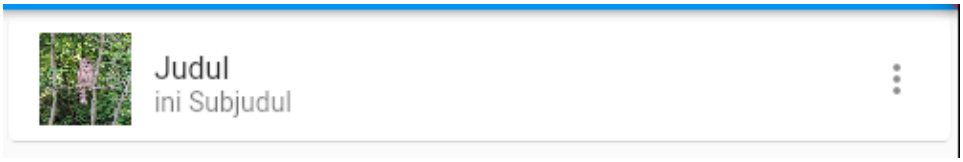
```
            return Container(
                 decoration: BoxDecoration(border: Border.all()),
                 padding: EdgeInsets.all(14),
                 child: Text(data[index]), //akses melalui indeks
...
```

Hasil dari code ini adalah

Code lengkap: https://pastebin.com/raw/628tU7Vv

Latihan 6

Gunakan Card untuk menampilkan foto, nama dan hobi (gambar bawah) menggunakan Listview.Builder. Judul jadi nama, sub judul adalah hobi. Nama dan hobi disimpan dalam bentuk Map<String, String>



Penanganan Tap pada Baris ListView

Untuk menangani tap pada baris listview, gunakan ListTile yang memiliki event onTap(). Contoh berikut menampilkan data yang di-tap dalam bentuk snackbar.

```
child: ListView.builder(
        itemCount: data.length,
        itemBuilder: (context, index) {
          return Card(
             child: ListTile(
             onTap: () {
               ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                     duration: Duration(seconds: 1),
                     content: Text('Halo ${data[index]}'),
               ));
              },
              title: Text(data[index]),
            ));
          },
        ),
```

Layout

Beberapa widget seperti Button tidak memiliki informasi posisi. Flutter menyediakan widget layout yang dapat digunakan mengatur posisi, ukuran, style dan pengelolaan posisi child widget.

Berikut kita akan bahas beberapa widget terkait posisi dan layout.

Container

Container memiliki peranan penting dalam layout. Container dapat digunakan untuk menambahkan padding, border (garis), margin, posisi dan ukuran widget. Container memiliki atribut child, heigh, width, padding, margin, decoration. Decoration berisi background image, background color dan border.

Sebagai contoh, jika Container digunakan pada program hello world yang pertama:

```
. . .
home: Scaffold(
        appBar: AppBar(
          title: const Text('Hello'),
        ),
        body: Container(
          decoration: BoxDecoration(
              color: const Color(0xff7c94b6),
              borderRadius: BorderRadius.circular(12),
              border: Border.all()),
          padding: EdgeInsets.all(14),
          child: Text('Hello World 2!'),
        ),
      ),

. . .
```

hasilnya adalah

Untuk mengatur ukuran garis border dan warna, dapat digunakan width dan color: `Border.all(width: 5, color: Colors.red))`

Code lengkap: https://pastebin.com/raw/XYNGDQ9F

MediaQuery dapat digunakan mengatur ukuran container agar mengikuti secara relatif dengan ukuran layar. Sebagai contoh, code berikut akan membuat ukuran Container selalu 0.65 dari ukuran layar dan akan otomatis berubah saat layar di-rotate atau diresize (jika platform-nya web).

```
Container(
        width: MediaQuery.of(context).size.width * 0.65,
. . .
```

Untuk membuat ukuran container mengisi penuh layar, dapat digunakan:

```
Container(
        width:double.infinity,
```

Catatan: **Container vs SizedBox**
keduanya mirip, Container dapat mengatur shape, padding, margin, decoration, alignment sedangkan SizedBox hanya ukuran saja. Untuk whitespace (ruang kosong), dianjurkan menggunakan SizedBox karena lebih ringan dibandingkan Container.

Align

Widget Align dapat digunakan untuk mengatur alignment **satu** widget. Jika sebelumnya kita sudah menggunakan Center, Align memberikan beberapa opsi lain seperti bottomLeft, bottomRight, topCenter dsb.

Sebagai contoh:

```
Align(
     alignment: Alignment.bottomLeft,
     child: Container(
                    decoration: BoxDecoration(border: Border.all()),
                    child: Text('Hello World'))
)
```

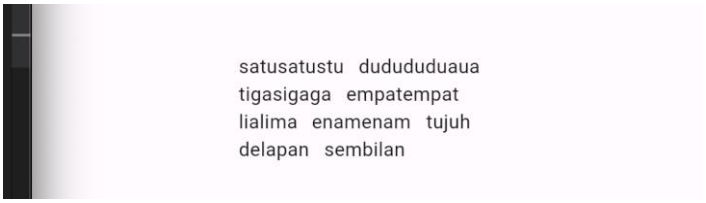
Wrap

Wrap adalah widget yang meletakkan widget childrennya secara horizontal, tetapi jika tidak cukup maka akan diletakkan di baris berikutnya.

