BAB III

APLIKASI E-COMMERCE - SHOPPING CART

A. Tujuan

1. Peserta didik dapat memahami konsep *Stateless Widget* dan *Stateful Widget*.
2. Peserta didik dapat membuat halaman *shopping cart* dari aplikasi eCommerce

B. Perlengkapan

1. Modul Aplikasi eCommerce - Shopping Cart
2. Komputer atau laptop dengan sistem operasi Windows, MacOS, maupun Linux yang sudah terpasang Flutter dan Android Studio.

C. Materi

a. Pengenalan *Stateless Widget* dan *Stateful Widget*

Ketika membuat sebuah kelas *widget*, Anda akan membuat kelas tersebut sebagai subkelas dari `StatelessWidget` atau `StatefulWidget`.

Stateless Widget adalah *widget* yang tidak memiliki *state*. Dengan kata lain, properti dari sebuah *stateless widget*, seperti warna, *widget* anak, dan sebagainya, tidak akan berubah sejak *widget* tersebut dibuat sampai dihancurkan.

Sementara itu, *stateful widget* dapat memiliki *state*, atau properti dari *widget* tersebut dapat berubah.

Lihat kembali aplikasi *Hello World* yang telah Anda buat sebelumnya. Anda dapat melihat bahwa *widget* yang Anda buat adalah `StatelessWidget`. Jika Anda ingin membuat aplikasi menjadi *stateful* (seperti dapat mengubah isi atau warna teks), Anda dapat mencoba mengubah *widget* `HelloWidget` menjadi `StatefulWidget` dengan merefaktor kode Anda sebagai berikut.

```dart
class HelloWidget extends StatefulWidget {
  HelloWidget({Key? key}) : super(key: key);

  @override
  State<HelloWidget> createState() => _HelloWidgetState();
}

class _HelloWidgetState extends State<HelloWidget> {
  @override
  Widget build(BuildContext context) {
    // fungsi build() pada HelloWidget sebelumnya
  }
}
```

Berbeda dengan `StatelessWidget` yang dapat memanggil fungsi `build()` secara langsung, `StatefulWidget` menyimpan sebuah objek `State<Widget>` yang merepresentasikan *state* dari widget tersebut. Anda perlu mendefinisikan *state* tersebut dengan membuat kelas yang merupakan subkelas dari `State<Widget>`.

Untuk membedakan sebuah *state* dari *state* lain, digunakan *private member field* dari kelas `State<Widget>`. Misalnya, jika Anda ingin membuat warna teks 'World' dapat diubah, Anda dapat menambahkan *field* `_worldColor` pada kelas `_HelloWidgetState`.

```dart
class _HelloWidgetState extends State<HelloWidget> {
  Color _worldColor = Colors.blue.shade700; // nilai awal

  @override
  Widget build(BuildContext context) {
    // fungsi build() pada HelloWidget sebelumnya
  }
}
```

Field `_worldColor` ini akan digunakan oleh *widget* `Text` bertuliskan 'World'. Jika nilai `_worldColor` diubah, maka warna dari *widget* `Text` ini akan berubah sesuai dengan nilai `_worldColor`. Ganti nilai properti `color` pada `Text` 'World' menjadi `_worldColor`.

```dart
@override
Widget build(BuildContext context) {
  return Container(
    color: Colors.white70,
    constraints: BoxConstraints.expand(),
    child: Column(
      children: [
        // ...
        Text(
          'World!',
          style: TextStyle(
            color: _worldColor, // Gunakan nilai state _worldColor
            fontWeight: FontWeight.bold,
            fontSize: 60.0,
          ),
          textDirection: TextDirection.ltr,
        ),
      ],
      mainAxisAlignment: MainAxisAlignment.center,
    ),
  );
}
```

Anda juga membutuhkan cara untuk mengubah *state* dan nilai `_worldColor` dari UI aplikasi. Anda dapat menggunakan tombol untuk mencapai hal ini. Terdapat beberapa kelas *widget* yang menyediakan *widget* tombol, salah satunya adalah `ElevatedButton`. *widget* ini memiliki properti antara lain

1. `style`: Menentukan gaya dari tombol, antara lain warna *background* atau konten dari tombol.
2. `onPressed`: Menentukan fungsi yang akan dipanggil ketika tombol ditekan.
3. `child`: Menentukan konten dari tombol, seperti teks atau *icon*.

