**Button**

Kali ini kita akan belajar menggunakan widget button. Widget button ini adalah widget yang dapat menerima trigger sentuhan atau dapat melakukan suatu fungsi ketika disentuh, widget-widget button tersebut antara lain:

**ElevatedButton**

ElevatedButton merupakan bagian dari Material Design widget dari Flutter. Untuk menggunakan ElevatedButton caranya seperti berikut:

1. ElevatedButton(
2. onPressed: (){
3. // Aksi ketika button diklik
4. },
5. child: Text("Tombol"),
6. );

Pada kode di atas ElevatedButton memiliki 2 parameter yaitu onPressed dan child. Parameter *onPressed* merupakan sebuah *function event* ketika tombol ditekan dan sebenarnya ada *event* lain seperti *onLongPress* dan *onHighlightChanged*. Parameter *child* diisi oleh widget pada umumnya.

**TextButton**

TextButton merupakan widget button yang memiliki tampilan yang polos selayaknya Text. TextButton umumnya digunakan pada toolbars, dialog, atau bersama komponen button lain. Contoh kode dari TextButton adalah seperti berikut:

1. TextButton(
2. onPressed: () {
3. // Aksi ketika button diklik
4. },
5. child: Text('Text Button'),
6. )

Sama halnya ElevatedButton, TextButton juga memiliki parameter onPressed dan child.

**OutlinedButton**

OutlinedButton juga merupakan bagian dari material design yang menyediakan tampilan TextButton dengan tambahan outline. OutlinedButton umumnya digunakan untuk tombol atau aksi yang penting, tetapi bukan aksi utama dalam aplikasi.

Berikut ini adalah contoh widget OutlinedButton:

1. OutlinedButton(
2. onPressed: () {
3. // Aksi ketika button diklik
4. },
5. child: Text('Outlined Button'),
6. )

Tampilan OutlinedButton sendiri akan seperti berikut:

**IconButton**

IconButton merupakan widget button dengan icon. Tak seperti widget tombol lainnya, widget IconButton ini tidak memiliki child. Perhatikan kode di bawah ini:

1. IconButton(
2. icon: Icon(Icons.volume\_up),
3. tooltip: 'Increase volume by 10',
4. onPressed: () {
5. // Aksi ketika button diklik
6. },
7. )

Seperti yang kita lihat di atas, IconButton tidak menggunakan child untuk isi (*content*) melainkan menggunakan parameter icon dan *tooltip* (penunjuk) untuk memberikan hint pada tombol.

**DropdownButton**

DropdownButton merupakan tombol yang saat diklik, akan muncul *pop-up* daftar beberapa item yang dapat kita pilih salah satu. Berikut contoh kodenya:

1. class FirstScreen extends StatefulWidget {
2. @override
3. \_FirstScreenState createState() => \_FirstScreenState();
4. }
6. class \_FirstScreenState extends State<FirstScreen> {
7. String? language;
9. @override
10. Widget build(BuildContext context) {
11. return Scaffold(
12. appBar: AppBar(
13. title: Text('First Screen'),
14. ),
15. body: DropdownButton<String>(
16. items: const <DropdownMenuItem<String>>[
17. DropdownMenuItem<String>(
18. value: 'Dart',
19. child: Text('Dart'),
20. ),
21. DropdownMenuItem<String>(
22. value: 'Kotlin',
23. child: Text('Kotlin'),
24. ),
25. DropdownMenuItem<String>(
26. value: 'Swift',
27. child: Text('Swift'),
28. ),
29. ],
30. value: language,
31. hint: Text('Select Language'),
32. onChanged: (String? value) {
33. setState(() {
34. language = value;
35. });
36. },
37. ),
38. );
39. }
40. }

Pada contoh tersebut DropdownButton tidak menggunakan child maupun children, akan tetapi menggunakan items di mana berisi list dari widget DropdownMenuItem. Pada widget DropdownMenuItem terdapat child untuk tiap itemnya dan value yang ada pada DropdownMenuItem adalah nilai dari tiap itemnya. Nantinya akan dibutuhkan parameter onChanged ketika ada perubahan atau ketika memilih salah satu dari item tersebut dan mengubah nilai language atau value dari DropdownButton tersebut. Sedangkan hint berfungsi ketika nilai value dari DropdownButton *null* atau kosong.