Contoh:

```
Container(
                width: 200,
                child: const Wrap(
                  spacing: 10,
                  children: [
                    Text("satusatustu"),
                    Text("dudududuaua"),
                    Text("tigasigaga"),
                    Text("empatempat"),
                    Text("lialima"),
                    Text("enamenam"),
                    Text("tujuh"),
                    Text("delapan"),
                    Text("sembilan"),
                  ],
                ))
```

Hasilnya



Row dan Column

Row dan Column adalah widget yang paling umum digunakan untuk layout. Widget ini memiliki atribut children yang berisi widget-widget yang akan berada di bawah Row atau Column. Pada contoh sebelumnya kita telah menggunakan widget Column, berikut code untuk contoh Row:

. . .

```
 body: Center(
          child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
            Container(
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Text('Hello World 1!')),
            Container(
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Text('Hello World 2!')),
            Container(
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Text('Hello World 3!')),
          ]),
.  . .
```

Hasilnya:

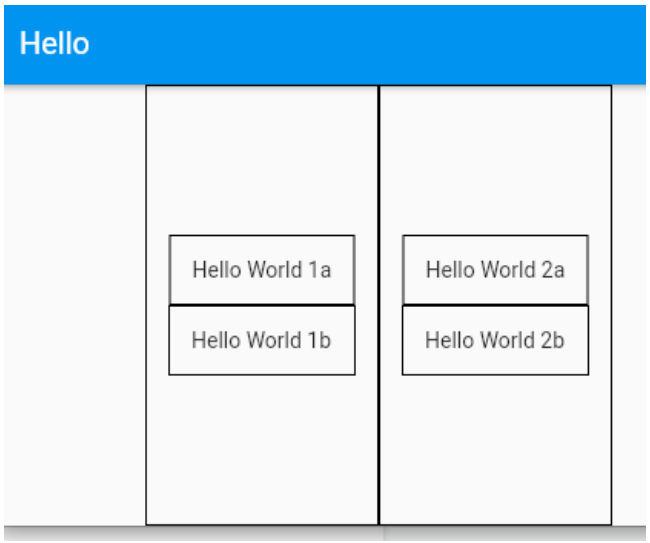
Source: https://pastebin.com/raw/qyTMuN59

Berikut adalah contoh kombinasi antara Row dan Column:

```
 body: Center(
          child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
            Container(
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      Container(
                          decoration: BoxDecoration(border: Border.all()),
                          padding: EdgeInsets.all(14),
                          child: Text('Hello World 1a')),
                      Container(
                          decoration: BoxDecoration(border: Border.all()),
                          padding: EdgeInsets.all(14),
                          child: Text('Hello World 1b')),
                    ])),
            Container(
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Column(
```

```
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Container(
                      decoration: BoxDecoration(border: Border.all()),
                      padding: EdgeInsets.all(14),
                      child: Text('Hello World 2a')),
                  Container(
                      decoration: BoxDecoration(border: Border.all()),
                      padding: EdgeInsets.all(14),
                      child: Text('Hello World 2b')),
                ])),
          ]),
        ),
```

Hasilnya (penggunaan border membantu untuk men-debug layout):



Code lengkap: https://pastebin.com/raw/e5xsmXbt

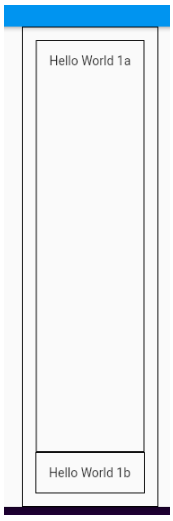
Expanded

Widget expanded digunakan agar widget yang berada di dalam Row dan Column dapat mengisi penuh parentnya.

Sebagai contoh, jika kita modifikasi contoh code sebelumnya (Column di dalam Row)

```
child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                Expanded(
                      child: Container(
                          decoration: BoxDecoration(border: Border.all()),
                          padding: const EdgeInsets.all(14),
                          child: Text('Hello World 1a')),
                    ),
                    Container(
                        decoration: BoxDecoration(border: Border.all()),
                        padding: const EdgeInsets.all(14),
                        child: const Text('Hello World 1b')),
                ])
```

Maka hasilnya adalah sebagai berikut, terlihat container di bagian atas mengisi penuh baris:



Code lengkap: https://pastebin.com/raw/0RhAssWt

Jika ada widget yang menghilang di dalam column (misalnya ListView), salah satu solusinya adalah membungkus widget tersebut dengan expanded.

MainAxisAlignment dan CrossAxisAlignment

mainAxisAlignment dan crossAxisAlignment digunakan untuk mengatur bagaimana row dan column mengelola posisi widget child.

Pada widget Row, atribut mainAxis mengatur axis **horizontal**. Sebagai contoh, jika kita modifikasi code sebelumnya (https://pastebin.com/raw/qyTMuN59) dan mengganti mainAxisAlignment menjadi spaceEvenly maka hasilnya:

```
Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
        Container(
              decoration: BoxDecoration(border: Border.all()),
              padding: EdgeInsets.all(14),
              child: Text('Hello World 1!')),

….
```

dapat dilihat berdasarkan axis horizontal (mainAxis), widget child didistribusikan secara merata:

Selain `MainAxisAlignment.center` dan `spaceEvenly`, terdapat `spaceBetween`, `spaceAround`. Perbandingan semua jenis dapat dilihat pada gambar berikut:

SpaceBetween:

SpaceAround:

SpaceEvenly:

Catatan: perbedaan evently dengan around, pada evenly jarak kiri-kanan layar juga disamakan dengan jarak antar widget

Center:

Selain itu `MainAxisAlignmen.start` digunakan untuk merapatkan ke kiri dan dan `MainAxisAlignment.end` ke kanan (untuk horizontal). Jika vertikal, start berada di atas dan end berada di bawah.

Pada widget Row, crossAxisAlignment adalah axis **vertikal**.

mainAxis pada Widget Column terbalik dengan Row. mainAxis Column adalah vertikal sedangkan crossAxis horizontal.

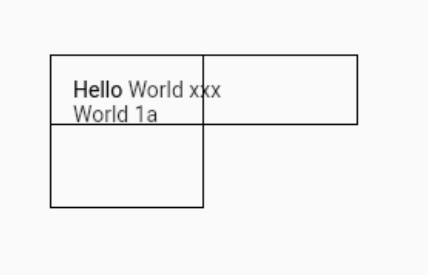
Stack

Stack mengatur widget di bawahnya dalam bentuk layer, sehingga widget dapat diatur bertumpuk.

