

<b>Universitas</b>	: Universitas Logistik dan Bisnis Internasional (ULBI)
<b>Program Studi</b>	: D4 Teknik Informatika
<b>Semester</b>	: 5 (Lima)
<b>Mata Kuliah</b>	: Pemrograman IV (Pemrograman Mobile)
<b>Topik</b>	: Praktikum 5 Membuat Layout

Buat project latihan baru dengan nama **latihan** kemudian buat kerangka utama pada **main.dart** seperti di bawah

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        useMaterial3: false,
      ),
      home: const FirstScreen(),
    );
  }
}
```

## A. Image

Dalam proses pengembangan aplikasi, gambar atau ilustrasi sangat penting untuk meningkatkan daya tarik tampilan. Pada pembelajaran ini, kita akan membahas cara menampilkan gambar baik yang berasal dari internet maupun yang sudah ada di dalam proyek aplikasi kita.

### A.1. Image.network

Untuk menampilkan gambar yang bersumber dari internet, kita akan menggunakan method `Image.network`. Cara penulisan method ini sebagai berikut:

```
Image.network(url)
```

Method ini cukup menambahkan URL gambar dari internet dan kita pun dapat menambahkan width dan height juga. Di bawah ini adalah contoh penggunaan **Image.network**:

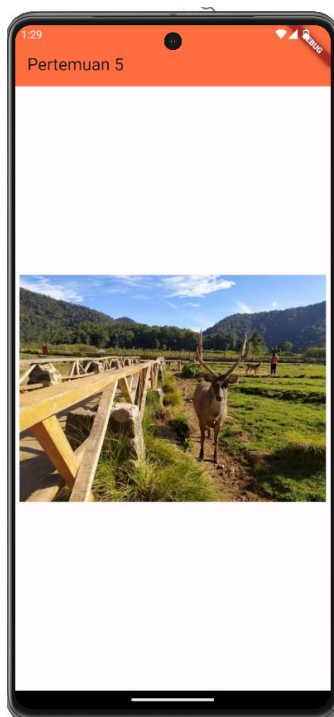
```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pertemuan 5'),
        backgroundColor: Colors.deepOrangeAccent,
      ),
      body: Center(
        child: Image.network(
          'https://www.radarbandung.id/wp-content/uploads/2022/09/ranca-
upas.jpg',
          width: 400,
          height: 400,
        ),
      ),
    );
  }
}

```

Pada kode di atas kita panggil method **Image.network** dengan url <https://www.radarbandung.id/wp-content/uploads/2022/09/ranca-upas.jpg> lalu beri width dan height masing-masing 400. Sehingga hasilnya seperti berikut:

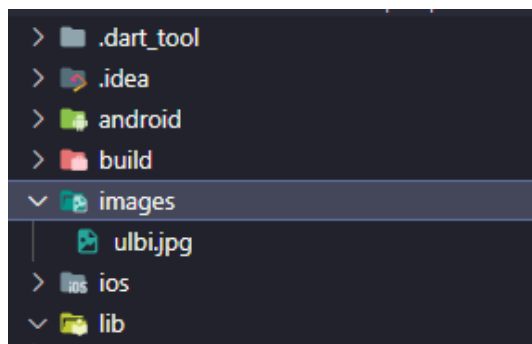


## A.2. Image.asset

Selain mengambil gambar dari internet, kita juga memiliki opsi untuk menampilkan gambar yang berasal dari aset di dalam proyek. Dalam konteks ini, aset merujuk pada gambar-gambar yang sudah dimasukkan ke dalam proyek dan telah didaftarkan di berkas `pubspec.yaml`.

Langkah awal yang harus kita lakukan adalah memasukkan gambar-gambar yang ingin digunakan ke dalam folder proyek. Flutter mendukung berbagai format gambar seperti JPEG, PNG, GIF, Animated GIF, WebP, Animated WebP, BMP, dan WBMP. Untuk format gambar yang tidak termasuk dalam daftar tersebut, Flutter akan menggunakan API yang tersedia pada platform masing-masing. Dengan demikian, jika platform native mendukung format gambar yang digunakan, Flutter akan mampu merender gambar tersebut tanpa masalah.

Pada contoh berikut kita menambahkan folder `images/` pada folder project.



Masukkan berkas gambar yang ingin kalian gunakan ke dalam folder image. Sebagai contoh kita menggunakan gambar bernama [ulbi.jpg](#).

Setelah menambahkan gambar pada project, saatnya kita mendaftarkan gambar tersebut pada **pubspec.yaml**. Di dalam berkas `pubspec.yaml`, kita bisa mendaftarkan aset gambar pada bagian flutter seperti di bawah ini:

```
flutter:  
  uses-material-design: true  
  
  assets:  
    - images/ulbi.jpg
```

Perhatikan pula indentasi kodenya. **assets:** berada sejajar dengan **uses-material-design:** yaitu berjarak 2 spasi dari ujung dan berada di dalam **flutter:** sedangkan **- images/ulbi.jpg** berada di dalam **assets:** dan berjarak 4 spasi dari ujung.

Pada contoh di atas kita telah menambahkan asset yang berisi lokasi gambar atau aset yang ingin kita gunakan. Karena kita menambahkan gambar **ulbi.jpg** pada folder images, maka lokasi gambar tersebut adalah **images/ulbi.jpg**.

Apabila ada banyak gambar yang kita masukkan ke dalam lokasi folder, dibandingkan menuliskan lokasi gambar satu per satu, kita bisa langsung menuliskan folder **images/** seperti berikut:

```
flutter:

uses-material-design: true

assets:
  - images/
```

Setelah menambahkan assets, kita harus me-refresh **pubspec.yaml** dengan cara save file pubspec.yaml bila menggunakan Visual Studio Code.

Setelah kita menambahkan asset ke dalam pubspec.yaml kita perlu melakukan full restart agar asset yang baru dapat digunakan dalam aplikasi.

Kita telah mendaftarkan suatu asset. Sekarang kita akan panggil asset tersebut pada kode kita dengan method **Image.asset**. Cara penulisannya seperti berikut:

```
Image.asset(lokasi_asset)
```

Contoh dalam kodenya akan seperti berikut dengan membuat class baru lagi:

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pertemuan 5'),
        backgroundColor: Colors.deepOrangeAccent,
      ),
      body: Center(
        child: Image.asset('images/ulbi.jpg', width: 400, height: 400),
      ),
    );
  }
}
```

Jika kita jalankan aplikasi Flutter, maka gambar akan tampil seperti berikut:



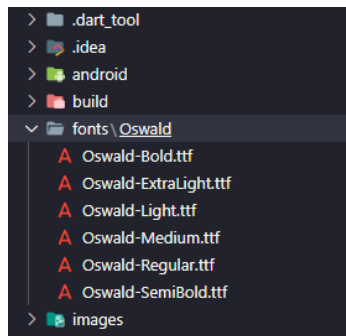
## **B. Font**

Ketika kita mengembangkan aplikasi, seringkali desainer User Interface menginginkan penggunaan font yang berbeda dari font default yang disediakan. Sebagai pengembang aplikasi, tugas kita adalah memasukkan font yang sesuai dengan desain UI ke dalam aplikasi.

Dalam pelajaran ini, kita akan membahas cara menambahkan font khusus ke dalam proyek Flutter. Sebelum kita memulai, langkah pertama adalah mendapatkan font tersebut, yang bisa berarti mengunduhnya dari internet atau menggunakan font yang sudah ada. Sebagai contoh, kita akan mendownload font dari [Google Fonts](https://fonts.google.com/specimen/Oswald), yaitu font bernama [Oswald](https://fonts.google.com/specimen/Oswald).