Selengkapnya tentang berbagai widget Button, bacalah pada tautan berikut:

* [Button Material Components](https://flutter.dev/docs/development/ui/widgets/material#Buttons)
* [ElevatedButton Class](https://api.flutter.dev/flutter/material/ElevatedButton-class.html)
* [TextButton Class](https://api.flutter.dev/flutter/material/TextButton-class.html)
* [OutlinedButton](https://api.flutter.dev/flutter/material/OutlinedButton-class.html)
* [IconButton Class](https://api.flutter.dev/flutter/material/IconButton-class.html)
* [DropdownButton Class](https://api.flutter.dev/flutter/material/DropdownButton-class.html)

**Input Widget**

Salah satu bentuk interaksi dengan pengguna adalah dengan menerima input. Ada beberapa input widget yang bisa digunakan supaya pengguna bisa berinteraksi dengan aplikasi. Perhatikan bahwa input pengguna ini berkaitan dengan state yang dapat sering berubah. Karena itu umumnya input widget akan ditempatkan di dalam StatefulWidget.

**TextField**

TextField merupakan sebuah widget yang digunakan untuk menerima input berupa teks yang berasal dari *keyboard*. Terdapat beberapa cara yang bisa Anda gunakan untuk mendapatkan nilai dari TextField. Salah satunya adalah melalui parameter onChanged.

1. String \_name = '';
3. TextField(
4. onChanged: (String value) {
5. setState(() {
6. \_name = value;
7. });
8. },
9. )

Parameter *onChanged* berisi sebuah fungsi yang akan dipanggil setiap terjadi perubahan inputan pada TextField. Pada fungsi ini, kita dapat mengubah nilai variabel state dengan memanggil fungsi setState().

Jika Anda tidak ingin mengambil nilai setiap perubahan, tetapi hanya ketika seluruh input sudah selesai di-*submit*, Anda dapat menggunakan parameter onSubmitted seperti berikut:

1. String \_name = '';
3. TextField(
4. onSubmitted: (String value) {
5. setState(() {
6. \_name = value;
7. });
8. },
9. )

Cara lain yang bisa kita gunakan adalah dengan TextEditingController. Dengan controller, kita cukup membuat variabel TextEditingController lalu menambahkannya ke widget TextField.

1. TextEditingController \_controller = TextEditingController();
3. TextField(
4. controller: \_controller,
5. ),

Ketika menggunakan controller, pastikan untuk menghapus controller ketika halaman atau widget sudah tidak digunakan. Ini bertujuan supaya tidak menimbulkan kebocoran memori (*memory leak*).

1. @override
2. void dispose() {
3. \_controller.dispose();
4. super.dispose();
5. }

Berikut ini adalah contoh penerapan widget TextField:

Untuk membuat TextField seperti di atas, Anda bisa menggunakan kode seperti berikut:

* [**onChanged**](https://www.dicoding.com/academies/159/tutorials/8606?from=6504#tab1-code1)
* [Controller](https://www.dicoding.com/academies/159/tutorials/8606?from=6504#tab1-code2)

1. class \_FirstScreenState extends State<FirstScreen> {
2. String \_name = '';
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. appBar: AppBar(
8. title: Text('First Screen'),
9. ),
10. body: Padding(
11. padding: const EdgeInsets.all(16.0),
12. child: Column(
13. children: [
14. TextField(
15. decoration: InputDecoration(
16. hintText: 'Write your name here...',
17. labelText: 'Your Name',
18. ),
19. onChanged: (String value) {
20. setState(() {
21. \_name = value;
22. });
23. },
24. ),
25. SizedBox(height: 20),
26. ElevatedButton(
27. onPressed: () {
28. showDialog(
29. context: context,
30. builder: (context) {
31. return AlertDialog(
32. content: Text('Hello, $\_name'),
33. );
34. });
35. },
36. child: Text('Submit'),
37. )
38. ],
39. ),
40. ),
41. );
42. }
43. }

**Switch**

Switch merupakan inputan yang mengembalikan nilai boolean true atau false. Perhatikan contoh berikut:

1. class \_FirstScreenState extends State<FirstScreen> {
2. bool lightOn = false;
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. appBar: AppBar(
8. title: Text('First Screen'),
9. ),
10. body: Switch(
11. value: lightOn,
12. onChanged: (bool value) {
13. setState(() {
14. lightOn = value;
15. });
17. ScaffoldMessenger.of(context).showSnackBar(
18. SnackBar(
19. content: Text(lightOn ? 'Light On' : 'Light Off'),
20. duration: Duration(seconds: 1),
21. ),
22. );
23. },
24. ),
25. );
26. }
27. }

Pada contoh tersebut value dari Switch berupa boolean di mana ketika boolean tersebut false maka Switch akan berada pada posisi nonaktif. Switch umumnya digunakan sebagai konfigurasi on/off pada halaman setting.