Pada contoh ini, akan dibuat tiga buah tombol yang disusun secara horizontal dalam sebuah *widget* `Row` dan memiliki warna berbeda. ketika ditekan, warna dari teks ‘World’ akan berubah sesuai dengan warna dari tombol.

Ketika salah satu tombol ditekan, fungsi pada properti `onPressed` akan mengubah nilai `_worldColor`. Agar *state* dari *widget* berubah saat nilai dari *field* `_worldColor` diubah, *statement* yang mengubah nilai `_worldColor` dibungkus dalam pemanggilan fungsi setState() sebagai berikut:

```
setState(() { //statement; });
```

`Row` yang menyimpan tombol-tombol tersebut diletakkan di dalam `Column` di bawah *widget-widget* lainnya, seperti terlihat pada kode berikut. nilai properti child

pada `ElevatedButton` dibiarkan kosong sehingga tombol hanya memperlihatkan warna.

```
@override
Widget build(BuildContext context) {
  return Container(
    color: Colors.white70,
    constraints: BoxConstraints.expand(),
    child: Column(
      children: [
        // SvgPicture, Text 'Hello', Text 'World'
        Row(
          children: [
            ElevatedButton(
              style: ElevatedButton.styleFrom(primary: Colors.blue.shade700),
              onPressed: () {
                setState(() { _worldColor = Colors.blue.shade700; });
              },
              child: null,
            ),
            ElevatedButton(
              // ... Tombol warna hijau
            ),
            ElevatedButton(
              // ... Tombol warna merah
            ),
          ],
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        ),
      ],
      mainAxisAlignment: MainAxisAlignment.center,
    ),
  );
}
```

Jalankan aplikasi, kemudian tekan salah satu tombol. Anda akan melihat bahwa warna dari teks ‘World’ akan berubah sesuai dengan warna tombol.

b. Membangun Aplikasi eCommerce - *ShoppingCart*

Pada bagian ini Anda akan memulai membuat aplikasi *eCommerce* berbekal dengan konsep-konsep yang telah dijelaskan sebelumnya. Salah satu fitur yang umum pada aplikasi *eCommerce* adalah *ShoppingCart*

Pertama-tama buatlah projek Flutter baru dan kemudian buat kode *boilerplate* pada `lib/main.dart` sebagai berikut.

File: lib/main.dart

File: lib/main.dart

```dart
import 'package:flutter/material.dart';
import 'ui/shopping_cart/index.dart';

// Widget utama
class StoreApp extends StatelessWidget {
  const StoreApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'InThe Store', // Judul bebas
      theme: ThemeData(
        primarySwatch: Colors.teal,
      ),
      home: const ShoppingCartPage(),
    );
  }
}

void main() {
  // Jalankan StoreApp
  runApp(const StoreApp());
}
```