### **B.1. Menambahkan Font ke Project**

Setelah mengunduh font, langkah selanjutnya kita akan memasukkan file-file font tersebut ke folder project. Pada contoh ini kita akan membuat folder fonts pada project kita, dan masukkan file-file font yang telah diunduh, seperti berikut:



## B.2. Image.network

Sama halnya dengan gambar, kita perlu mendaftarkan font pada berkas pubspec.yaml sebagai asset seperti berikut:

```
flutter:  
  
  uses-material-design: true  
  
  assets:  
    - images/  
  
  fonts:  
    - family: Oswald  
      fonts:  
        - asset: fonts/Oswald/Oswald-Regular.ttf
```

Sama halnya dengan gambar, font ada dalam bagian flutter. Untuk mendaftarkan font, kita membuat bagian fonts yang ada dalam blok flutter.

Untuk mendaftarkan font **Oswald** kita tuliskan **Oswald** pada bagian **family** yang nantinya akan menjadi nama font yang kita panggil pada kode dart. Lalu dalam **family** kita masukkan fonts yang di dalamnya terdapat asset yang nanti akan mengarah pada **file font.ttf**. Contoh di atas kita menambahkan **asset fonts/oswald/Oswald-Regular.ttf**.

## B.3. Menggunakan Font pada Kode

Setelah kita mendaftarkan font pada **pubspec.yaml** kita akan gunakan font tersebut pada kode kita. Seperti contoh di bawah ini kita akan menggunakan font pada widget **Text**:

```
class MyCustomFont extends StatelessWidget {  
  const MyCustomFont({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Pertemuan 5'),  
        backgroundColor: Colors.deepOrangeAccent,  
      ),  
      body: const Center(  
        child: Text(  

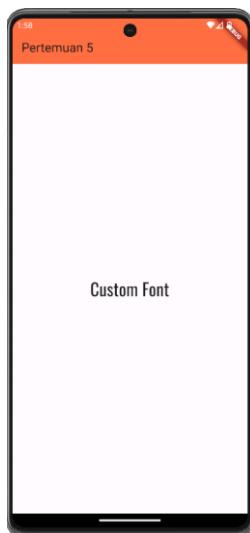
```

```

        'Custom Font',
        style: TextStyle(
          fontFamily: 'Oswald',
          fontSize: 30,
        ),
      ),
    ),
  );
}
}

```

Pada kode di atas kita menambahkan fontFamily pada **TextStyle**. Kita cukup panggil nama font family yang telah kita daftarkan pada **pubspec.yaml**. Hasilnya akan seperti berikut:



Tulisan "**Custom Font**" akan berubah menjadi font **Oswald** sesuai dengan yang telah kita daftarkan. Jangan lupa! Setelah kita menambahkan package atau pun asset ke dalam **pubspec.yaml** kita perlu melakukan **full restart** agar asset yang baru dapat digunakan dalam aplikasi.

#### B.4. Mengubah Font Default

Selain kita dapat mengubah font family pada satu per satu widget Text, kita dapat membuat font yang kita daftarkan menjadi default. Caranya dengan menambahkan parameter **fontFamily** pada kelas **ThemeData** yang ada pada parameter theme di **MaterialApp** seperti berikut:

```

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',

```

```

    theme: ThemeData(
      primarySwatch: Colors.blue,
      fontFamily: 'Oswald',
    ),
    home: const MyCustomFont(),
  );
}
}

```

### C. ListView

Cara penggunaan ListView ini mirip dengan Column atau Row di mana Kalian memasukkan widget yang ingin disusun sebagai children dari ListView.

```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ListView(
        children: <Widget>[
          Container(
            height: 250,
            decoration: BoxDecoration(
              color: Colors.grey,
              border: Border.all(color: Colors.black),
            ),
            child: const Center(
              child: Text(
                '1',
                style: TextStyle(fontSize: 50),
              ),
            ),
          ),
          Container(
            height: 250,
            decoration: BoxDecoration(
              color: Colors.grey,
              border: Border.all(color: Colors.black),
            ),
            child: const Center(
              child: Text(
                '2',
                style: TextStyle(fontSize: 50),
              ),
            ),
          ),
          Container(

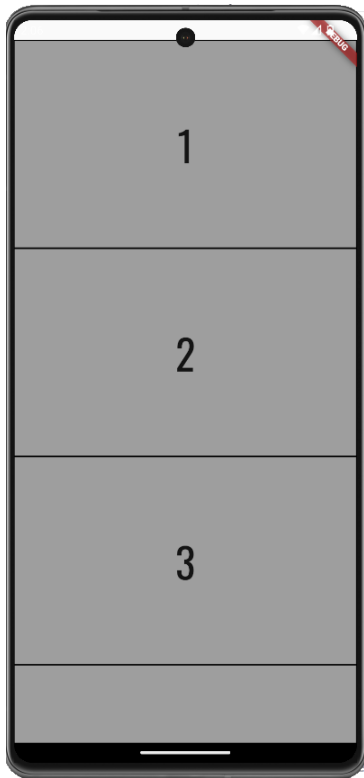
```

```

        height: 250,
        decoration: BoxDecoration(
          color: Colors.grey,
          border: Border.all(color: Colors.black),
        ),
        child: const Center(
          child: Text(
            '3',
            style: TextStyle(fontSize: 50),
          ),
        ),
      ),
    ),
    Container(
      height: 250,
      decoration: BoxDecoration(
        color: Colors.grey,
        border: Border.all(color: Colors.black),
      ),
      child: const Center(
        child: Text(
          '4',
          style: TextStyle(fontSize: 50),
        ),
      ),
    ),
  ],
),
);
}
}

```

Ketika dijalankan, aplikasi akan menjadi seperti berikut dan coba lakukan scroll ke atas dan ke bawah:



### C.1. Menampilkan Item Secara Dinamis

Selain memasukkan widget satu per satu ke dalam **children** dari **ListView**, Kalian juga dapat menampilkan list secara dinamis. Ini sangat berguna ketika kalian memiliki banyak item dengan jumlah yang tidak menentu.

Misalnya kita ingin menampilkan daftar angka dari 1 sampai 10.

```
final List<int> numberList = const <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

Caranya, masukkan variabel atau list kalian sebagai children lalu panggil fungsi `map()`. Fungsi `map` ini berguna untuk memetakan atau mengubah setiap item di dalam list menjadi objek yang kita inginkan. Fungsi `map` ini membutuhkan satu buah parameter berupa fungsi.

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

  final List<int> numberList = const <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ListView(
        children: numberList.map((number) {}),
      ),
    );
  }
}
```

Karena parameter children ini membutuhkan nilai berupa list widget, maka kita perlu mengembalikan setiap item dari **numberList** menjadi **widget** yang akan ditampilkan. Ubah fungsi kalian menjadi seperti berikut:

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

  final List<int> numberList = const <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ListView(
        children: numberList.map((number) {
          return Container(
            height: 250,
            decoration: BoxDecoration(
              color: Colors.grey,
              border: Border.all(color: Colors.black),
            ),
            child: Center(
              child: Text(
                '$number', // Ditampilkan sesuai item
                style: const TextStyle(fontSize: 50),
              ),
            ),
          );
        }).toList(),
      ),
    );
  }
}
```

Perhatikan di akhir kita perlu mengembalikan fungsi map menjadi objek List lagi dengan fungsi **.toList()**. Lakukan **hot reload** pada aplikasi kalian untuk melihat hasil perubahan dan coba scroll ke bawah.

## C.2. Menggunakan ListView.builder

Selain mengisi parameter children dari ListView seperti sebelumnya, kita juga bisa memanfaatkan method ListView.builder. ListView.builder lebih cocok digunakan pada ListView dengan jumlah item yang cukup besar. Ini karena Flutter hanya akan merender tampilan item yang terlihat di layar dan tidak me-render seluruh item ListView di awal.

ListView.builder memerlukan dua parameter yaitu **itemBuilder** dan **itemCount**. Parameter itemBuilder merupakan fungsi yang mengembalikan widget untuk ditampilkan. Sedangkan itemCount kita isi dengan jumlah seluruh item yang ingin ditampilkan.

Berikut ini adalah contoh kode penggunaan ListView.builder:

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

  final List<int> numberList = const <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ListView.builder(
        itemCount: numberList.length,
        itemBuilder: (BuildContext context, int index) {
          return Container(
            height: 250,
            decoration: BoxDecoration(
              color: Colors.grey,
              border: Border.all(color: Colors.black),
            ),
            child: Center(
              child: Text(
                '${numberList[index]}',
                style: const TextStyle(fontSize: 50),
              ),
            ),
          );
        },
      ),
    );
  }
}
```

### C.3. Menggunakan ListView.separated

Cara lain untuk membuat ListView adalah dengan metode ListView.separated. ListView jenis ini akan menampilkan daftar item yang dipisahkan dengan separator. Penggunaan ListView.separated mirip dengan builder, yang membedakan adalah terdapat satu parameter tambahan wajib yaitu separatorBuilder yang mengembalikan Widget yang akan berperan sebagai separator.

Berikut ini adalah contoh kode dari ListView.separated:

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

  final List<int> numberList = const <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

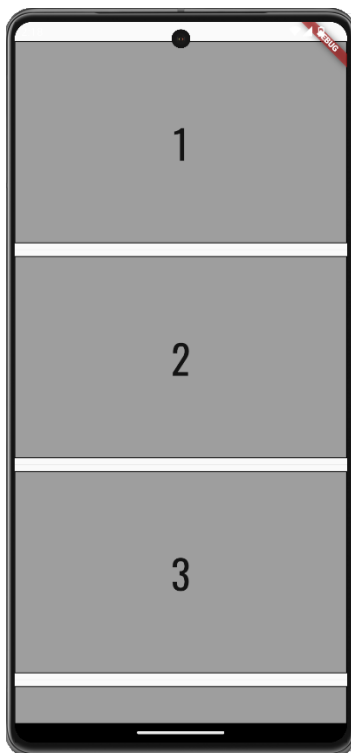
  @override
```

```

Widget build(BuildContext context) {
  return Scaffold(
    body: ListView.separated(
      itemCount: numberList.length,
      itemBuilder: (BuildContext context, int index) {
        return Container(
          height: 250,
          decoration: BoxDecoration(
            color: Colors.grey,
            border: Border.all(color: Colors.black),
          ),
          child: Center(
            child: Text(
              '${numberList[index]}',
              style: const TextStyle(fontSize: 50),
            ),
          ),
        );
      },
      separatorBuilder: (BuildContext context, int index) {
        return const Divider();
      },
    ),
  );
}

```

Jika kode di atas dijalankan, maka tampilan aplikasi adalah seperti ini:



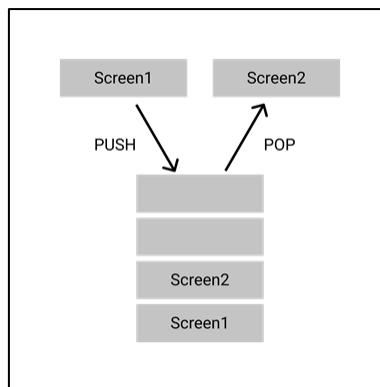
## D. Navigation

Kita telah bisa membuat satu tampilan screen (layar/page) pada pembelajaran sebelumnya. Namun, pada saat membangun sebuah aplikasi kita akan membuat banyak sekali screen dan kita akan berpindah dari satu screen ke screen lainnya.

Pada Flutter kita akan menggunakan sebuah class bernama **Navigator**. Dengan Navigator ini kita akan berpindah dari satu screen ke screen lainnya. Berikut ini contohnya:

### [BUKA DISINI](#)

Perlu kita ketahui bahwa konsep navigasi pada Flutter mirip sekali dengan pemrograman Android, yakni bahwa ketika berpindah screen/activity akan menjadi tumpukan (stack). Jadi ketika berpindah dari satu screen ke screen lain (push), maka screen pertama akan ditumpuk oleh screen kedua. Kemudian apabila kembali dari screen kedua ke pertama, maka screen kedua akan dihapus (pop).



Kita akan membuat kode seperti contoh di atas. Kita membutuhkan halaman kedua yang kodenya seperti berikut:

```
class SecondScreen extends StatelessWidget {
  const SecondScreen({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Screen'),
      ),
      body: Center(
        child: OutlinedButton(
          child: const Text('Kembali'),
          onPressed: () {},
        ),
      ),
    );
  }
}
```

Lalu, kode untuk halaman pertama akan seperti berikut:

```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Pindah Screen'),
          onPressed: () {},
        ),
      ),
    );
  }
}

```

### D.1. Navigator.push

Untuk berpindah ke screen kedua kita akan menggunakan sebuah method **Navigator.push**. Navigator.push memiliki dua parameter. Pertama ialah context dan yang kedua Route. Parameter context ini merupakan variabel BuildContext yang ada pada method build. Parameter route berguna untuk menentukan tujuan ke mana kita akan berpindah screen. Route tersebut kita isikan dengan MaterialPageRoute yang di dalamnya terdapat builder yang nantinya akan diisi dengan tujuan screen-nya. Maka untuk melakukan perpindahan screen kita akan membuat event **onPressed** pada tombol ElevatedButton yang ada pada screen pertama:

```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Pindah Screen'),
          onPressed: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) {
              return const SecondScreen();
            }));
          },
        ),
      ),
    );
  }
}

```

```
}
```

## D.2. Navigator.pop

Setelah dapat berpindah ke screen lain maka kita akan belajar menggunakan **Navigator.pop** untuk kembali ke screen sebelumnya.

Pada **Navigator.pop** kita hanya cukup menambahkan parameter context yang merupakan variabel dari method build.

Untuk kembali dari screen kedua kita dapat menambahkan event **onPressed** pada OutlinedButton yang ada pada screen kedua dan kita masukkan **Navigator.pop** pada event, seperti berikut:

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

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

Jika kode di atas dijalankan, maka tampilan aplikasi adalah [seperti ini](#)

## D.3. Mengirimkan Data Antar Halaman

Seringkali beberapa halaman pada aplikasi perlu saling berinteraksi dengan berbagi dan saling mengirimkan data. Pada Flutter kita memanfaatkan constructor dari sebuah class untuk mengirimkan data antar halaman.

Sebagai contoh kita memiliki pesan yang akan dikirimkan dari First Screen menuju Second Screen.

```
final String message = 'Hello from First Screen!';
```

Untuk mengirimkan variabel message tersebut ke Second Screen, maka kita akan mengirimkannya sebagai parameter dari constructor kelas SecondScreen seperti berikut:

```
class FirstScreen extends StatelessWidget {  
  const FirstScreen({super.key});  
  
  final String message = 'Hello from First Screen!';  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('First Screen'),  
      ),  
      body: Center(  
        child: ElevatedButton(  
          child: const Text('Pindah Screen'),  
          onPressed: () {  
            Navigator.push(context,  
              MaterialPageRoute(builder: (context) => SecondScreen(message)));  
          },  
        ),  
      ),  
    );  
  }  
}
```

Kode di atas seharusnya masih error, agar Second Screen bisa menerima data tersebut, maka kita perlu mengubah default constructor-nya dan menambahkan variabel untuk menampung datanya.