Sebagai contoh:

```
body: Center(
          child: Stack(children: [
            Container(
                width: 100,
                height: 100,
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Text('Hello World 1a')),
            Container(
                width: 200,
                decoration: BoxDecoration(border: Border.all()),
                padding: EdgeInsets.all(14),
                child: Text('Hello World xxx')),
          ]),
        ),
```

Hasilnya:



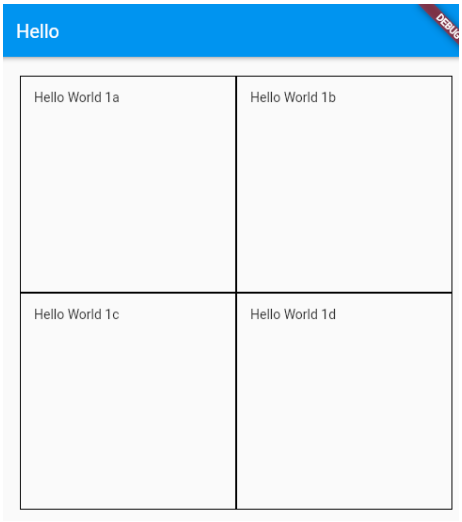
Source: https://pastebin.com/raw/25rBWV7L

GridView

Gridview menyusun widget dalam bentuk array dua dimensi yang dapat di-scroll. Sebagai contoh:

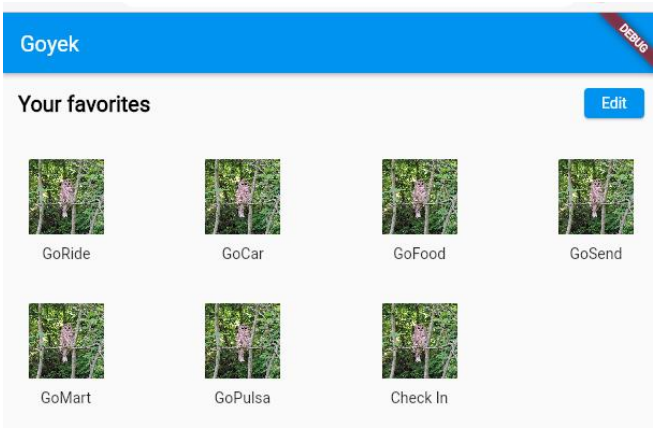
```
body: Center(
          child: GridView(
              padding: const EdgeInsets.all(20),
              gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
              ),
              children: [
                Container(
                    decoration: BoxDecoration(border: Border.all()),
                    padding: EdgeInsets.all(14),
                    child: Text('Hello World 1a')),
                Container(
                    decoration: BoxDecoration(border: Border.all()),
                    padding: EdgeInsets.all(14),
                    child: Text('Hello World 1b')),
                Container(
                    decoration: BoxDecoration(border: Border.all()),
                    padding: EdgeInsets.all(14),
                    child: Text('Hello World 1c')),
                Container(
                    decoration: BoxDecoration(border: Border.all()),
                    padding: EdgeInsets.all(14),
                    child: Text('Hello World 1d')),
              ]),
        ),
```

Hasilnya:

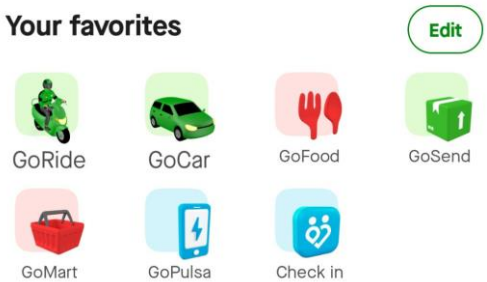


Source code: https://pastebin.com/raw/S0hQy3gt

Latihan 7

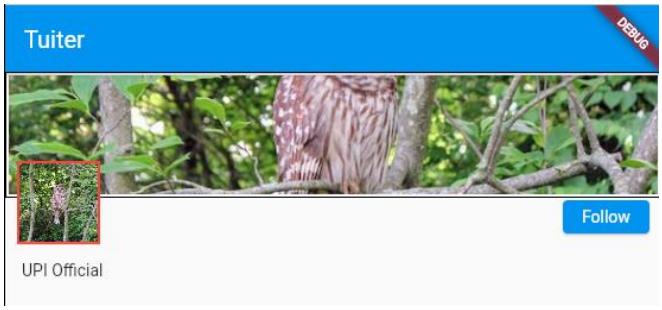


Bonus, buat semirip ini:



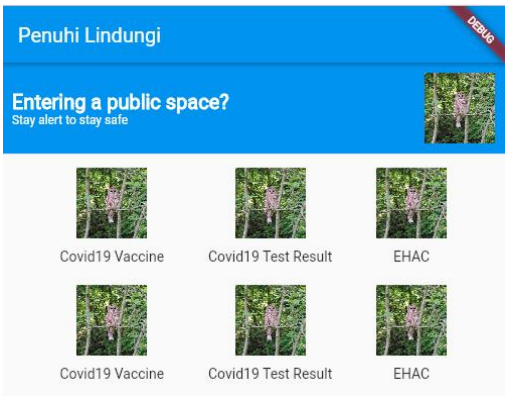
Latihan 8

Petunjuk: Gunakan atribut BoxFit pada image dan `width:double.infinity` untuk container

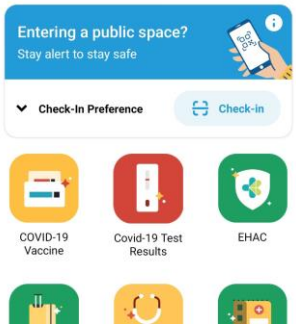


Bonus, buat semirip ini:

Latihan 9



Bonus, buat semirip ini:



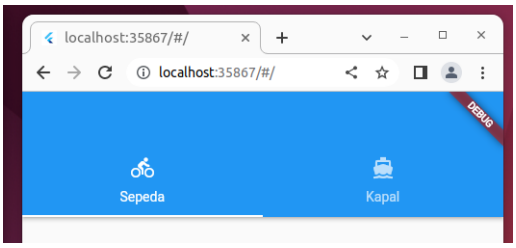
Widget Navigasi

Tab

Tab dibuat dengan TabController, TabBar dan TabView, penggunaannya dapat dilihat pada gambar di bawah:

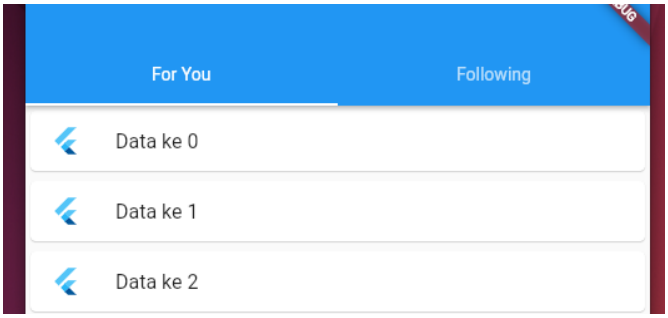
```
...
  @override
  Widget build(BuildContext context) {
  return MaterialApp(
        home: DefaultTabController(
          length: 2,
          child: Scaffold(
              appBar: AppBar(
                  title: const Text('Fakultas'),
                  bottom: const TabBar(tabs: [
                    Tab(icon: Icon(Icons.directions_bike), text: "Sepeda"),
                    Tab(icon: Icon(Icons.directions_boat), text: "Kapal")
                  ])),
              body: const TabBarView(
                 children: [
                    Center(child: Text("isi tab 1")),
                    Center(child: Text("isi tab 2"))
                ],
              ))));
  }
...
```

Hasilnya

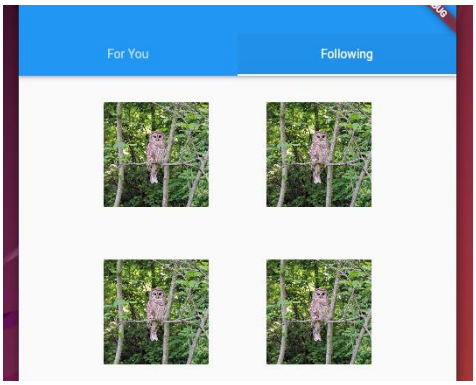


Latihan 10

Buatlah tab seperti berikut. Tab pertama adalah "For You" seperti berikut.
Catatan, gunakan FlutterLogo() untuk menampilkan icon default.



Sedangkan tab kedua "Following" sebagai berikut



Isi kedua tab menggunakan ListView.

Bottom Navigation Bar

Bottom navigator mirip seperti tab, tetapi posisinya berada di bawah. BottomNavigator berperan sebagai navigasi utama app karena posisinya yang strategis dapat diakses dengan satu tangan pada aplikasi mobile.

Dianjurkan jumlah minimal item navigasi adalah 3 dan maksimal 5. Hati-hati menggunakan kombinasi antara tab dan bottom navigation secara bersamaan karena dapat membingungkan user.

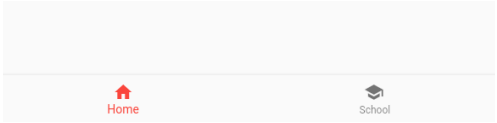
Pada Bottom Navigation, perubahan indeks karena user men-tap dikelola sendiri sehingga memerlukan statefullwidget atau state management yang lain.

Widget yang digunakan untuk BottomNavigation adalah BottomNavigationBarItem (untuk button). Berikut contoh code-nya:

```
class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  MyAppState createState() {
  return MyAppState();
  }
}

class MyAppState extends State<MyApp> {
  int idx = 0; //index yang aktif

  //isi body akan sesuai index
  static const List<Center> halaman = [
  Center(child: Text("satu")),
  Center(child: Text("dua"))
  ];

  //event saat button di-tap
  void onItemTap(int index) {
     setState(() {
       idx = index;
     });
  }

  @override
  Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
      appBar: AppBar(title: const Text("Bottom Nav")),
      body: halaman[idx],
      bottomNavigationBar: BottomNavigationBar(
          currentIndex: idx,
          selectedItemColor: Colors.red,
          onTap: onItemTap, //event saat button di tap
          items: const <BottomNavigationBarItem>[
             BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
```

```
            BottomNavigationBarItem(icon: Icon(Icons.school), label: " School"),
        ]),
  ));
  }
}
```

Hasilnya:



Source code lengkap: https://pastebin.com/raw/Rq7dM8Cz

Agar lebih modular, setiap halaman dapat disimpan di file yang terpisah. Buat dua file dart:

file **satu.dart**

```
import 'package:flutter/widgets.dart';
class LayarSatu extends StatelessWidget {
  const LayarSatu({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
  return const Center(
    child: Text('Ini layar pertama!'),
  );
  }
}
```

file **dua.dart**

```
import 'package:flutter/widgets.dart';
class LayarDua extends StatelessWidget {
  const LayarDua({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
  return const Center(
    child: Text('Ini layar kedua!'),
  );
  }
}
```