**Radio**

Radio merupakan inputan yang digunakan untuk memilih salah satu dari beberapa pilihan dalam suatu kelompok. Berikut contohnya:

1. class \_FirstScreenState extends State<FirstScreen> {
2. String language;
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. appBar: AppBar(
8. title: Text('First Screen'),
9. ),
10. body: Column(
11. mainAxisSize: MainAxisSize.min,
12. children: <Widget>[
13. ListTile(
14. leading: Radio<String>(
15. value: 'Dart',
16. groupValue: language,
17. onChanged: (String value) {
18. setState(() {
19. language = value;
20. showSnackbar();
21. });
22. },
23. ),
24. title: Text('Dart'),
25. ),
26. ListTile(
27. leading: Radio<String>(
28. value: 'Kotlin',
29. groupValue: language,
30. onChanged: (String value) {
31. setState(() {
32. language = value;
33. showSnackbar();
34. });
35. },
36. ),
37. title: Text('Kotlin'),
38. ),
39. ListTile(
40. leading: Radio<String>(
41. value: 'Swift',
42. groupValue: language,
43. onChanged: (String value) {
44. setState(() {
45. language = value;
46. showSnackbar();
47. });
48. },
49. ),
50. title: Text('Swift'),
51. ),
52. ],
53. ),
54. );
55. }
57. void showSnackbar() {
58. ScaffoldMessenger.of(context).showSnackBar(
59. SnackBar(
60. content: Text('$language selected'),
61. duration: Duration(seconds: 1),
62. ),
63. );
64. }
65. }

Pada contoh tersebut terdapat variable language yang digunakan pada groupValue tiap Radio. Language inilah yang menyimpan nilai Radio yang dipilih. Nilainya akan berubah ketika fungsi onChanged terpanggil.

**Checkbox**

Checkbox merupakan inputan benar atau salah. Checkbox akan berisi centang jika nilainya adalah benar dan kosong jika salah. Seperti pada contoh berikut:

1. class \_FirstScreenState extends State<FirstScreen> {
2. bool agree = false;
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. appBar: AppBar(
8. title: Text('First Screen'),
9. ),
10. body: ListTile(
11. leading: Checkbox(
12. value: agree,
13. onChanged: (bool value) {
14. setState(() {
15. agree = value;
16. });
17. },
18. ),
19. title: Text('Agree / Disagree'),
20. ),
21. );
22. }
23. }

Kode di atas jika dijalankan akan tampil seperti berikut:

Ada beberapa tautan yang dapat Anda baca untuk memahami tentang widget-widget input yang ada pada Flutter, antara lain:

* [Input and selections widgets](https://flutter.dev/docs/development/ui/widgets/material#Input%20and%20selections)
* [TextField Class](https://api.flutter.dev/flutter/material/TextField-class.html)
* [Switch Class](https://api.flutter.dev/flutter/material/Switch-class.html)
* [Radio Class](https://api.flutter.dev/flutter/material/Radio-class.html)
* [Checkbox Class](https://api.flutter.dev/flutter/material/Checkbox-class.html)

# Image

Dalam pengembangan suatu aplikasi kita tidak akan lepas dari image atau gambar untuk membuat tampilan semakin menarik. Pada materi kali ini kita akan belajar bagaimana menampilkan gambar dari internet dan project asset.

### Image.network

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

1. 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:

1. class FirstScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(
6. title: Text('First Screen'),
7. ),
8. body: Center(
9. child: Image.network(
10. 'https://picsum.photos/200/300',
11. width: 200,
12. height: 200,
13. ),
14. ),
15. );
16. }
17. }

Pada kode di atas kita panggil method Image.network dengan url https://picsum.photos/200/300 lalu beri width dan height masing-masing 200. Sehingga hasilnya seperti berikut:

### Image.asset

Selain melalui internet, kita juga dapat menampilkan gambar yang bersumber dari asset project.. Asset di sini berupa gambar-gambar yang nantinya didaftarkan pada project. Untuk mendaftarkan asset gambar pada project kita harus menambahkannya pada berkas **pubspec.yaml**.

Pertama kita harus menambahkan terlebih dahulu gambar yang akan didaftarkan ke dalam folder project kita. Saat ini Flutter mendukung beberapa jenis format gambar, seperti JPEG, PNG, GIF, Animated GIF, WebP, Animated WebP, BMP, dan WBMP. Di luar format tersebut, Flutter akan memanfaatkan API dari masing-masing platform. Jika platform native mendukung format gambar yang digunakan, maka gambar tersebut akan bisa di-render oleh Flutter.