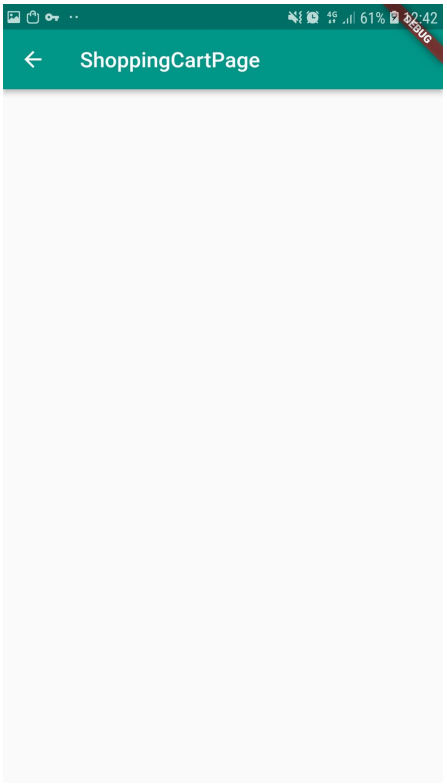
Untuk mempermudah organisasi kode, Anda dapat membagi kode ke dalam beberapa *file*. Misalnya, *file* `lib/main.dart` memuat kode untuk menjalankan aplikasi, kemudian, komponen-komponen *user interface* disimpan di dalam sebuah folder `lib/ui`. UI untuk *shopping cart* disimpan dalam folder `lib/ui/shopping_cart` dan widget halaman *shopping cart* disimpan dalam *file* `lib/ui/shopping_cart/index.dart`.

file: lib/ui/shopping_cart/index.dart

File: **lib/ui/shopping_cart/index.dart**

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

// Halaman Shopping Cart
class ShoppingCartPage extends StatelessWidget {
  const ShoppingCartPage({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: const Icon(Icons.arrow_back),
          tooltip: 'Back',
          onPressed: () {},
        ),
        title: const Text('Shopping Cart')
      ),
      body: null,
    );
  }
}
```

Ketika dijalankan, aplikasi akan menuju halaman *shopping cart*.



b.1. Membuat Daftar item *Shopping Cart* dengan ListView

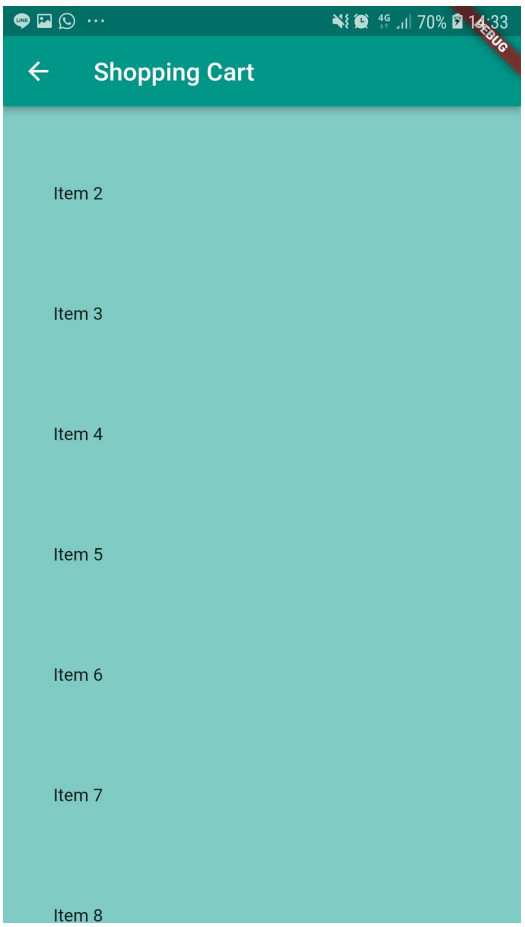
`ListView` menampilkan daftar item atau *widget*. Hal ini mirip dengan *widget* `Column`, tetapi konten `ListView` dapat di-*scroll*. Oleh karena itu, `ListView` cocok digunakan untuk menampilkan daftar dengan jumlah item yang banyak atau belum diketahui.

Pada *file* `lib/ui/shopping_cart/index.dart`, ubah body dari `Scaffold` pada kelas `ShoppingCartPage` menjadi `ListView`, dengan children berupa *list* dari `Container` yang memuat nama item. Buat item dari `ListView` sebanyak mungkin.

file: lib/ui/shopping_cart/index.dart

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar( ... ),
    body: ListView(
      children: [
        Container(
          color: Colors.teal.shade200,
          padding: const EdgeInsets.all(40.0),
          child: const Text('Item 1')
        ),
        Container(
          color: Colors.teal.shade200,
          padding: const EdgeInsets.all(40.0),
          child: const Text('Item 2')
        ),
        // ... dan seterusnya
      ]
    ),
  );
}
```

Jalankan aplikasi. Jika jumlah item yang anda tambahkan cukup banyak Anda dapat men-*scroll* layar untuk melihat item-item selanjutnya.