Lalu tambahkan import dua file ini di program bottom navigation:

```
import 'satu.dart';
import 'dua.dart';

…
@override
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
    appBar: AppBar(title: const Text("Bottom Nav")),
    body: case2(idx),
```

Tambahkan method untuk memilih widget yang akan mengisi body

```
case2(int idx) {
  switch (idx) {
      case 0: {return const LayarSatu();}
      case 1: {return const LayarDua();}
  }
}
```

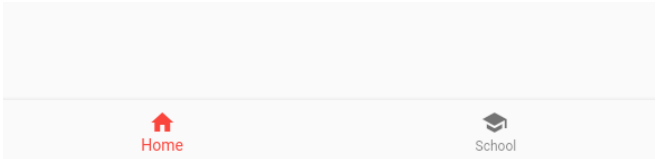
Source main.dart dapat dilihat di:
https://pastebin.com/raw/hW6kYGMS

Todo: Material Design3: Navigation Bar

Badge

Badge adalah widget untuk menampilkan informasi kecil tambahan di kanan atas icon (tidak harus icon sebenarnya, apapun yang merupakan child dari badges).

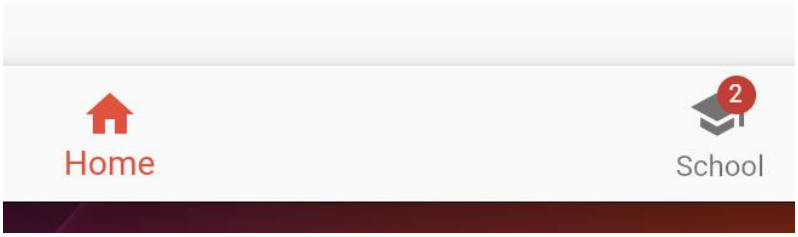
Kita akan memodifikasi code sebelumnya https://pastebin.com/raw/Rq7dM8Cz yang awalnya sebagai berikut



Tambahkan badge di icon School (sekitar baris 42).

```
. . .
BottomNavigationBarItem(
              icon: Badge(label: Text("2"), child: Icon(Icons.school)),
              label: " School"),
...
```

hasilnya



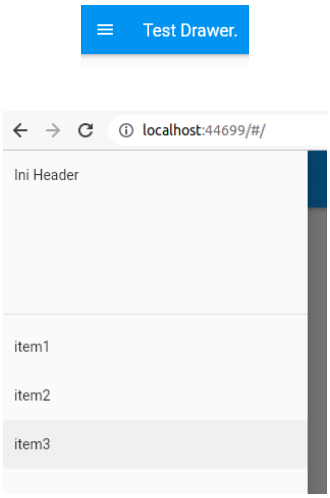
Drawer

Drawer bermanfaat jika jumlah item navigasi lebih dari 5. Drawer perlu dikombinasikan dengan DrawerHeader dan ListTile. Contoh codenya adalah sebagai berikut:

```
...
class HomePageState extends State<HomePage> {
  int idx = 0;

  //halaman body sesuai pilihan
  static const List<Center> halaman = [
  Center(child: Text("satu")),
  Center(child: Text("dua")),
  Center(child: Text("tiga")),
  ];

  void gantiItem(int index) {
  setState(() {
       idx = index;
  });
  }

  @override
  Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(title: const Text("Test Drawer.")),
      body: halaman[idx],
      drawer: Drawer(
          child: ListView(
        children: [
          const DrawerHeader(child: Text("Ini Header")),
          ListTile(
```

```
                title: const Text("item1"),
                onTap: () {
                  gantiItem(0);
                }),
            ListTile(
                title: const Text("item2"),
                onTap: () {
                  gantiItem(1);
                }),
            ListTile(
                title: const Text("item3"),
                onTap: () {
                  gantiItem(2);
                })
          ],
        )));
  }
}
```

Hasilnya adalah sebagai berikut jika menu ditekan



Code lengkap: https://pastebin.com/raw/GHDSvEfa

AppBar

App bar yang berada di atas layar berfungsi untuk branding (menampilkan nama app), judul layar, navigasi dan aksi.

Berikut contoh code-nya.

```
home: Scaffold(
      appBar: AppBar(
        leading: FlutterLogo(),
        backgroundColor: Colors.blueGrey,
        title: Text('My App'),
        actions: <Widget>[
                IconButton(
                        icon: Icon(Icons.account_circle_rounded),
                        onPressed: () {
                          // icon account di tap
                        },
                ),
                IconButton(
                        icon: Icon(Icons.settings),
                        onPressed: () {
                          // icon setting ditap
                        },
                ),
        ],
      ),
```

Hasilnya sebagai berikut (icon setting tertutup banner):

Rincian widget terkait dengan UI, khususnya yang mengikuti standar Material design dapat dilihat di: https://docs.flutter.dev/development/ui/widgets/material

Routing

Mobile app umumnya terdiri dari beberapa screen. Flutter menyebut proses perpindahan antar screen sebagai "route" yang ditangani oleh widget Navigator.

Widget Navigator mengelola history navigasi (bisa kembali ke screen awal saat button back di-tap) melalui navigation stack. Widget ini juga menangani pertukaran data antar screen. Ada tiga class yang terlibat: Navigator, Overlay dan Route.