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

Masukkan berkas gambar yang ingin Anda gunakan ke dalam folder **image**. Sebagai contoh kita menggunakan gambar bernama **android.png**.

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:

1. ...
2. flutter:
4. uses-material-design: true
6. # To add assets to your application, add an assets section, like this:
7. # assets:
8. #  - images/a\_dot\_burr.jpeg
9. #  - images/a\_dot\_ham.jpeg
11. ...

Daftarkan asset gambar seperti berikut:

1. ...
2. flutter:
3. uses-material-design: true
5. assets:
6. - images/android.png
7. ...

Hapus juga tanda pagar (#) atau komentar yang tidak diperlukan. Perhatikan pula indentasi kodenya. assets: berada sejajar dengan uses-material-design: yaitu berjarak 2 spasi dari ujung dan berada di dalam flutter: sedangkan - images/android.png 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 **android.png** pada folder images, maka lokasi gambar tersebut adalah **images/android.png**.

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:

1. ...
2. flutter:
4. uses-material-design: true
6. assets:
7. - images/
8. ...

Setelah menambahkan assets, kita harus me-refresh **pubspec.yaml** dengan cara save file pubspec.yaml bila menggunakan Visual Studio Code atau menekan 'Packages get' yang ada di pojok kanan atas untuk Android Studio.

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:

1. Image.asset(lokasi\_asset)

Contoh dalam kodenya akan seperti berikut:

1. class FirstScreen extends StatelessWidget {
3. @override
4. Widget build(BuildContext context) {
5. return Scaffold(
6. appBar: AppBar(
7. leading: IconButton(
8. icon: Icon(Icons.menu, color: Colors.white,),
9. ),
10. title: Text('First Screen'),
11. actions: <Widget>[
12. IconButton(
13. icon: Icon(Icons.search, color: Colors.white,),
14. )
15. ],
16. ),
17. body: Center(
18. child: Image.asset('images/android.png', width: 200, height: 200),
19. ),
20. floatingActionButton: FloatingActionButton(
21. child: Icon(Icons.add),
22. ),
23. );
24. }
25. }

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

Untuk mempelajari widget Image lebih lanjut, Anda dapat membaca dokumentasinya pada tautan [Image Class](https://api.flutter.dev/flutter/widgets/Image-class.html).

## Font

Dalam pengembangan suatu aplikasi, seorang User Interface desainer dapat menggunakan font berbeda dengan default font yang ada. Sebagai pengembang aplikasi kita diharuskan menambahkan font pada aplikasi yang dirancang oleh desainer agar sesuai dengan desain User Interface.

Pada pembelajaran kali ini kita akan belajar bagaimana menambahkan font pada Flutter. Sebelum kita memulai pembelajaran, kita akan mengunduh font yang ada di internet atau menggunakan font yang telah dimiliki. Pada contoh ini kita akan mengunduh salah satu font dari [Google Fonts](https://fonts.google.com/) yaitu [Oswald](https://fonts.google.com/specimen/Oswald).

### 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:

### Mendaftarkan Font di pubspec.yaml

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

1. flutter:
3. uses-material-design: true
4. assets:
5. - images/
7. fonts:
8. - family: Oswald
9. fonts:
10. - 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.

### 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**:

1. 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.

### 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:

1. class MyApp extends StatelessWidget {
2. // This widget is the root of your application.
3. @override
4. Widget build(BuildContext context) {
5. return MaterialApp(
6. title: 'Flutter Demo',
7. theme: ThemeData(
8. fontFamily: 'Oswald',
9. primarySwatch: Colors.blue,
10. ),
11. home: FirstScreen(),
12. );
13. }
14. }

## ListView

Pada Codelab kedua kita telah menggunakan dan menyinggung sedikit tentang widget **ListView**. Widget ini digunakan untuk menampilkan beberapa item dalam bentuk baris atau kolom dan bisa di-scroll.