Tentunya menambahkan *widget* untuk setiap item pada `ListView` terasa sangat boros, karena banyak komponen yang ditulis berulang-ulang. Untuk mempermudah dan mempersingkat kode, `ListView` menyediakan konstruktor `Listview.builder`. `ListView.builder` menerima beberapa argumen. Argumen `itemCount` menentukan banyaknya item yang akan dibuat dalam `ListView`. Argumen `itemBuilder` merupakan sebuah fungsi yang menerima indeks untuk membangun *widget* item sesuai dengan indeks item pada *list*.

Dibandingkan dengan menulis *list* yang berisi *widget* item, Anda dapat menulis *list* items yang berisi nama item bertipe `String`. Kemudian Anda dapat mengganti `ListView` yang Anda buat sebelumnya menjadi `ListView.builder` dengan fungsi `itemBuilder` mengindeks *list* items untuk mendapatkan nama *item*. Implementasi fungsi `build()` pada `ShoppingCartPage` akan menjadi sebagai berikut.

file: lib/ui/shopping_cart/index.dart

```dart
@override
Widget build(BuildContext context) {
  final List<String> items = [
    'Item 1', 'Item 2', 'Item 3', 'Item 4', 'Item 5',
    'Item 6', 'Item 7', 'Item 8', 'Item 9', 'Item 10',
  ];

  return Scaffold(
    appBar: AppBar( ... ),
    body: ListView.builder(
      itemCount: items.length,
      itemBuilder: (BuildContext context, int index) {
        return Container(
          color: Colors.teal.shade200,
          padding: const EdgeInsets.all(40.0),
          child: Text(items[index]),
        );
      },
    ),
  );
}
```

Jika Anda menjalankan aplikasi, aplikasi akan menampilkan daftar yang sama dengan yang sebelumnya.

b.2. Membuat *widget* item *Shopping Cart* dengan ListTile

Flutter menyediakan *widget* `ListTile` yang dapat digunakan untuk mengisi `ListView`. `ListTile` memuat sebuah teks dengan tambahan ikon di sebelah kiri dan kanan. Komponen dari `ListTile` adalah:

1. `title`: Memuat judul atau konten utama dari *item*
2. `subtitle`: Memuat konten tambahan atau deskripsi dari *item*
3. `leading`: *Widget* yang ditampilkan di sebelah kiri konten. Dapat berupa gambar, *icon*, atau *avatar*.
4. `trailing`: *Widget* yang ditampilkan di sebelah kanan konten. Biasanya berupa tombol untuk melakukan aksi pada *item*.

Pada aplikasi ini, Anda dapat menggunakan `ListTile` untuk menampilkan informasi dari item pada *Shopping Cart*. properti `title` memuat nama *item*, `subtitle` memuat harga per satuan *item,* `leading` memuat gambar item, dan trailing memuat aksi yang dapat dilakukan pada *item*. Pada fungsi `build()`, buat

list `prices` dan `pictures` untuk menampung harga dan gambar dari *item* dengan indeks yang bersesuaian.

file: lib/ui/shopping_cart/index.dart

```
@override
Widget build(BuildContext context) {
  final List<String> items = [
    'Item 1', 'Item 2', 'Item 3', 'Item 4', 'Item 5',
    'Item 6', 'Item 7', 'Item 8', 'Item 9', 'Item 10',
  ];

  final List<int> prices = [
    10000, 25000, 100000, 42000, 77000,
    78000, 11000, 6000, 149000, 200000,
  ];

  final List<String> pictures = [
    'assets/shoe1.png', 'assets/shoe2.jpg', 'assets/shoe3.jpg',
    'assets/shoe1.png', 'assets/shoe2.jpg', 'assets/shoe3.jpg',
    'assets/shoe1.png', 'assets/shoe2.jpg', 'assets/shoe3.jpg',
    'assets/shoe1.png', 'assets/shoe2.jpg',
  ];
```