Ada dua jenis Navigator. Pertama Navigator 1.0 yang lebih tepat untuk mobile app dan Navigator 2.0 yang mendukung deep link. Deep link digunakan pada aplikasi web saat user masuk ke aplikasi melalui link URL (misal link langsung ke suatu produk). Walaupun tidak melalui halaman pertama, kita tetap ingin fungsi-fungsi navigasi tetap berjalan. Navigator 2.0 lebih rumit dibandingkan Navigator 1.0

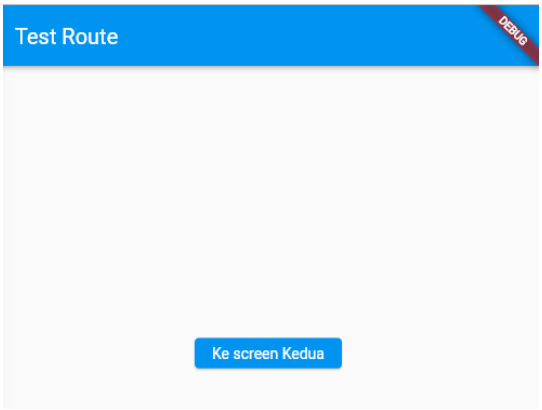
Navigator 1.0

Berikut adalah contoh penggunaan Navigator 1.0

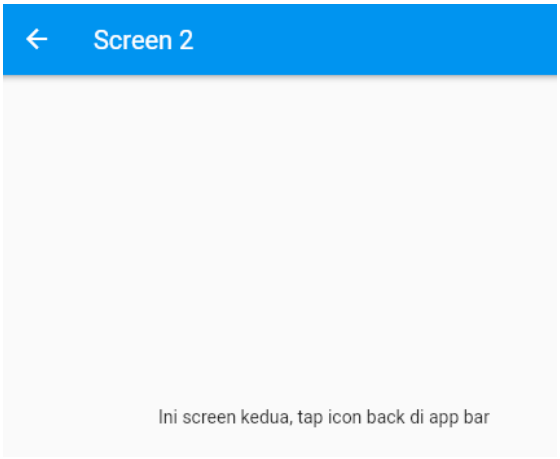
```
  import 'package:flutter/material.dart';

//screen kedua
class LayarKedua extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
  return Scaffold(
       appBar: AppBar(
            title: const Text(' Ini Screen 2'),
       ),
       body: Center(child: Text("screen kedua, tap icon back di app bar")),
  );
  }
}

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
```

```
  Widget build(BuildContext context) {
     return MaterialApp(title: 'Test Route', home: MyHome());
  }
}

//perlu  class MyHome ini karena Navigator perlu akses ke parent Material App
class MyHome extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        title: const Text('Test Route'),
    ),
    body: Center(
      child: ElevatedButton(
          child: Text('Ke screen Kedua'),
          onPressed: () {
               Navigator.of(context).push(MaterialPageRoute(builder: (context) {
                    return LayarKedua();
               }));
          }),
    ),
  );
  }
}
```

Hasilnya



Saat button di-tap akan terjadi perpindahan ke screen ke-2. Pada screen ke-2, jika icon back ditekan "←" maka akan kembali ke screen pertama.



Code lengkap: https://pastebin.com/raw/UcgnpMPC

Catatan: untuk meningkatkan modularitas, class LayarKedua dapat dipisah ke dalam file lain, misalnya `layarkedua.dart` kemudian digunakan melalui impor di layar pertama.

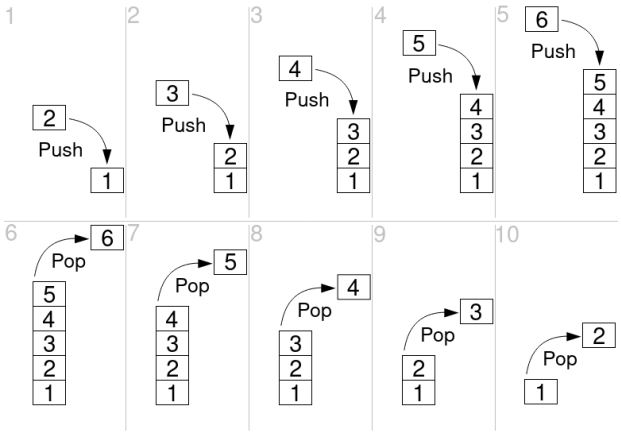
```
import 'layarkedua.dart';
```

Proses kembali ke layar sebelumnya dapat dilakukan manual dengan method pop(). Coba modifikasi code sebelumnya, dengan menambahkan button yang jika di-tap akan kembali ke layar sebelumnya.

```
. . .
      body: Center(
          child: ElevatedButton(
          child: const Text("Ini screen kedua, tap icon back di app bar"),
          onPressed: () {
              Navigator.of(context).pop();
          },
      )));
. . .
```

Code lengkap: https://pastebin.com/raw/MQvETG44

Navigator menggunakan push dan pop karena mengelola screen dalam bentuk stack. Gambar berikut memperlihatkan mekanisme push dan pop di dalam stack (LIFO: Last in First Out). Itu sebabnya saat fungsi pop() dipanggil, screen yang aktif "dibuang" dan kembali ke screen sebelumnya.



Passing Parameter

Kita dapat mengirimkan data ke screen yang akan ditampilkan dengan menggunakan constructor screen tersebut.

Berikut modifikasi program sebelumnya untuk screen kedua yang dipanggil:

```
class LayarKedua extends StatelessWidget {
  const LayarKedua({Key? key, required this.pesan}) : super(key: key);
  final String pesan;
. . .
```

Selanjutnya variabel ini dapat digunakan

```
body: Center(
```

```
          child: ElevatedButton(
        child: Text("Ini screen kedua, ada pesan: $pesan"),
        onPressed: () {
          Navigator.of(context).pop();
        },
      )));
```

Sedangkan di layar pertama, passing parameter dalam constructor:

```
onPressed: () {
     Navigator.of(context).push(MaterialPageRoute(builder: (context) {
              return const LayarKedua(pesan: "haloo ini pesan dari screen 1");
            }));
      }),
```

Code: https://pastebin.com/raw/ri86RebY

Menerima Result dari Screen yang Dipanggil

Kalau sebelumnya kita mengirimkan data, sekarang setelah memanggil suatu screen kita mengharapkan data dari screen tersebut.