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

1. class ScrollingScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. body: ListView(
6. children: <Widget>[
7. Container(
8. height: 250,
9. decoration: BoxDecoration(
10. color: Colors.grey,
11. border: Border.all(color: Colors.black),
12. ),
13. child: Center(
14. child: Text(
15. '1',
16. style: TextStyle(fontSize: 50),
17. ),
18. ),
19. ),
20. Container(
21. height: 250,
22. decoration: BoxDecoration(
23. color: Colors.grey,
24. border: Border.all(color: Colors.black),
25. ),
26. child: Center(
27. child: Text(
28. '2',
29. style: TextStyle(fontSize: 50),
30. ),
31. ),
32. ),
33. Container(
34. height: 250,
35. decoration: BoxDecoration(
36. color: Colors.grey,
37. border: Border.all(color: Colors.black),
38. ),
39. child: Center(
40. child: Text(
41. '3',
42. style: TextStyle(fontSize: 50),
43. ),
44. ),
45. ),
46. Container(
47. height: 250,
48. decoration: BoxDecoration(
49. color: Colors.grey,
50. border: Border.all(color: Colors.black),
51. ),
52. child: Center(
53. child: Text(
54. '4',
55. style: TextStyle(fontSize: 50),
56. ),
57. ),
58. ),
59. ],
60. ),
61. );
62. }
63. }

Ketika dijalankan, aplikasi akan menjadi seperti berikut:

### Menampilkan Item Secara Dinamis

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

Misalnya kita ingin menampilkan daftar angka dari 1 sampai 10.

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

Caranya, masukkan variabel atau list Anda 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 atau lambda.

1. class ScrollingScreen extends StatelessWidget {
2. final List<int> numberList = <int>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. body: ListView(
8. children: numberList.map((number) {});
9. ),
10. );
11. }
12. }

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

1. class ScrollingScreen extends StatelessWidget {
2. final List<int> numberList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. body: ListView(
8. children: **numberList.map((number) {**
9. **return Container(**
10. **height: 250,**
11. **decoration: BoxDecoration(**
12. **color: Colors.grey,**
13. **border: Border.all(color: Colors.black),**
14. **),**
15. **child: Center(**
16. **child: Text(**
17. **'$number', // Ditampilkan sesuai item**
18. **style: TextStyle(fontSize: 50),**
19. **),**
20. **),**
21. **);**
22. **}).toList(),**
23. ),
24. );
25. }
26. }

Perhatikan di akhir kita perlu mengembalikan fungsi map menjadi objek List lagi dengan fungsi .toList(). Lakukan **hot reload** pada aplikasi Anda untuk melihat hasil perubahan.

### 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:

1. ListView.builder(
2. itemBuilder: (BuildContext context, int index) {
3. return Container(
4. height: 250,
5. decoration: BoxDecoration(
6. color: Colors.grey,
7. border: Border.all(color: Colors.black),
8. ),
9. child: Center(
10. child: Text(
11. '${numberList[index]}',
12. style: TextStyle(fontSize: 50),
13. ),
14. ),
15. );
16. },
17. itemCount: numberList.length,
18. ),

### 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:

1. ListView.separated(
2. itemBuilder: (BuildContext context, int index) {
3. return Container(
4. height: 250,
5. decoration: BoxDecoration(
6. color: Colors.grey,
7. border: Border.all(color: Colors.black),
8. ),
9. child: Center(
10. child: Text(
11. '${numberList[index]}',
12. style: TextStyle(fontSize: 50),
13. ),
14. ),
15. );
16. },
17. separatorBuilder: (BuildContext context, int index) {
18. return Divider();
19. },
20. itemCount: numberList.length,
21. ),

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

Untuk mempelajari fitur ListView selengkapnya dapat Anda baca pada tautan dokumentasi [ListView Class](https://api.flutter.dev/flutter/widgets/ListView-class.html).

## Expanded & Flexible

Sejauh ini kita telah mempelajari beberapa widget dasar dan bagaimana menyusunnya secara horizontal maupun vertikal. Dalam pengembangan aplikasi mobile kita tahu bahwa terdapat banyak sekali perangkat dengan ukuran layar yang berbeda pula. Untuk itu penting bagi kita untuk bisa menyusun tampilan yang responsif terhadap ukuran layar.

Kira-kira bagaimana Anda akan menyusun layout dengan tampilan seperti berikut?

Tentunya akan sangat merepotkan apabila kita mengatur tinggi dari masing-masing kotak, bukan? Belum lagi jika harus mengembangkan aplikasi di ukuran yang lebih besar seperti perangkat tablet.

### Expanded

Flutter memiliki widget Expanded yang dapat mengembangkan child dari Row atau Column sesuai dengan ruang yang tersedia. Cara menggunakannya Anda cukup membungkus masing-masing child ke dalam Expanded.