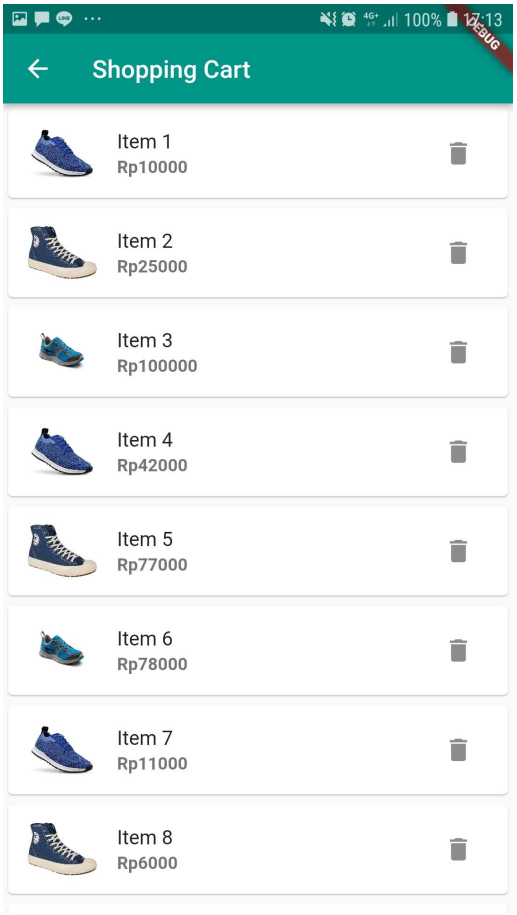
Ganti fungsi `itemBuilder` pada `ListView.builder` sehingga mengembalikan `ListTile`.

file: lib/ui/shopping_cart/index.dart

```
      body: ListView.builder(
        itemCount: items.length,
        itemBuilder: (BuildContext context, int index) {
          return Card(
            child: ListTile(
              title: Text(items[index]),
              leading: Image.asset(
                pictures[index],
                height: 56.0,
                width: 56.0,
                fit: BoxFit.cover,
              ),
              subtitle: Text(
                'Rp${prices[index]}',
                style: const TextStyle(fontWeight: FontWeight.bold),
              ),
              trailing: IconButton(
                icon: const Icon(Icons.delete),
                onPressed: () {},
              ),
            ),
          );
        },
      ),
```

Perlu diperhatikan bahwa jika Anda menggunakkan aset gambar pada `ListTile`, Anda perlu menambahkan folder aset tersebut disimpan pada file `pubspec.yaml`.

Aplikasi akan terlihat sebagai berikut.



b.3 Membuat struktur data Product

Pada bagian sebelumnya, Anda membagi data *item* pada *shopping cart* ke dalam tiga buah *list* (items, prices, images). Hal ini kurang terstruktur. Anda dapat menggabungkan ketiga properti dari *item* tersebut ke dalam sebuah struktur data Product.

Buat folder lib/models dan buat *file* lib/models/product.dart. Pada *file* tersebut, definisikan sebuah kelas Product yang memiliki *field* id, name, description, price, dan image.

file: lib/models/product.dart

```dart
class Product {
  Product({
    required this.id,
    required this.name,
    this.description = "",
    required this.price,
    this.image = "",
  });

  String id;
  String name;
  String description;
  int price;
  String image;
}
```

Beberapa hal yang perlu diperhatikan:

1. Sebuah konstruktor kelas dapat menerima argumen yang sesuai dengan nama *field*. Nilai argumen `this.foo` pada konstruktor akan disimpan pada *field* `foo` pada kelas.
2. Sebuah argumen dapat diberikan kata kunci `required`. Dengan begitu nilai argumen harus didefinisikan dalam memanggil konstruktor.
3. Argumen yang tidak diberikan kata kunci `required` bersifat opsional. Jika nilai argumen tidak diberikan saat memanggil konstruktor, argumen tersebut akan menggunakan nilai *default* (Pada contoh ini, argumen description memiliki nilai *default* “”).