```
class SecondScreen extends StatelessWidget {  
  final String message;  
  
  const SecondScreen(this.message, {super.key});  
}
```

Kemudian kita dapat menampilkan data yang diterima melalui variabel yang kita buat.

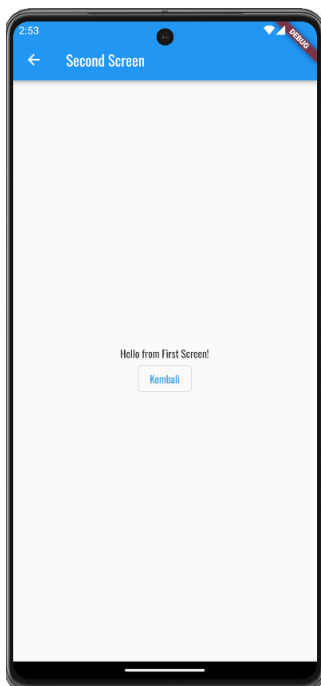
```
class SecondScreen extends StatelessWidget {  
  final String message;  
  
  const SecondScreen(this.message, {super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Second Screen'),  
      ),  
    );  
  }  
}
```

```

    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(message),
          OutlinedButton(
            child: const Text('Kembali'),
            onPressed: () {
              Navigator.pop(context);
            },
          ),
        ],
      ),
    ),
  ),
);
}
}

```

Sehingga tampilan Second Screen akan menampilkan pesan dari First Screen seperti berikut:

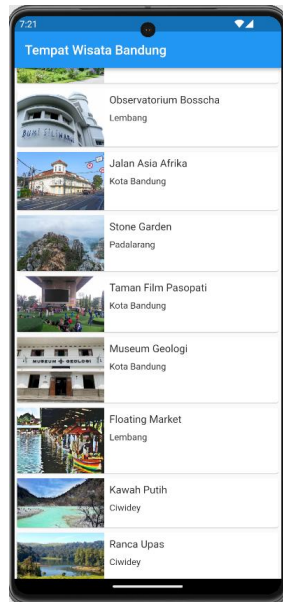


Untuk memahami materi yang sudah dijelaskan di atas secara lanjut ,silahkan pelajari dokumentasi berikut:

- [Image Class](#)
- [Google Fonts](#)
- [ListView Class](#)
- [Navigation Cookbook](#)

# PERCOBAAN PRAKTIKUM

Setelah mempelajari beberapa materi tambahan, sekarang saatnya kita melanjutkan praktikum pada pertemuan ke-4 yang bertema aplikasi wisata. Pada praktikum ini kita akan membuat aplikasi dengan tampilan seperti berikut:



1. Lanjut project Flutter yang sudah dibuat kemarin di Pertemuan 4 (**p4\_1\_npm**).
2. Untuk memudahkan dalam membaca sekaligus merapikan kode, mari kita pindahkan widget atau kelas **DetailScreen** ke sebuah *file* dart baru. Kalian dapat membuat *file* baru dengan cara **klik kanan pada folder lib → New → Dart File**. Berikan nama **detail\_screen.dart**.
3. Kalian akan mendapati beberapa eror akibat adanya *library* atau *package* yang belum terpasang. Pada *file detail\_screen.dart* tambahkan kode **import** berikut di baris paling atas untuk menggunakan *package* material design di dalam file.

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

4. Selanjutnya karena kita akan menggunakan *file widget* **DetailScreen** di file **main.dart**, maka kita juga perlu melakukan **import** berkas **detail\_screen.dart** ke dalam **main.dart**.

```
import 'detail_screen.dart';
```

5. Kemudian kita akan menambahkan sebuah gambar ke tampilan paling atas halaman. Gambar ini akan kita ambil dari asset. Untuk itu, kita perlu menambahkan berkas yang ingin ditampilkan ke dalam *project* dan menambahkannya pada *file pubspec.yaml*. Aset gambar dapat kalian unduh pada [tautan berikut](#). Kemudian buat folder **images**, lalu semua gambar yang ada di link di atas masukkan ke dalam folder **images**.

```
flutter:

uses-material-design: true

assets:
  - images/
```

Tambahkan widget Image di child paling atas dari Column

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Image.asset('images/ranca-upas.jpg'),
            Container(
              margin: const EdgeInsets.only(top: 16.0),
              child: const Text(
                'Ranca Upas',
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 30.0,
                  fontWeight: FontWeight.bold,
                ), // TextStyle
              ), // Text
            ), // Container
          ],
        ),
      ),
    );
  }
}
```

Jalankan aplikasi untuk melihat perubahan.



- Selanjutnya kita akan menampilkan beberapa gambar lagi di bagian bawah. Kali ini kita akan mengambil gambar melalui url. Mari kita mulai dengan satu gambar terlebih dahulu. Kita tambah gambar setelah Container terakhir yang berisi deskripsi

```
Container(
  padding: const EdgeInsets.all(16.0),
  child: const Text(
    'Ranca Upas Ciwidey adalah kawasan bumi perkemahan di bawah pengelolaan perhutani. Tempat ini berada di k',
    textAlign: TextAlign.justify,
    style: TextStyle(
      fontSize: 16.0,
      color: Color.fromARGB(255, 84, 1, 133),
    ), // TextStyle
  ), // Text
), // Container
Image.network('https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'),
```

```
Image.network('https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'),
```

Apabila gambar yang kita tampilkan terlalu besar sementara layar pada perangkat terlalu kecil, maka akan terlihat tampilan garis hitam-kuning yang menunjukkan terjadi *overflow*. Kondisi *overflow* ini terjadi ketika konten yang kita tampilkan melebihi luas layar yang ada.



7. Sebagai solusi, tentunya kita bisa mengubah ukuran dari gambar, namun tentunya tidak praktis jika kita harus mengubah ukuran setiap gambar yang ditampilkan. Tentu ada banyak sekali ukuran layar yang tersedia, bukan? Solusi lainnya yaitu dengan menerapkan scrolling. Salah satu *widget scrolling* yang bisa kita manfaatkan adalah **SingleChildScrollView**. Widget ini membutuhkan satu child yang nantinya bisa di-*scroll* pada layar. Pindahkan widget *Column* ke dalam **SingleChildScrollView** supaya nantinya bisa di-*scroll* dengan cara **klik widget Column → Klik kanan → Refactor → Wrap with widget → ketik SingleChildScrollView**.

```
class DetailScreen extends StatelessWidget {  
  const DetailScreen({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: SafeArea(  
        child: SingleChildScrollView(  
          child: Column(  
            crossAxisAlignment: CrossAxisAlignment.stretch,  
            children: <Widget>[  
              Image.asset('images/ranca-upas.jpg'),  
              Container(  

```

Jalankan *hot reload*. Seharusnya masalah *overflow* sudah teratasi dengan adanya *scrolling*.

