### **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.

* class Rainbow extends StatelessWidget {
* const Rainbow({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Column(
* children: <Widget>[
* Expanded(
* child: Container(
* color: Colors.red,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.orange,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.yellow,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.green,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.blue,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.indigo,
* ),
* ),
* Expanded(
* child: Container(
* color: Colors.purple,
* ),
* ),
* ],
* );
* }
* }

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.

* Expanded(
* flex: 2,
* child: Container(
* color: Colors.blue,
* ),
* ),

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:

* class ExpandedFlexiblePage extends StatelessWidget {
* const ExpandedFlexiblePage({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* body: SafeArea(
* child: Column(
* children: [
* Row(
* children: const [
* ExpandedWidget(),
* FlexibleWidget(),
* ],
* ),
* Row(
* children: const [
* ExpandedWidget(),
* ExpandedWidget(),
* ],
* ),
* Row(
* children: const [
* FlexibleWidget(),
* FlexibleWidget(),
* ],
* ),
* Row(
* children: const [
* FlexibleWidget(),
* ExpandedWidget(),
* ],
* ),
* ],
* ),
* ),
* );
* }
* }
* class ExpandedWidget extends StatelessWidget {
* const ExpandedWidget({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Expanded(
* child: Container(
* decoration: BoxDecoration(
* color: Colors.teal,
* border: Border.all(color: Colors.white),
* ),
* child: Padding(
* padding: const EdgeInsets.all(16.0),
* child: Text(
* 'Expanded',
* style: TextStyle(
* color: Colors.white,
* fontSize: 24,
* ),
* ),
* ),
* ),
* );
* }
* }
* class FlexibleWidget extends StatelessWidget {
* const FlexibleWidget({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Flexible(
* child: Container(
* decoration: BoxDecoration(
* color: Colors.tealAccent,
* border: Border.all(color: Colors.white),
* ),
* child: Padding(
* padding: const EdgeInsets.all(16.0),
* child: Text(
* 'Flexible',
* style: TextStyle(
* color: Colors.teal,
* fontSize: 24,
* ),
* ),
* ),
* ),
* );
* }
* }

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:

* class SecondScreen extends StatelessWidget {
* const SecondScreen({Key? key}) : super(key: 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({Key? key}) : super(key: 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**

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

* Navigator.push(context, MaterialPageRoute(builder: (context) {
* return WidgetScreen();
* }));

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:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: 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();
* }));
* },
* ),
* ),
* );
* }
* }

### 

### 

### 

### 

### **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.

* 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:

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

### 

### 

### 

### **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 {
* 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)));**
* },
* ),
* ),
* );
* }
* }

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, {Key? key}) : super(key: key);
* }

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

* class SecondScreen extends StatelessWidget {
* final String message;
* const SecondScreen(this.message, {Key? key}) : super(key: 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:

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.

* class HomePage extends StatelessWidget {
* const HomePage({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* **Size screenSize = MediaQuery.of(context).size;**
* **Orientation orientation = MediaQuery.of(context).orientation;**
* return Scaffold(
* backgroundColor: Colors.blueGrey,
* body: Column(
* mainAxisAlignment: MainAxisAlignment.center,
* crossAxisAlignment: CrossAxisAlignment.stretch,
* children: [
* Text(
* 'Screen width: ${screenSize.width.toStringAsFixed(2)}',
* style: const TextStyle(color: Colors.white, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* Text(
* 'Orientation: $orientation',
* style: const TextStyle(color: Colors.white, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* ],
* ),
* );
* }
* }

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:

* class HomePage extends StatelessWidget {
* const HomePage({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* Size screenSize = MediaQuery.of(context).size;
* return Scaffold(
* backgroundColor: Colors.blueGrey,
* body: Row(
* children: [
* Expanded(
* child: LayoutBuilder(
* builder: (BuildContext context, BoxConstraints constraints) {
* return Column(
* mainAxisAlignment: MainAxisAlignment.center,
* crossAxisAlignment: CrossAxisAlignment.stretch,
* children: [
* Text(
* 'MediaQuery: ${screenSize.width.toStringAsFixed(2)}',
* style: const TextStyle(color: Colors.white, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* Text(
* 'LayoutBuilder: ${constraints.maxWidth}',
* style: const TextStyle(color: Colors.white, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* ],
* );
* },
* ),
* ),
* Expanded(
* flex: 3,
* child: LayoutBuilder(
* builder: (BuildContext context, BoxConstraints constraints) {
* return Container(
* color: Colors.white,
* child: Column(
* mainAxisAlignment: MainAxisAlignment.center,
* crossAxisAlignment: CrossAxisAlignment.stretch,
* children: [
* Text(
* 'MediaQuery: ${screenSize.width.toStringAsFixed(2)}',
* style: const TextStyle(color: Colors.blueGrey, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* Text(
* 'LayoutBuilder: ${constraints.maxWidth}',
* style: const TextStyle(color: Colors.blueGrey, fontSize: 18),
* textAlign: TextAlign.center,
* ),
* ],
* ),
* );
* },
* ),
* ),
* ],
* ),
* );
* }
* }

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:

* class ResponsivePage extends StatelessWidget {
* const ResponsivePage({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(),
* body: LayoutBuilder(
* builder: (BuildContext context, BoxConstraints constraints) {
* if (constraints.maxWidth < 600) {
* return ListView(
* children: \_generateContainers(),
* );
* } else if (constraints.maxWidth < 900) {
* return GridView.count(
* crossAxisCount: 2,
* children: \_generateContainers(),
* );
* } else {
* return GridView.count(
* crossAxisCount: 6,
* children: \_generateContainers(),
* );
* }
* },
* ),
* );
* }
* List<Widget> \_generateContainers() {
* return List<Widget>.generate(20, (index) {
* return Container(
* margin: const EdgeInsets.all(8),
* color: Colors.blueGrey,
* height: 200,
* );
* });
* }
* }