Untuk menggunakan struktur data ini, Anda perlu mengimpor *file* `lib/models/product.dart`. Ada dua cara untuk melakukan impor.

1. Menggunakan *relative path* dari *file*. Misalnya, jika Anda ingin mengimpor `lib/models/product.dart` ke `lib/ui/shopping_cart/index.dart`, anda menuliskan import `“../../models/product.dart”;`
2. Menggunakan *absolute path* dengan awalan `package:<nama_projek>` yang merujuk pada folder `lib` pada projek Anda. Sehingga Anda menuliskan `import “package:<name_projek>/models/product.dart”;`

Anda dapat memanggil konstruktor sebagai berikut.

`Product(id: “123”, name: “Product 1”, decription: “a product”, price: 14000, image: “assets/shoe1.png”)`

Kemudian, anda dapat mengganti definisi *item shopping cart* dengan *list* kelas Product.

file: lib/ui/shopping_cart/index.dart

```dart
final List<Product> items = [
  Product(id: '1', name: 'Item 1', price: 10000, image: 'assets/shoe1.png'),
  Product(id: '2', name: 'Item 2', price: 10000, image: 'assets/shoe2.jpg'),
  Product(id: '3', name: 'Item 3', price: 10000, image: 'assets/shoe3.jpg'),
  Product(id: '4', name: 'Item 4', price: 10000, image: 'assets/shoe1.png'),
  Product(id: '5', name: 'Item 5', price: 10000, image: 'assets/shoe2.jpg'),
  Product(id: '6', name: 'Item 6', price: 10000, image: 'assets/shoe3.jpg'),
  Product(id: '7', name: 'Item 7', price: 10000, image: 'assets/shoe1.png'),
  Product(id: '8', name: 'Item 8', price: 10000, image: 'assets/shoe2.jpg'),
  Product(id: '9', name: 'Item 9', price: 10000, image: 'assets/shoe3.jpg'),
  Product(id: '10', name: 'Item 10', price: 10000, image: 'assets/shoe1.png'),
  Product(id: '11', name: 'Item 11', price: 10000, image: 'assets/shoe2.jpg'),
  Product(id: '12', name: 'Item 12', price: 10000, image: 'assets/shoe3.jpg'),
];
```

```dart
body: ListView.builder(
  itemCount: items.length,
  itemBuilder: (BuildContext context, int index) {
    return Card(
      child: ListTile(
        title: Text(items[index].name),
        leading: Image.asset(
          items[index].image,
          height: 56.0,
          width: 56.0,
          fit: BoxFit.cover,
        ),
        subtitle: Text(
          'Rp${items[index].price}',
          style: const TextStyle(fontWeight: FontWeight.bold),
        ),
        trailing: IconButton(
          icon: const Icon(Icons.delete),
          onPressed: () {},
        ),
      ),
    );
  },
),
```

b.4 Membuat *widget* pengaturan jumlah item *shopping cart*

Jika Anda perhatikan, *item* yang ada di daftar *shopping* cart berbeda satu sama yang lain. Bagaimana jika Anda ingin membeli lebih dari satu *item* dengan jenis yang sama? Pada aplikasi *eCommerce* pada umumnya, untuk setiap *item* pada *shopping cart*, terdapat *widget* yang menampilkan jumlah yang akan dibeli. Jumlah ini dapat diatur dengan menekan tombol tambah dan kurang.

Anda dapat menambahkan *widget* ini pada aplikasi Anda. Karena *widget* ini memiliki *state* yang berubah (jumlah *item*), maka *widget* ini diturunkan dari StatefulWidget.