8. Selanjutnya kita akan menambahkan beberapa gambar lagi yang disusun secara horizontal. Kalian mungkin mengira untuk menggunakan widget *Row* supaya gambar bisa tersusun secara horizontal. Namun, perlu diingat bahwa kita juga memerlukan fitur *scrolling* agar tidak terjadi *overflow*. Oleh karena itu, kita akan menggunakan **ListView** dengan cara Refactor. Widget ini memungkinkan kita untuk menerapkan *scrolling* terhadap beberapa item (*children*).

```
ListView(  
  children: [  
    Image.network('https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'),  
    Image.network('https://www.rancaupas.id/wp-content/uploads/2023/08/Ranca-Upas-Igloo-Camp-Ciwidey.jpg'),  
    Image.network('https://awsimages.detik.net.id/community/media/visual/2020/11/11/ranca-upas_169.jpeg?w=1200'),  
  ],  
)
```

Penempatan kode di atas adalah di bagian bawah setelah widget *Container* terakhir, bisa lihat gambar di bawah

```
Container(  
  padding: const EdgeInsets.all(16.0),  
  child: const Text(  
    'Ranca Upas Ciwidey adalah kawasan bumi perkemahan di bawah pengelolaan perhutani. Tempat ini berada di k  
    textAlign: TextAlign.justify,  
    style: TextStyle(  
      fontSize: 16.0,  
      color: Color.fromARGB(255, 84, 1, 133),  
    ), // TextStyle  
  ), // Text  
) // Container  
ListView(  
  children: [  
    Image.network(  
      'https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'), // Image.network  
    Image.network(  
      'https://www.rancaupas.id/wp-content/uploads/2023/08/Ranca-Upas-Igloo-Camp-Ciwidey.jpg'), // Image.net  
    Image.network(  
      'https://awsimages.detik.net.id/community/media/visual/2020/11/11/ranca-upas_169.jpeg?w=1200'), // Imaj  
  ],  
) // ListView
```

Jika kalian menjalankan aplikasi atau melakukan hot reload, aplikasi akan muncul pesan eror pada log. Kenapa ya? *ListView* diletakkan di dalam *Column*, di mana keduanya sama-sama memiliki atribut *height* yang memakan space di sepanjang layar. Sebagai solusi kita perlu memberikan ukuran tinggi yang statis terhadap *ListView*. Namun *ListView* tidak memiliki parameter *height*, lantas bagaimana nih? Caranya, gunakan widget lain yang memiliki parameter *height*. Kalian dapat membungkus widget *ListView* ke dalam *Container* atau pun *SizedBox*. Ukuran tinggi ini nantinya juga digunakan sebagai tinggi *Image* yang tampil. klik widget *ListView* → Klik kanan → Refactor → Wrap with widget → ketik **SizedBox** → tambah parameter *height* dengan nilai 150 → Restart.

```

EdgeInsets(
  height: 150,
  child: ListView(
    children: [
      Image.network(
        'https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'), // Image.network
      Image.network(
        'https://www.rancaupas.id/wp-content/uploads/2023/08/Ranca-Upas-Igloo-Camp-Ciwidey.jpg'), // Image
      Image.network(
        'https://awsimages.detik.net.id/community/media/visual/2020/11/11/ranca-upas_169.jpeg?w=1200'), //
    ],
  ), // ListView
), // SizedBox

```

9. Secara default arah scroll dari **ListView** adalah vertikal. Untuk mengubahnya menjadi horizontal kita cukup menambahkan parameter **scrollDirection** bernilai **Axis.horizontal**.

```

EdgeInsets(
  height: 150,
  child: ListView(
    scrollDirection: Axis.horizontal,
    children: [
      Image.network(
        'https://www.rancaupas.id/wp-c
      Image.network(
        'https://www.rancaupas.id/wp-c
      Image.network(
        'https://awsimages.detik.net.i
    ],
  ), // ListView
), // SizedBox

```

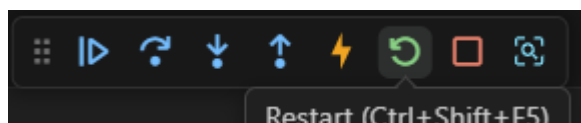
10. Selanjutnya, kita akan sedikit merapikan tampilan gambar supaya terlihat lebih rapi dan menarik. Tambahkan *Padding* pada masing-masing *Image* supaya antar gambar tidak terlalu rapat (Gunakan Refactor → Wrap with Padding).

```

EdgeInsets(
  height: 150,
  child: ListView(
    scrollDirection: Axis.horizontal,
    children: [
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: Image.network(
          'https://www.rancaupas.id/wp-content/uploads/2022/09/DSC05254-2-1024x683.jpg'), // Image.network
        ), // Padding
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: Image.network(
          'https://www.rancaupas.id/wp-content/uploads/2023/08/Ranca-Upas-Igloo-Camp-Ciwidey.jpg'), // Image
        ), // Padding
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: Image.network(
          'https://awsimages.detik.net.id/community/media/visual/2020/11/11/ranca-upas_169.jpeg?w=1200'), //
        ), // Padding
    ],
  ), // ListView
), // SizedBox

```

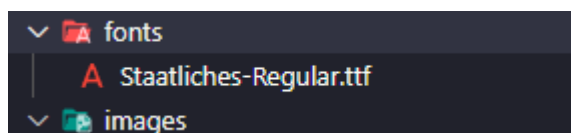
Jalankan Hot Restart





Sekarang gambar di bawah dapat kita scroll secara horizontal.

11. Bagaimana membuat gambar memiliki sudut yang membulat? Sekali lagi, dokumentasi adalah sahabat terbaik dalam mengembangkan aplikasi Flutter. Kalian dapat memanfaatkan mesin pencari untuk menemukan widget sesuai keinginan. Misalnya, dengan memanfaatkan Google atau GPT kalian dapat menemukan bahwa ada *widget* yang memungkinkan gambar memiliki *radius*, yaitu **ClipRRect**. Masukkan *widget Image* kalian sebagai **child** dari **ClipRRect** dan berikan **borderRadius** dengan nilai **BorderRadius.circular()**, maka kalian akan mendapatkan Image dengan sudut yang tak bersiku. **Silahkan dicoba mandiri! Maka nanti hasil akan seperti ini.**
12. Selanjutnya, kita akan menggunakan *custom Font*. Kalian bebas menggunakan *font* kesukaan kalian. Pada contoh ini akan menggunakan font Staatliches. Tambahkan font yang akan digunakan ke dalam *project* dan daftarkan pada *pubspec.yaml*.



```
flutter:

uses-material-design: true

assets:
  - images/

  fonts:
    - family: Staatliches
      fonts:
        - asset: fonts/Staatliches-Regular.ttf
```

13. Tambahkan parameter *fontFamily* pada widget *TextStyle* untuk menerapkan *style* pada Text.

```
child: Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: <Widget>[
    Image.asset('images/ranca-upas.jpg'),
    Container(
      margin: const EdgeInsets.only(top: 16.0),
      child: const Text(
        'Ranca Upas',
        textAlign: TextAlign.center,
        style: TextStyle(fontSize: 30.0, fontFamily: 'Staatliches'),
      ), // Text
    ), // Container
```

14. Jika kalian memiliki beberapa teks dengan *style* yang sama, Kalian dapat menggunakan variabel untuk menyimpan **TextStyle** dan meringkas kode. Tempatkan kode berikut di bawah import **material.dart**

```
var iniFontCustom = const TextStyle(fontFamily: 'Staatliches');
```

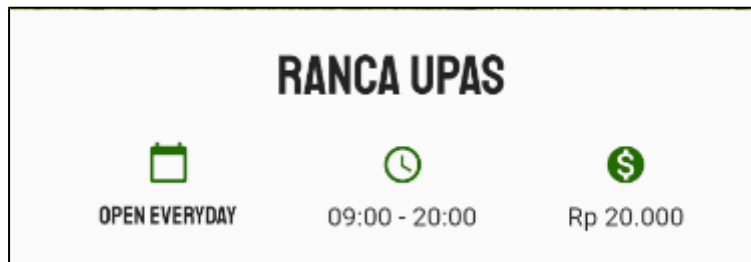
Gunakan variabel tersebut pada masing-masing widget yang membutuhkan. Misal pada widget Text yang berisi Open Everyday

```
Text(
  'Open Everyday',
  style: iniFontCustom,
), // Text
```