Data dapat dikirim melalui method pop(). Pada screen ke-2, tambahkan code berikut:

```
onPressed: () {
          Navigator.of(context).pop("ini pesan dari screen 2");
        },
```

Data ini ditangkap melalui method push sebagai return.

```
onPressed: () async {
            hasil = await Navigator.of(context)
                .push(MaterialPageRoute(builder: (context) {
              return const LayarKedua(pesan: "haloo ini pesan dari screen 1");
            }));
          }),
```

Future, Async dan Wait dibahas di bagian awal modul ini (bagian Dart). Return dari push() bertipe future sehingga diperlukan await untuk menunggu screen ini selesai.

Code lengkap: https://pastebin.com/raw/HJBdjr6s

Named Route

Route dapat diberi nama sehingga code lebih mudah untuk dibaca. Berikut code untuk menambahkan route. Modifikasi code pertama https://pastebin.com/raw/MQvETG44 dengan tambahan sebagai berikut:

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Test Route',
       //atribut home dihapus!
    routes: {
        '/': (context) => const MyHome(),
        '/layar2': (context) => const LayarKedua()
      },
  );
  }
}
```

dan saat pemanggilanya dapat digunakan method pushNamed() dengan nama route-nya.

```
onPressed: () {
             Navigator.of(context).pushNamed("/layar2");
          }
```

Walaupun code jadi lebih mudah dibaca, tapi kelemahannya jika terjadi salah penulisan pada nama route, maka app akan error pada saat runtime, bukan compile time.

Code lengkap: https://pastebin.com/raw/S8tM92dU

Notifikasi

Notifikasi adalah informasi singkat untuk user mengenai suatu event. Notifikasi dapat digunakan sebagai reminder, informasi dari user lain atau meminta user membuka app untuk melakukan suatu aksi. Flutter memiliki dua jenis notifikasi. Local notification bersumber dari aplikasi, sedangkan push notifications bersumber dari server yang dikirimkan walaupun aplikasi dalam kondisi tidak aktif.

Local Notification

https://pub.dev/packages/flutter_local_notifications

flutter_local_notifications: ^14.0.0

Push Notification

Push Notification (informasi yang di-push ke user) merupakan salah satu kelebihan dari aplikasi mobile. Push notification terjadi saat aplikasi sedang tidak digunakan (berada di luar UI app).

Salah satu implementasi notifikasi, khususnya di Android adalah menggunakan Firebase Cloud Messaging, yang memungkikan kita mengirimkan notifikasi walaupun aplikasi dalam kondisi tidak aktif.

[todo]

Referensi