Buatlah sebuah *stateful widget* ShoppingCartItemQty setelah definisi ShoppingCartPage pada file lib/ui/shopping_cart/index.dart. *Widget* ini menyimpan *state* berupa *private field* _qty. Struktur *widget* cukup sederhana, berupa sebuah Row yang terdiri dari IconButton dengan *icon* tanda kurang, Text yang menampilkan nilai _qty, dan IconButton dengan *icon* tanda tambah. Selain itu terdapat juga IconButton dengan *icon* hapus. Jika tombol tambah atau kurang ditekan, properti onPressed pada kedua IconButton akan memanggil setState yang mengubah nilai _qty.

file: lib/ui/shopping_cart/index.dart

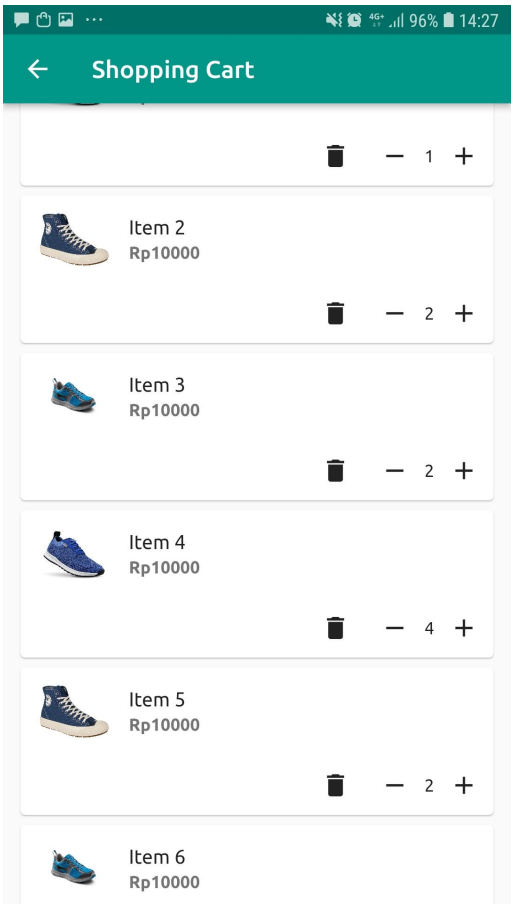
```dart
class ShoppingCartItemQty extends StatefulWidget {
  const ShoppingCartItemQty({
    Key? key,
  }) : super(key: key);

  @override
  State<ShoppingCartItemQty> createState() => _ShoppingCartItemQtyState();
}

class _ShoppingCartItemQtyState extends State<ShoppingCartItemQty> {
  int _qty = 1;

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.end,
      children: [
        IconButton(
          icon: const Icon(Icons.delete),
          onPressed: () {},
        ),
        // SizedBox(width: 12),
        IconButton(
          icon: const Icon(Icons.remove),
          onPressed: () {
            setState(() { if (_qty > 1) _qty--; });
          },
        ),
        Text('$_qty'),
        IconButton(
          icon: const Icon(Icons.add),
          onPressed: () {
            setState(() { _qty++; });
          },
        ),
      ],
    );
  }
}
```

Setelah itu, Anda dapat menyusun ListTile dari *item Shopping Cart* yang dibangun oleh ListView.builder dengan *widget* ShoppingCartItemQty secara vertikal sebagai anak dari sebuah *widget* Column.

file: lib/ui/shopping_cart/index.dart

```
body: ListView.builder(
  padding: const EdgeInsets.all(10),
  itemCount: items.length,
  itemBuilder: (BuildContext context, int index) {
    return Card(
      child: Column(
        children: [
          ListTile(
            // ListTile yang sama dengan sebelumnya.
          ),
          const ShoppingCartItemQty(),
        ]
      ),
    );
  },
),
```

Jalankan aplikasi dan tekan tombol tambah dan kurang untuk melihat perubahannya.



D. Referensi

Flutter. 2022. *Flutter Documentation*. [Online] Internet: https://docs.flutter.dev. Diakses 27 Juli 2022.