Jika terdapat *error* hapus *const* pada *Widget Row*

```
child: const Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    Column(
      children: <Widget>[
        Icon(
          Icons.calendar_today,
          color: Color.fromARGB(255, 34, 107, 0),
        ), // Icon
        SizedBox(height: 8.0),
        Text(
          'Open Everyday',
          style: iniFontCustom,
        ), // Text
      ], // <Widget>[]
    ), // Column
```

Maka hasil akan seperti ini



**Lakukan juga perubahan *font* pada jam dan harga**

15. Hasil sementara praktikum ini, rehat dulu...



16. Sekarang kita akan lanjut dalam penggunaan **Navigation** dan **menampilkan beberapa lokasi wisata di Bandung**, yang selanjutnya kita lakukan adalah membuat halaman baru untuk menampilkan daftar tempat wisata. Buat *file* baru **main\_screen.dart** lalu buat widget untuk halaman **MainScreen**.

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Tempat Wisata Bandung'),
      ),
    );
  }
}
```

17. Jangan lupa untuk mengganti halaman utama yang ditampilkan pada *file* **main.dart**.  
Kemudian tambahkan juga **useMaterial3: false**.

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Tempat Wisata Bandung',
      theme: ThemeData(
        useMaterial3: false,
      ), // ThemeData
      home: const MainScreen(),
    ); // MaterialApp
  }
}
```

18. Sebagai body dari Scaffold kita akan menggunakan widget Card. Widget ini adalah widget material design yang menghasilkan tampilan seperti kartu dengan ujung yang membulat dan bayangan di belakang. Kemudian susun *Row* dan *Column* seperti contoh untuk menyusun *child* dari Card. Kodenya akan seperti berikut pada **main\_screen.dart**:

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Tempat Wisata Bandung'),
      ),
    );
  }
}
```

```

body: Card(
  child: Row(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Image.asset('images/ranca-upas.jpg'),
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          mainAxisAlignment: MainAxisAlignment.min,
          children: <Widget>[
            Text(
              'Ranca Upas',
              style: const TextStyle(fontSize: 16.0),
            ),
            const SizedBox(
              height: 10,
            ),
            Text('Ciwidey'),
          ],
        ),
      ),
    ],
  ),
);
}

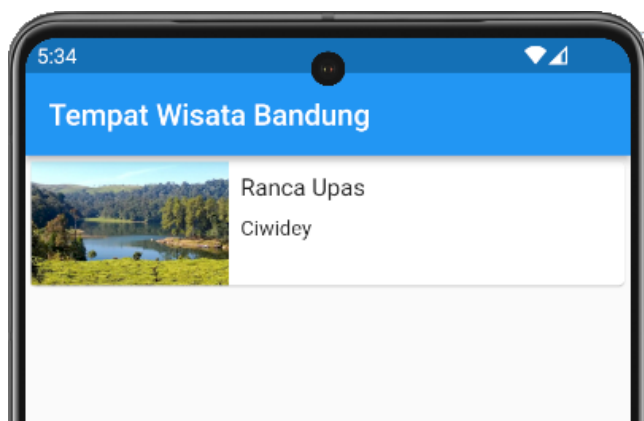
```

19. Jalankan aplikasinya. Aplikasi akan mengalami *overflow* karena aset gambar yang terlalu besar.



Kita bisa saja mengatur tinggi gambar secara manual, namun kali ini kita akan memanfaatkan widget **Expanded** agar tampilan juga dapat menyesuaikan di perangkat yang lebih besar atau kecil. Bungkus masing-masing item dari widget **Row** ke dalam **Expanded**. Berikan parameter **flex** yang menurut kalian cocok pada **main\_screen.dart**.

```
child: Row(  
  crossAxisAlignment: CrossAxisAlignment.start,  
  children: <Widget>[  
    Expanded(  
      flex: 1,  
      child: Image.asset('images/ranca-upas.jpg'),  
    ),  
    Expanded(  
      flex: 2,  
      child: Padding(  
        padding: const EdgeInsets.all(8.0),  
        child: Column(  
          crossAxisAlignment: CrossAxisAlignment.start,  
          mainAxisAlignment: MainAxisAlignment.min,  
          children: <Widget>[  
            Text(  
              'Ranca Upas',  
              style: const TextStyle(fontSize: 16.0),  
            ),  
            const SizedBox(  
              height: 10,  
            ),  
            Text('Ciwidey'),  
          ],  
        ),  
      ),  
    ),  
  ],  
)
```



20. Item pertama kalian sudah siap. Selanjutnya kita akan membuat kartu ini bisa diklik untuk berpindah ke halaman detail. Kita bisa menggunakan widget **InkWell** yang menyediakan parameter **onTap**. Pindahkan widget **Card** menjadi child dari **InkWell**. Gunakan refactor yaaa guys

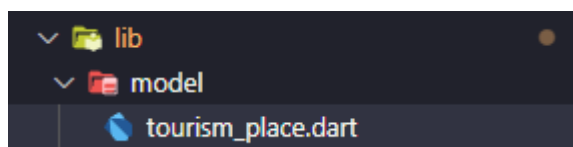
```
class MainScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Wisata Bandung'),
      ),
      body: InkWell(
        onTap: () {},
        child: Card(...),
      ),
    );
  }
}
```

21. Parameter **onTap** menerima argumen berupa sebuah *anonymous function*. Di sini kita akan menambahkan **Navigator** untuk berpindah ke halaman detail.

```
onTap: () {
  Navigator.push(context, MaterialPageRoute(builder: (context) {
    return DetailScreen();
  }));
},
```

Jalankan aplikasi. Seharusnya sampai langkah ini aplikasi kalian sudah dapat berpindah halaman ketika item diklik.

22. Selanjutnya kita akan menampilkan beberapa item ke MainScreen. Di kelas ini kita masih menggunakan data statis dan lokal yang disimpan pada objek *List*. Sebelumnya, buatlah kelas sebagai *blueprint* untuk menyimpan objek tempat wisata kita. Buat folder baru di dalam folder **lib**, lalu berikan nama **model**. Di dalam folder model buat *file* dart bernama **tourism\_place.dart**.



23. Di dalam **tourism\_place.dart** buat data class yang akan menjadi blueprint objek tempat wisata.

```
class TourismPlace {
  String name;
  String location;
  String description;
  String openDays;
  String openTime;
  String ticketPrice;
  String imageAsset;
  List<String> imageUrls;

  TourismPlace({
    required this.name,
    required this.location,
    required this.description,
    required this.openDays,
    required this.openTime,
    required this.ticketPrice,
    required this.imageAsset,
    required this.imageUrls,
  });
}
```

24. Siapkan data statis yang ingin ditampilkan, kalian dapat menyalin kode berikut dan taruh di file **tourism\_place.dart** paling bawah.

Copy code dari [sini](#)

Jangan lupa untuk menambahkan *import file* **tourism\_place.dart** pada *file* **main\_screen.dart**.