Alessandria, Simone. *Flutter Projects: A Practical, Project-based Guide to Building Real-world Cross-platform Mobile Applications and Games*. Packt Publishing, 2020.

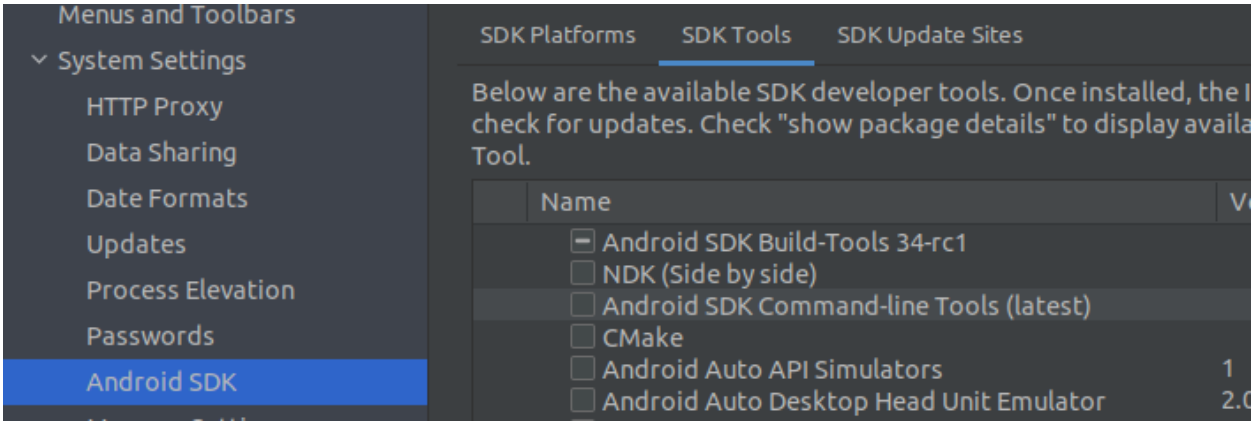
Bailey, Thomas. *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter 2.5 and Dart*, 2nd Edition

https://api.flutter.dev

Tambahan

Flutter dan Android Studio

Setting → SDK Manager, command line tools



Todo lain-lain:

SafeArea
TextSpan
Positioned
Flexible
isExpanded (DropDown)
SingleChildScrollView
Carousel
with (keyword seperti interface)
passing data antar widget
Bottomsheet
Expansion Panel

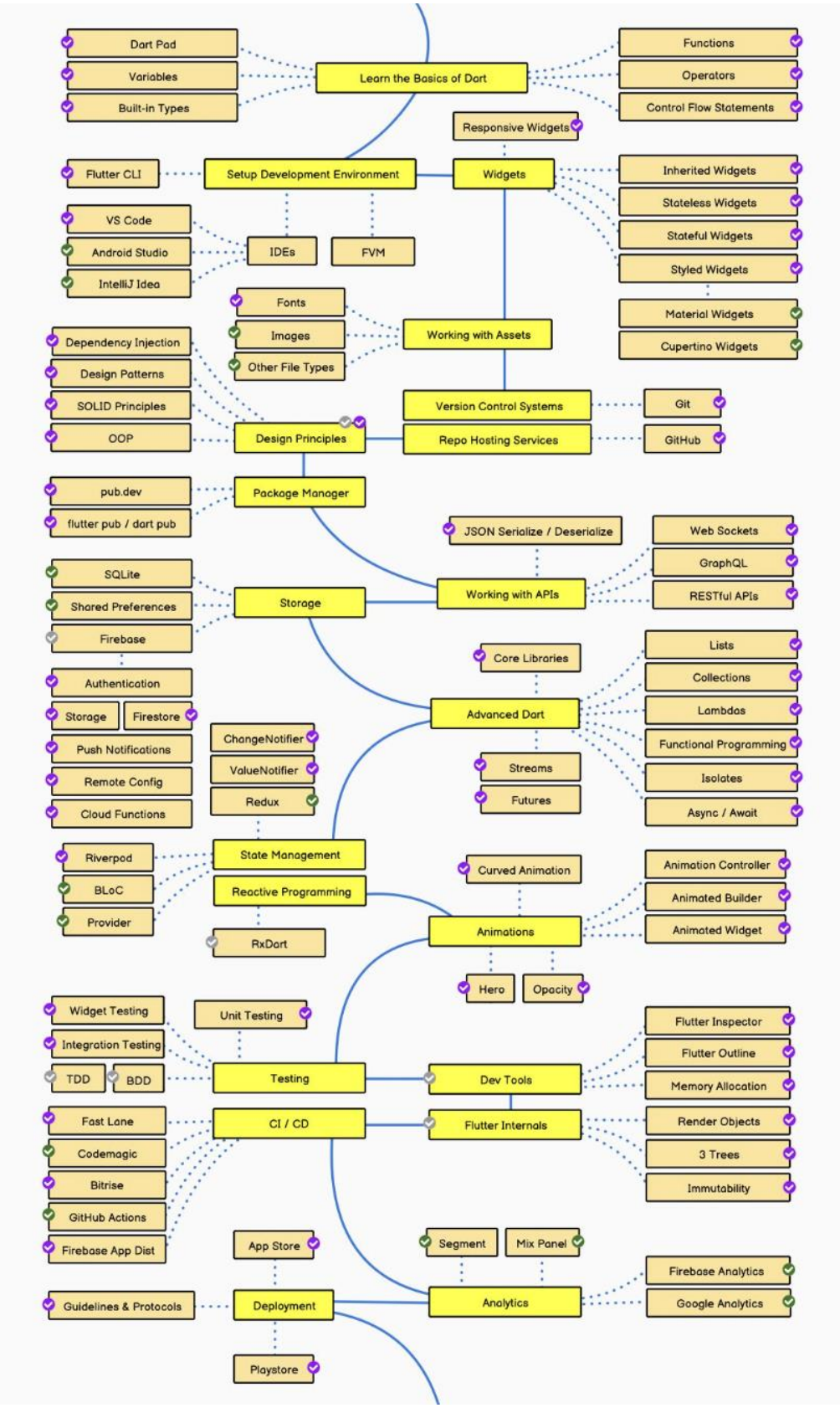
CircleAvatar, Divider

https://api.flutter.dev/flutter/material/ListTile-class.html?gclid=CjwKCAjwscGjBhAXEiwAswQqNEK78WoWKLTYXAQVpG1vi7a7CFOFFCuHP33Kk8QCFpv4w0SI1OrMABoCzUQQAvD_BwE&gclsrc=aw.ds

Icon:
https://fonts.google.com/icons

App hello world paling sederhana

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override

  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```


Dart Pad
Variables
Built-in Types
Learn the Basics of Dart
Functions
Operators
Control Flow Statements
Responsive Widgets
Flutter CLI
Setup Development Environment
Widgets
Inherited Widgets
Stateless Widgets
Stateful Widgets
Styled Widgets
VS Code
Android Studio
IntelliJ Idea
IDEs
FVM
Material Widgets
Cupertino Widgets
Fonts
Images
Other File Types
Working with Assets
Dependency Injection
Design Patterns
SOLID Principles
OOP
Design Principles
Version Control Systems
Repo Hosting Services
Git
GitHub
pub.dev
flutter pub / dart pub
Package Manager
JSON Serialize / Deserialize
Web Sockets
GraphQL
RESTful APIs
Working with APIs
SQLite
Shared Preferences
Firebase
Storage
Authentication
Storage
Firestore
Push Notifications
Remote Config
Cloud Functions
Core Libraries
Lists
Collections
Lambdas
Functional Programming
Isolates
Async / Await
Advanced Dart
Streams
Futures
ChangeNotifier
ValueNotifier
Redux
Riverpod
BLoC
Provider
State Management
Reactive Programming
RxDart
Curved Animation
Animation Controller
Animated Builder
Animated Widget
Animations
Hero
Opacity
Widget Testing
Integration Testing
TDD
BDD
Unit Testing
Testing
Dev Tools
Flutter Inspector
Flutter Outline
Memory Allocation
Fast Lane
Codemagic
Bitrise
GitHub Actions
Firebase App Dist
CI / CD
Flutter Internals
Render Objects
3 Trees
Immutability
App Store
Segment
Mix Panel
Firebase Analytics
Google Analytics
Guidelines & Protocols
Deployment
Analytics
Playstore