1. class Rainbow extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Column(
5. children: <Widget>[
6. Expanded(
7. child: Container(
8. color: Colors.red,
9. ),
10. ),
11. Expanded(
12. child: Container(
13. color: Colors.orange,
14. ),
15. ),
16. Expanded(
17. child: Container(
18. color: Colors.yellow,
19. ),
20. ),
21. Expanded(
22. child: Container(
23. color: Colors.green,
24. ),
25. ),
26. Expanded(
27. child: Container(
28. color: Colors.blue,
29. ),
30. ),
31. Expanded(
32. child: Container(
33. color: Colors.indigo,
34. ),
35. ),
36. Expanded(
37. child: Container(
38. color: Colors.purple,
39. ),
40. ),
41. ],
42. );
43. }
44. }

Saat aplikasi dijalankan, masing-masing container akan menempati ruang kosong yang ada. Jika Anda menjalankan di ukuran layar yang berbeda, maka ukuran container juga akan menyesuaikan.

Bisa kita lihat seluruh container menempati ruang dengan ukuran yang sama. Ini disebabkan Expanded memiliki parameter flex yang memiliki nilai default 1. Anda dapat mengubah nilai flex ini sesuai perbandingan yang diinginkan. Misalnya Anda memberikan nilai flex 2 pada salah satu container.

1. Expanded(
2. flex: 2,
3. child: Container(
4. color: Colors.blue,
5. ),
6. ),

Maka container berwarna biru ini akan menjadi lebih besar dengan perbandingan 2/(1 + 1 + 1 + 1 + 2 + 1 + 1) atau 2/8 dari halaman.

### Flexible

Sama seperti Expanded, widget Flexible digunakan untuk mengatur ukuran widget di dalam Row atau Column secara fleksibel. Perbedaan Flexible dan Expanded adalah widget Flexible memungkinkan child widget-nya berukuran lebih kecil dibandingkan ukuran ruang yang tersisa. Sementara, child widget dari Expanded harus menempati ruang yang tersisa dari Column atau Row.

Berikut ini adalah contoh perbedaan antara Expanded dan Flexible:

Kode untuk tampilan seperti di atas adalah seperti berikut:

1. class ExpandedFlexiblePage extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. body: SafeArea(
6. child: Column(
7. children: [
8. Row(
9. children: [
10. ExpandedWidget(),
11. FlexibleWidget(),
12. ],
13. ),
14. Row(
15. children: [
16. ExpandedWidget(),
17. ExpandedWidget(),
18. ],
19. ),
20. Row(
21. children: [
22. FlexibleWidget(),
23. FlexibleWidget(),
24. ],
25. ),
26. Row(
27. children: [
28. FlexibleWidget(),
29. ExpandedWidget(),
30. ],
31. ),
32. ],
33. ),
34. ),
35. );
36. }
37. }
39. class ExpandedWidget extends StatelessWidget {
40. @override
41. Widget build(BuildContext context) {
42. return Expanded(
43. child: Container(
44. decoration: BoxDecoration(
45. color: Colors.teal,
46. border: Border.all(color: Colors.white),
47. ),
48. child: Padding(
49. padding: const EdgeInsets.all(16.0),
50. child: Text(
51. 'Expanded',
52. style: TextStyle(
53. color: Colors.white,
54. fontSize: 24,
55. ),
56. ),
57. ),
58. ),
59. );
60. }
61. }
63. class FlexibleWidget extends StatelessWidget {
64. @override
65. Widget build(BuildContext context) {
66. return Flexible(
67. child: Container(
68. decoration: BoxDecoration(
69. color: Colors.tealAccent,
70. border: Border.all(color: Colors.white),
71. ),
72. child: Padding(
73. padding: const EdgeInsets.all(16.0),
74. child: Text(
75. 'Flexible',
76. style: TextStyle(
77. color: Colors.teal,
78. fontSize: 24,
79. ),
80. ),
81. ),
82. ),
83. );
84. }
85. }

Dokumentasi berikut ini dapat Anda pelajari untuk memaksimalkan penggunaan widget Expanded dan Flexible:

* [Expanded Class](https://api.flutter.dev/flutter/widgets/Expanded-class.html)
* [Flexible Class](https://api.flutter.dev/flutter/widgets/Flexible-class.html)

## 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.

Dalam pemrograman Android kita mengenal Intent lalu pada pemrograman website terdapat tag untuk berpindah dari satu page ke page lain. Pada Flutter kita akan menggunakan sebuah class bernama **Navigator**. Dengan Navigator ini kita akan berpindah dari satu screen ke screen lainnya. Berikut ini contohnya:

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:

1. class SecondScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(
6. title: Text('Second Screen'),
7. ),
8. body: Center(
9. child: OutlinedButton(
10. child: Text('Kembali'),
11. onPressed: () {},
12. ),
13. ),
14. );
15. }
16. }

Lalu, kode untuk halaman pertama akan seperti berikut:

1. class FirstScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(
6. title: Text('First Screen'),
7. ),
8. body: Center(
9. child: ElevatedButton(
10. child: Text('Pindah Screen'),
11. onPressed: () {},
12. ),
13. ),
14. );
15. }
16. }

### Navigator.push

Untuk berpindah ke screen kedua kita akan menggunakan sebuah method Navigator.push, method tersebut ditulis seperti berikut:

1. Navigator.push(context, MaterialPageRoute(builder: (context) {
2. return WidgetScreen();
3. }));

Pada kode di atas 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:

1. class FirstScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(
6. title: Text('First Screen'),
7. ),
8. body: Center(
9. child: ElevatedButton(
10. child: Text('Pindah Screen'),
11. onPressed: () {
12. Navigator.push(context, MaterialPageRoute(builder: (context) {
13. return SecondScreen();
14. }));
15. },
16. ),
17. ),
18. );
19. }
20. }

### Navigator.pop

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

1. Navigator.pop(context)

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:

1. class SecondScreen extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(
6. title: Text('Second Screen'),
7. ),
8. body: Center(
9. child: OutlinedButton(
10. child: Text('Kembali'),
11. onPressed: () {
12. Navigator.pop(context);
13. },
14. ),
15. ),
16. );
17. }
18. }

### 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.

1. 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:

1. class FirstScreen extends StatelessWidget {
2. final String message = 'Hello from First Screen!';
4. @override
5. Widget build(BuildContext context) {
6. return Scaffold(
7. appBar: AppBar(
8. title: Text('First Screen'),
9. ),
10. body: Center(
11. child: ElevatedButton(
12. child: Text('Pindah Screen'),
13. onPressed: () {
14. **Navigator.push(context,**
15. **MaterialPageRoute(builder: (context) => SecondScreen(message)));**
16. },
17. ),
18. ),
19. );
20. }
21. }

Agar Second Screen bisa menerima data tersebut, maka kita perlu mengubah default constructor-nya dan menambahkan variabel untuk menampung datanya.

1. class SecondScreen extends StatelessWidget {
2. final String message;
4. SecondScreen(this.message);
5. }

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

1. class SecondScreen extends StatelessWidget {
2. final String message;
4. SecondScreen(this.message);
6. @override
7. Widget build(BuildContext context) {
8. return Scaffold(
9. appBar: AppBar(
10. title: Text('Second Screen'),
11. ),
12. body: Center(
13. child: Column(
14. mainAxisAlignment: MainAxisAlignment.center,
15. children: [
16. Text(message),
17. OutlinedButton(
18. child: Text('Kembali'),
19. onPressed: () {
20. Navigator.pop(context);
21. },
22. ),
23. ],
24. ),
25. ),
26. );
27. }
28. }

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

Anda dapat memahami Navigation secara mendalam dengan membaca dokumentasi [Navigation Cookbook](https://flutter.dev/docs/cookbook/navigation).

## Responsive Layout

Seperti yang kita tahu, Flutter merupakan framework untuk mengembangkan aplikasi pada berbagai platform. Pada platform mobile sendiri tersedia banyak ukuran layar dari ukuran jam hingga tablet. Ditambah Flutter baru saja mengumumkan dukungan untuk platform web dan desktop. Itu artinya, satu hal yang penting untuk kita pahami adalah bagaimana menerapkan layout yang mampu beradaptasi dengan berbagai ukuran layar yang berbeda.

Pada materi ini kita akan mulai membahas bagaimana mengimplementasikan layout yang responsif.

### MediaQuery

Pendekatan pertama yang akan kita lakukan adalah menggunakan Media Query. Jika Anda sudah familier dengan pengembangan web, mungkin Anda sudah tidak asing dengan konsep ini. MediaQuery adalah kelas yang dapat kita gunakan untuk mendapatkan ukuran dan juga orientasi layar.

Mari kita lihat contoh penerapan MediaQuery.