```
import 'model/tourism_place.dart';
```

25. Selanjutnya kita akan menampilkan variabel **tourismPlaceList** di atas menjadi item card yang dapat diklik. Tambahkan widget **ListView** sebagai *body* dari **Scaffold** di **main\_screen.dart**. Klik kanan pada widget **Inkwell** → Refactor → Wrap with builder → Ubah builder menjadi **ListView.builder**. Jangan lupa untuk mengganti tampilan item secara dinamis sesuai data dari objek **TourismPlace**.

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Tempat Wisata Bandung'),

```

```

    ),
    body: ListView.builder(
      itemBuilder: (context, index) {
        final TourismPlace place = tourismPlaceList[index];
        return InkWell(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) {
              return DetailScreen();
            }));
          },
          child: Card(
            child: Row(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Expanded(
                  flex: 1,
                  child: Image.asset(place.imageAsset),
                ),
                Expanded(
                  flex: 2,
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      mainAxisAlignment: MainAxisAlignment.min,
                      children: <Widget>[
                        Text(
                          place.name,
                          style: const TextStyle(fontSize: 16.0),
                        ),
                        const SizedBox(
                          height: 10,
                        ),
                        Text(place.location),
                      ],
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
      itemCount: tourismPlaceList.length,
    ),
  );
}
}

```

Jalankan aplikasi untuk melihat hasil perubahan.



26. Agar halaman detail bisa menampilkan informasi sesuai tempat wisata yang dipilih, kita perlu mengirimkan data **TourismPlace** melalui *constructor*. Buka file **detail\_screen.dart** lalu tambahkan variabel serta *constructor*-nya.

```
import 'model/tourism_place.dart';

class DetailScreen extends StatelessWidget {
  final TourismPlace place;

  const DetailScreen({super.key, required this.place});

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

Jangan lupa untuk menambahkan data **place** pada constructor class MainScreen.

```
Navigator.push(context, MaterialPageRoute(builder: (context) {
  return DetailScreen(place: place);
})); // MaterialPageRoute
```

27. Coba untuk lakukan perubahan tampilan item secara dinamis sesuai data dari objek **TourismPlace** yang ada pada **detail\_screen.dart** seperti pada **main\_screen.dart**. Seperti:
- place.name
  - place.imageAsset
  - place.openDays
  - place.openTime
  - place.ticketPrice
  - place.description

## Contoh

```
class DetailScreen extends StatelessWidget {
  final TourismPlace place;

  const DetailScreen({Key? key, required this.place}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: SingleChildScrollView(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: <Widget>[
              Image.asset(place.imageAsset),
              Container(
                margin: const EdgeInsets.only(top: 16.0),
                child: Text(
                  place.name,
                  textAlign: TextAlign.center,
```

Untuk widget **SizedBox** (paling bawah) yang berisi list gambar horizontal tidak usah diubah terlebih dahulu. Nanti akan ada di tahap selanjutnya

```
SizedBox(
  height: 150,
  child: ListView(
    scrollDirection: Axis.horizontal,
    children: [
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(15),
          child: Image.network(
            'https://cdn.idntimes.com/content-images/post/20201106/19437160-453398458365585-5644109604204838912-n-
          ), // ClipRRect
        ), // Padding
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(15),
          child: Image.network(
            'https://cdn.idntimes.com/content-images/post/20201106/34091980-2204188269801480-248274445720879104-n-
          ), // ClipRRect
        ), // Padding
      Padding(
        padding: const EdgeInsets.all(4.0),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(15),
          child: Image.network(
            'https://cdn.idntimes.com/content-images/post/20201106/39408216-228473327819821-1612494975352700928-n-
          ), // ClipRRect
        ), // Padding
      ],
    ), // ListView
  ), // SizedBox
```

28. Jika sudah, coba lihat ketika klik menu selain Ranca Upas tetapi gambar yang tampil masih statis. Namun untuk informasi lainnya seharusnya susah sesuai dengan menu yang diklik, misal klik Observatorium Bosscha maka yang akan tampil adalah data mengenai Observatorium Bosscha.



Untuk mengubahnya kita buka *file detail\_screen.dart* pada widget **SizedBox** paling bawah ubah kode yang lama menjadi seperti ini untuk menampilkan gambar dinamis

```

SizedBox(
  height: 150,
  child: ListView(
    scrollDirection: Axis.horizontal,
    children: place.imageUrls.map((url) {
      return Padding(
        padding: const EdgeInsets.all(4.0),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(10),
          child: Image.network(url),
        ),
      );
    }).toList(),
  ),
),

```

29. Selanjutnya, kita akan menambahkan tombol navigasi untuk kembali ke halaman daftar tempat wisata. Tombol ini akan kita taruh di atas gambar utama atau gambar dari aset. Kita akan menggunakan widget **Stack**. Widget ini digunakan untuk menyusun widget seperti Column atau Row, bedanya widget pada Stack disusun secara bertumpuk (stacked). Ubah kode kalian menjadi seperti berikut:

```
Stack(
  children: <Widget>[
    Image.asset(place.imageAsset),
    IconButton(icon: const Icon(Icons.arrow_back), onPressed: () {}))
  ],
),
```

Replace baris di bawah dengan kode di atas pada file **detail\_screen.dart**

```
return Scaffold(
  body: SafeArea(
    child: SingleChildScrollView(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Image.asset(place.imageAsset),
          Container(
            margin: const EdgeInsets.only(top: 16.0),
```

Tambahkan fungsionalitas agar ketika icon back ini diklik akan kembali ke halaman sebelumnya.

```
IconButton(
  icon: const Icon(Icons.arrow_back),
  onPressed: () {
    Navigator.pop(context);
  },
),
```



30. Lakukan juga beberapa perubahan tampilan supaya ikon navigasi tidak bertabarakkan dengan latar belakangnya. Yang pertama adalah membungkus Widget **IconButton** di dalam Widget **CircleAvatar**, dengan cara **klik kanan di Widget IconButton → Refactor → Wrap with Widget → Ketik CircleAvatar → Tambah parameter backgroundColor**. Kemudian ubah **backgroundColor** dari **CircleAvatar** menjadi abu dan warna panah dari **IconButton** menjadi putih. Sehingga akan terlihat seperti di bawah

```
children: <Widget>[
  Image.asset(place.imageAsset),
  CircleAvatar(
    backgroundColor: Colors.grey,
    child: IconButton(
      icon: const Icon(Icons.arrow_back),
      color: Colors.white,
      onPressed: () {
        Navigator.pop(context);
      },
    ),
  ),
],
```

Selanjutnya membungkus Widget **CircleAvatar** di dalam Widget **Padding**, dengan cara klik kanan di Widget **CircleAvatar** → Refactor → Wrap with Padding. Sehingga akan terlihat seperti di bawah

```
children: <Widget>[
  Image.asset(place.imageAsset),
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: CircleAvatar(
      backgroundColor: Colors.grey,
      child: IconButton(
        icon: const Icon(Icons.arrow_back),
        color: Colors.white,
        onPressed: () {
          Navigator.pop(context);
        },
      ),
    ),
  ),
],
```

Jika dirasa tidak puas dengan background abu di notification bar



Kita bisa hapus Widget **SafeArea** pada body dengan cara klik kanan di Widget **SafeArea** → Refactor → Remove this widget

```
return Scaffold(
  body: SafeArea(
```

Kemudian buat widget **SafeArea** pada Widget **Padding** setelah **imageAsset** dengan cara klik kanan di Widget **Padding** → Refactor → Wrap with widget → Ketik **SafeArea**.

```
children: <Widget>[
  Image.asset(place.imageAsset),
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: CircleAvatar(
      backgroundColor: Colors.grey,
      child: IconButton(
        icon: const Icon(Icons.arrow_back),
        color: Colors.white,
        onPressed: () {
          Navigator.pop(context);
        },
      ), // IconButton
    ), // CircleAvatar
  ), // Padding
], // <Widget>[]
```

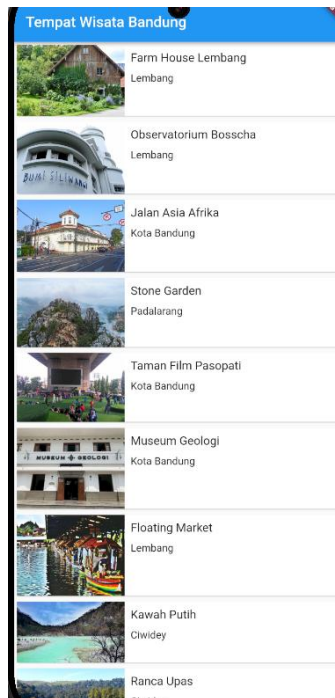
Menjadi seperti ini

```
children: <Widget>[
  Image.asset(place.imageAsset),
  SafeArea(
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: CircleAvatar(
        backgroundColor: Colors.grey,
        child: IconButton(
          icon: const Icon(Icons.arrow_back),
          color: Colors.white,
          onPressed: () {
            Navigator.pop(context);
          },
        ), // IconButton
      ), // CircleAvatar
    ), // Padding
  ), // SafeArea
], // <Widget>[]
```

Tadaaa berhasil....



31. Tambahan pada gambar di bawah pada **main\_screen.dart** bisa dilihat ukuran gambarnya ada yang besar dan kecil. Agar gambar di setiap item memiliki ukuran yang seragam (tinggi dan lebar konsisten), kalian bisa membungkus **Image.asset** dengan **AspectRatio**. Dengan begitu, gambar akan dipotong dan dipaskan dalam rasio aspek tertentu, sehingga semua gambar memiliki dimensi yang serupa tanpa mengubah aspek rasio terlalu banyak. Lalu menggunakan **BoxFit.cover** untuk memastikan gambar memenuhi ruang tanpa terdistorsi.



Berikut contoh cara melakukannya cari **widget Image** berikut

```

    SizedBox(
      height: 150,
      child: ListView(
        scrollDirection: Axis.horizontal,
        children: place.imageUrls.map((url) {
          return Padding(
            padding: const EdgeInsets.all(4.0),
            child: ClipRRect(
              borderRadius: BorderRadius.circular(10),
              child: Image.network(url),
            ), // ClipRRect
          ); // Padding
        }).toList(),
      ), // ListView
    ), // SizedBox
  
```

Klik widget Image → klik kanan → Refactor → Wrap with Widget → Ketik AspectRatio  
 → tambahkan property aspectRatio dengan ukuran 16 : 9 sebagai berikut

```

    return Padding(
      padding: const EdgeInsets.all(4.0),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(10),
        child: AspectRatio(
          aspectRatio: 16 / 9,
          child: Image.network(url),
        ), // AspectRatio
      ), // ClipRRect
    ); // Padding
  
```

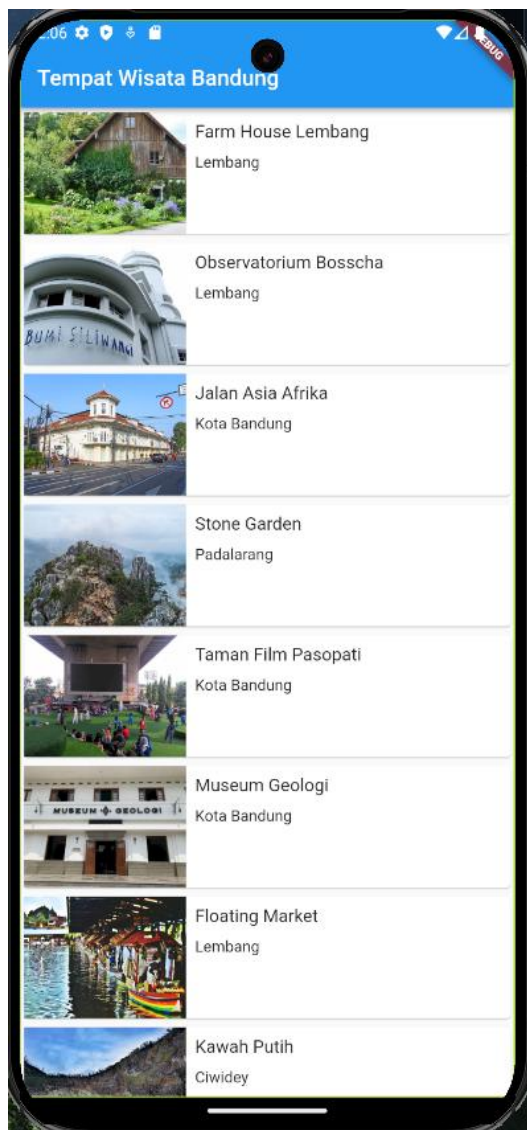
Kemudian sekarang tambahkan property fit di dalam Widget Image seperti berikut

```

    SizedBox(
      height: 150,
      child: ListView(
        scrollDirection: Axis.horizontal,
        children: place.imageUrls.map((url) {
          return Padding(
            padding: const EdgeInsets.all(4.0),
            child: ClipRRect(
              borderRadius: BorderRadius.circular(10),
              child: AspectRatio(
                aspectRatio: 16 / 9,
                child: Image.network(url, fit: BoxFit.cover),
              ), // AspectRatio
            ), // ClipRRect
          ); // Padding
        }).toList(),
      ), // ListView
    ), // SizedBox
  ), // StatelessWidget
);

```

Sekarang semua gambar ukurannya sudah sama



32. Selanjutnya adalah kita juga akan mengubah ukuran gambar pada `detail_screen.dart` agar sama semua. Perhatikan gambar di bawah dengan ukuran yang tidak sama



Berikut contoh cara melakukannya cari **widget Image** berikut

```
child: Card(
  child: Row(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Expanded(
        flex: 1,
        child: Image.asset(place.imageAsset),
      ), // Expanded
      Expanded(
        flex: 2,
        child: Padding(
```

Klik widget Image → klik kanan → Refactor → Wrap with Widget → Ketik AspectRatio  
→ tambahkan property aspectRatio dengan ukuran 4 : 3 sebagai berikut

```
Expanded(
  flex: 1,
  child: AspectRatio(
    aspectRatio: 4 / 3,
    child: Image.asset(place.imageAsset),
  ), // AspectRatio
), // Expanded
```

Kemudian sekarang tambahkan property fit di dalam Widget Image seperti berikut

```
Expanded(
  flex: 1,
  child: AspectRatio(
    aspectRatio: 4 / 3,
    child: Image.asset(place.imageAsset, fit: BoxFit.cover),
  ), // AspectRatio
), // Expanded
```

Sekarang gambar di bagian bawah ukurannya telah sama semua



33. Ok tahap di atas adalah tahap akhir dari praktikum ini

### **Pengumpulan Hasil :**

Jika sudah berhasil, hasil praktikum dimasukkan ke dalam github masing-masing. Buat folder dengan nama **Pertemuan05** kemudian di dalamnya buat 2 folder yaitu **folder Praktikum** dan **Output**.

Isi folder **Praktikum** adalah aplikasi flutter yang sudah dibuat, tetapi **sebelum melakukan push ke github** wajib jalankan perintah “**flutter clean**” pada VSCode. Isi folder Output adalah hasil dari aplikasi flutter yang sudah dibuat.

Setelah melakukan perintah **flutter clean** kalian bisa langsung push ke dalam repository kalian.