1. class HomePage extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. **Size screenSize = MediaQuery.of(context).size;**
5. **Orientation orientation = MediaQuery.of(context).orientation;**
7. return Scaffold(
8. backgroundColor: Colors.blueGrey,
9. body: Column(
10. mainAxisAlignment: MainAxisAlignment.center,
11. crossAxisAlignment: CrossAxisAlignment.stretch,
12. children: [
13. Text(
14. 'Screen width: ${screenSize.width.toStringAsFixed(2)}',
15. style: TextStyle(color: Colors.white, fontSize: 18),
16. textAlign: TextAlign.center,
17. ),
18. Text(
19. 'Orientation: $orientation',
20. style: TextStyle(color: Colors.white, fontSize: 18),
21. textAlign: TextAlign.center,
22. ),
23. ],
24. ),
25. );
26. }
27. }

Sekarang jalankan aplikasi untuk melihat ukuran layarnya.

### LayoutBuilder

Cara lain yang bisa kita gunakan adalah dengan widget LayoutBuilder. Perbedaan umum antara MediaQuery dan Layout Builder adalah MediaQuery akan mengembalikan ukuran layar yang digunakan, sedangkan LayoutBuilder mengembalikan ukuran maksimum dari widget tertentu.

Berikut ini adalah contoh kode yang menunjukkan perbedaan antara MediaQuery dan LayoutBuilder:

1. class HomePage extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. Size screenSize = MediaQuery.of(context).size;
6. return Scaffold(
7. backgroundColor: Colors.blueGrey,
8. body: Row(
9. children: [
10. Expanded(
11. child: LayoutBuilder(
12. builder: (BuildContext context, BoxConstraints constraints) {
13. return Column(
14. mainAxisAlignment: MainAxisAlignment.center,
15. crossAxisAlignment: CrossAxisAlignment.stretch,
16. children: [
17. Text(
18. 'MediaQuery: ${screenSize.width.toStringAsFixed(2)}',
19. style: TextStyle(color: Colors.white, fontSize: 18),
20. textAlign: TextAlign.center,
21. ),
22. Text(
23. 'LayoutBuilder: ${constraints.maxWidth}',
24. style: TextStyle(color: Colors.white, fontSize: 18),
25. textAlign: TextAlign.center,
26. ),
27. ],
28. );
29. },
30. ),
31. ),
32. Expanded(
33. flex: 3,
34. child: LayoutBuilder(
35. builder: (BuildContext context, BoxConstraints constraints) {
36. return Container(
37. color: Colors.white,
38. child: Column(
39. mainAxisAlignment: MainAxisAlignment.center,
40. crossAxisAlignment: CrossAxisAlignment.stretch,
41. children: [
42. Text(
43. 'MediaQuery: ${screenSize.width.toStringAsFixed(2)}',
44. style: TextStyle(color: Colors.blueGrey, fontSize: 18),
45. textAlign: TextAlign.center,
46. ),
47. Text(
48. 'LayoutBuilder: ${constraints.maxWidth}',
49. style: TextStyle(color: Colors.blueGrey, fontSize: 18),
50. textAlign: TextAlign.center,
51. ),
52. ],
53. ),
54. );
55. },
56. ),
57. ),
58. ],
59. ),
60. );
61. }
62. }

Hasil ketika dijalankan pada browser akan seperti ini:

Ubahlah ukuran jendela browser untuk melihat perubahan ukuran layar atau media yang digunakan.

Dengan mendapatkan ukuran lebar dan tinggi layar seperti di atas, kita bisa menentukan tampilan konten berdasarkan ukuran layar yang digunakan. Dalam responsive design, terdapat breakpoint yang merupakan “titik” di mana layout akan beradaptasi untuk memberikan pengalaman pengguna sebaik mungkin.

Dengan kode di bawah ini berarti akan terdapat tiga model tampilan berdasarkan ukuran layar:

1. class ResponsivePage extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return Scaffold(
5. appBar: AppBar(),
6. body: LayoutBuilder(
7. builder: (BuildContext context, BoxConstraints constraints) {
8. if (constraints.maxWidth < 600) {
9. return ListView(
10. children: \_generateContainers(),
11. );
12. } else if (constraints.maxWidth < 900) {
13. return GridView.count(
14. crossAxisCount: 2,
15. children: \_generateContainers(),
16. );
17. } else {
18. return GridView.count(
19. crossAxisCount: 6,
20. children: \_generateContainers(),
21. );
22. }
23. },
24. ),
25. );
26. }
28. List<Widget> \_generateContainers() {
29. return List<Widget>.generate(20, (index) {
30. return Container(
31. margin: const EdgeInsets.all(8),
32. color: Colors.blueGrey,
33. height: 200,
34. );
35. });
36. }
37. }

Berikut adalah tampilan dari kode di atas ketika dijalankan:

