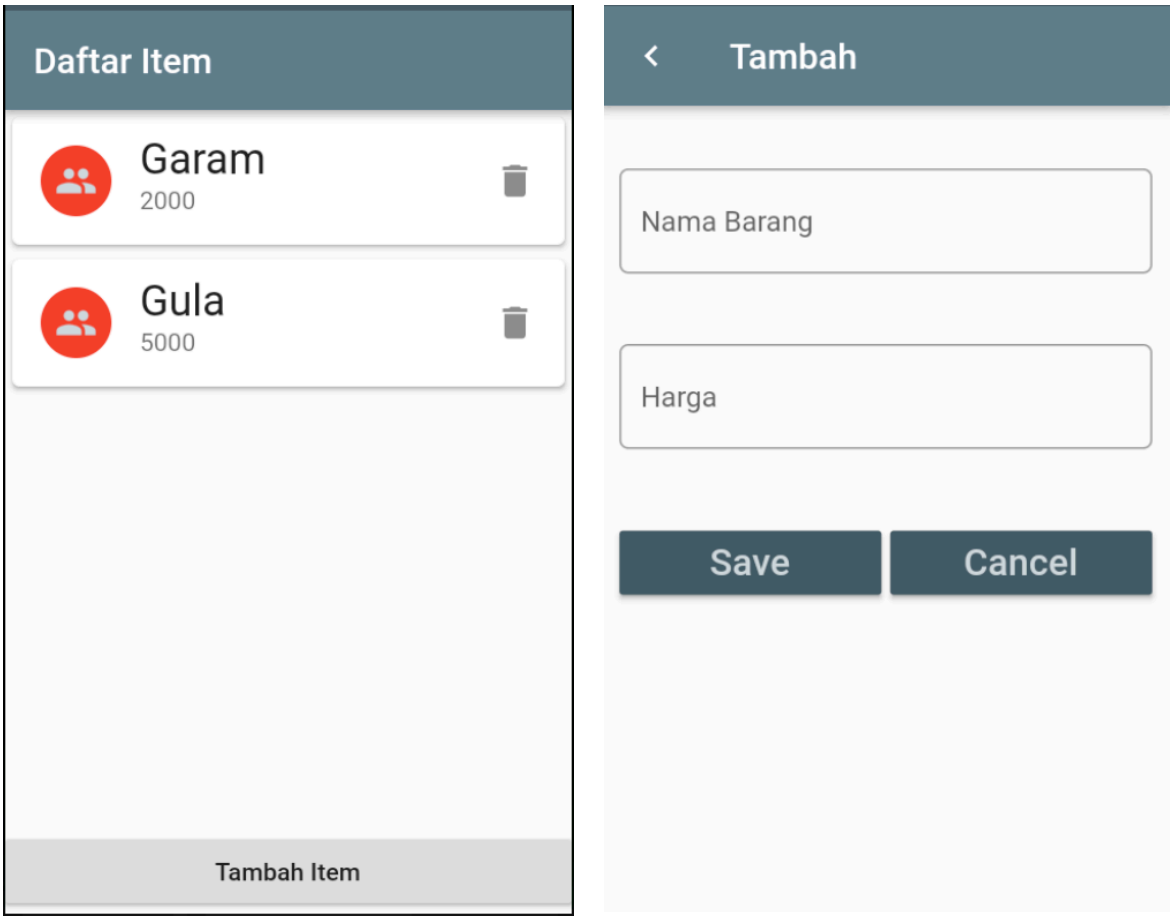
BAB 6. DATABASE SQLITE

6.1 Desain Aplikasi

Pada praktikum bab 6 anda akan belajar cara untuk melakukan CRUD (create, read, update dan delete) pada SQLite. Untuk pembuatan aplikasi ini dimulai dengan membuat sebuah desain yang dapat dimulai dengan membuat wireframe atau desain utuh di perangkat lunak seperti adobe xd atau sketch dan yang lain. Untuk menyederhanakan proses desain anda sudah disediakan template sederhana di starter code dengan menggunakan template material design. Berikut ini desain mockup aplikasi yang dibuat.



6.2 Teori

6.2.1 Operasi pada SQLite

Berikut adalah daftar SQL statement yang dapat anda lakukan pada SQLite berdasarkan sumber https://www.sqlitetutorial.net/ :

1. Simple Query, contohnya Select
2. Mengurutkan (sorting), contohnya Order By

3. Filtering data, contohnya Select Distinct, Where, Between, In, Like, Glob, IS NULL
4. Join tables, contohnya SQLite Join, Inner Join, Left Join, Cross Join, Self Join, Full Outer Join
5. Group Data, contohnya Group By, Having
6. Set Operator, contohnya Union, Except dan Intersect
7. Subquery
8. Case statement
9. Mengubah data, contohnya Update, Delete, Insert dan Replace
10. Transaction Statement
11. Constraints
12. View
13. Indexed
14. Triggered
15. Full text search
16. Sqlite Tools, contohnya SQLite Commands, SQLite Show Table, SQLite Dump, SQLite Import CSV dan SQLite Export CSV

Pada praktikum ini kita fokus untuk melakukan simple query dan mengubah data pada SQLite.

6.3 Praktikum

Untuk memanfaatkan SQLite pada Flutter berikut adalah langkah-langkah yang perlu untuk dilakukan:

6.3.1 Mengatur dependencies pada pubspec.yaml

Tambahkan dependency sqlite dan path_provider seperti pada code di bawah ini selanjutnya tekan yang diberi kotal merah untuk download packages yang baru. Untuk nilai **Any** dapat diisi versi terbaru atau sesuai kebutuhan. Versi packages sqflite dapat dicek di https://pub.dev/packages/sqflite dan path_provider dapat dicek di https://pub.dev/packages/path_provider.

6.3.2 Membuat Data Model

Buat class Model Item dengan nama **item.dart** yang dibuat hamper sama dengan praktikum sebelumnya hanya ditambahkan atribut _id. Jadi total ada 3 atribut yaitu _id, _name, dan _price.

```
class Item{
  int _id;
  String _name;
```

```
  int _price;


}
```

Selanjutnya buat getter dan setter untuk masing-masing variabel. Untuk getter dan setter secara otomatis dapat degenerate menggunakan extension pada vscode dengan nama Dart Getter and Setters. Jika menggunakan extension tersebut restart VSCode dan klik kanan pada variabel yang akan degenerate getter dan setternya.

getter akan mengambil nilai yang dimasukkan ke consturctor dan setter ini akan dipakai untuk mengembalikan nilai yang dimasukkan dari constructor, untuk setiap variable.

```
class Item{
  int _id;
  String _name;
  int _price;



 int get id => _id;


 String get name => this._name;
 set name(String value) => this._name = value;


 get price => this._price;
 set price( value) => this._price = value;


 // konstruktor versi 1
  Item(this._name, this._price);
}
```

Kita akan membuat beberapa contructor pada class Item. Pertama buat contructor untuk mengeset nilai name dan price secara bersama-sama.

```
// konstruktor versi 1
  Item(this._name, this._price);
```

Contructor kedua adalah berbentuk map digunakan untuk mengambil data dari sql yang tersimpan berbentuk Map setelah itu akan disimpan kembali dalam bentuk variabel

```
// konstruktor versi 2: konversi dari Map ke Item
  Item.fromMap(Map<String, dynamic> map) {
    this._id = map['id'];
    this._name = map['name'];
    this._price = map['price'];
  }
```

Terakhir membuat method Map untuk melakukan update dan insert.

```
// konversi dari Item ke Map
  Map<String, dynamic> toMap() {
    Map<String, dynamic> map = Map<String, dynamic>();
    map['id'] = this._id;
    map['name'] = name;
    map['price'] = price;
    return map;
  }


}
```

Code lengkap dari Item.Dart adalah sebagai berikut:

```
class Item{
  int _id;
  String _name;
  int _price;



 int get id => _id;


 String get name => this._name;
 set name(String value) => this._name = value;


 get price => this._price;
 set price( value) => this._price = value;


 // konstruktor versi 1
  Item(this._name, this._price);
```

```
// konstruktor versi 2: konversi dari Map ke Item
  Item.fromMap(Map<String, dynamic> map) {
    this._id = map['id'];
    this._name = map['name'];
    this._price = map['price'];
  }


   // konversi dari Item ke Map
  Map<String, dynamic> toMap() {
    Map<String, dynamic> map = Map<String, dynamic>();
    map['id'] = this._id;
    map['name'] = name;
    map['price'] = price;
    return map;
  }

}
```

6.3.3 Membuat Db Helper

Langkah praktikum kedua adalah membuat dbhelper.dart. Pertama adalah membuat fungsi untuk menginisialisasi database.

Future adalah “tipe data” yang terpanggil dengan adanya delay atau “keterlambatan”. Tidak seperti method lainnya, sistem akan terus menjalankan method tersebut sampai method itu selesai berjalan. Contohnya ketika kita akan mengambil data yang ada di dalam database/API , kita membutuhkan method Future untuk mengambil data di dalam database/API tersebut. Untuk lebih jelasnya kalian bisa membuka sumber dibawah:

1. https://medium.com/flutter-community/a-guide-to-using-futures-in-flutter-for-beginners-ebeddfbfb967
2. https://www.youtube.com/watch?v=g9Uk1Xou0m4

Didalam flutter ada async dan await.

- async : menggunakan future pada sebuah method, sehingga membuat sistem menunggu sampai terjadi Blocking. Makanya, method tersebut harus ditandai dengan async.
- await : Jika ada method yang ditandai await, maka artinya sistem harus menunggu sampai syntax tersebut selesai berjalan.

Pada source code, variable directory akan menunggu sampai method getApplicationDocumentsDirectory mengerjakan tugasnya. Method getApplicationDocumentsDirectory() berfungsi untuk mengambil direktori folder aplikasi untuk menempatkan data yang dibuat pengguna sehingga tidak dapat dibuat ulang oleh aplikasi tersebut. Setelah itu kita gunakan variable String path, untuk membuat nama database kita dengan mengambil lokasi directory nya dan menambahkannya dengan nama database item.db.

Setelah itu kita baru membuat database dan akses untuk membuka database-nya dengan openDatabase. Dalam method ini akan membutuhkan nama database-nya, dengan variable path yang kita buat seblumnya. Kemudian ada parameter version dan onCreate.

- version: bisa dikatakan version adalah level untuk penggunaan databasenya. Karena kita bahkan belum memiliki tablenya, kita cukup menulisnya '1'.
- onCreate: bukan cuma onCreate, kita bisa menambahkanya dengan onConfigure, onUpgrade, dll. Tapi yang kita bahas hanya onCreate saja. Seperti namanya, fungsinya untuk membuat table supaya kita dapat mengaksesnya.

```
Future<Database> initDb() async {


  //untuk menentukan nama database dan lokasi yg dibuat
    Directory directory = await getApplicationDocumentsDirectory();
    String path = directory.path + 'item.db';


   //create, read databases
    var itemDatabase = openDatabase(path, version: 1, onCreate: _createDb);


    //mengembalikan nilai object sebagai hasil dari fungsinya
    return itemDatabase;
  }
```

Selanjutnya membuat method untuk mengcreate database seperti berikut.

```
  //buat tabel baru dengan nama item
  void _createDb(Database db, int version) async {
    await db.execute('''
      CREATE TABLE item (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        price INTEGER
      )
    ''');
  }
```

Selanjutnya membuat fungsi untuk melakukan CRUD (create, read, update dan delete). Method pada insert, update, dan delete struktur dan logikanya sama. Hanya syntax SQL nya yang berbeda.

Pada source code variable Database db akan membuka akses ke database. Variable count digunakan untuk menampung hasil SQL — nya. Bertipe Integer karena ketika sistem berhasil dieksekusi, nilai yang dikeluarkan adalah 1.

```
//create databases
  Future<int> insert(Item object) async {
    Database db = await this.database;
    int count = await db.insert('item', object.toMap());
    return count;
  }
//update databases
  Future<int> update(Item object) async {
    Database db = await this.database;
    int count = await db.update('item', object.toMap(),
                                where: 'id=?',
                                whereArgs: [object.id]);
    return count;
  }

//delete databases
  Future<int> delete(int id) async {
    Database db = await this.database;
    int count = await db.delete('item',
                                where: 'id=?',
                                whereArgs: [id]);
    return count;
  }

  Future<List<Item>> getItemList() async {
    var itemMapList = await select();
    int count = itemMapList.length;
    List<Item> itemList = List<Item>();
    for (int i=0; i<count; i++) {
      itemList.add(Item.fromMap(itemMapList[i]));
    }
    return itemList;
  }
```

Untuk source code lengkapnya adalah sebagai berikut.

```
 import 'package:sqflite/sqflite.dart';
import 'dart:async';
import 'dart:io';
import 'package:path_provider/path_provider.dart';

import 'item.dart';

class DbHelper {
  static DbHelper _dbHelper;
  static Database _database;
  DbHelper._createObject();

  Future<Database> initDb() async {

  //untuk menentukan nama database dan lokasi yg dibuat
    Directory directory = await getApplicationDocumentsDirectory();
    String path = directory.path + 'item.db';

   //create, read databases
    var itemDatabase = openDatabase(path, version: 4, onCreate: _createDb);

    //mengembalikan nilai object sebagai hasil dari fungsinya
    return itemDatabase;
  }
  //buat tabel baru dengan nama item
  void _createDb(Database db, int version) async {
    await db.execute('''
      CREATE TABLE item (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        price INTEGER
      )
    ''');
  }

//select databases
  Future<List<Map<String, dynamic>>> select() async {
    Database db = await this.initDb();
```

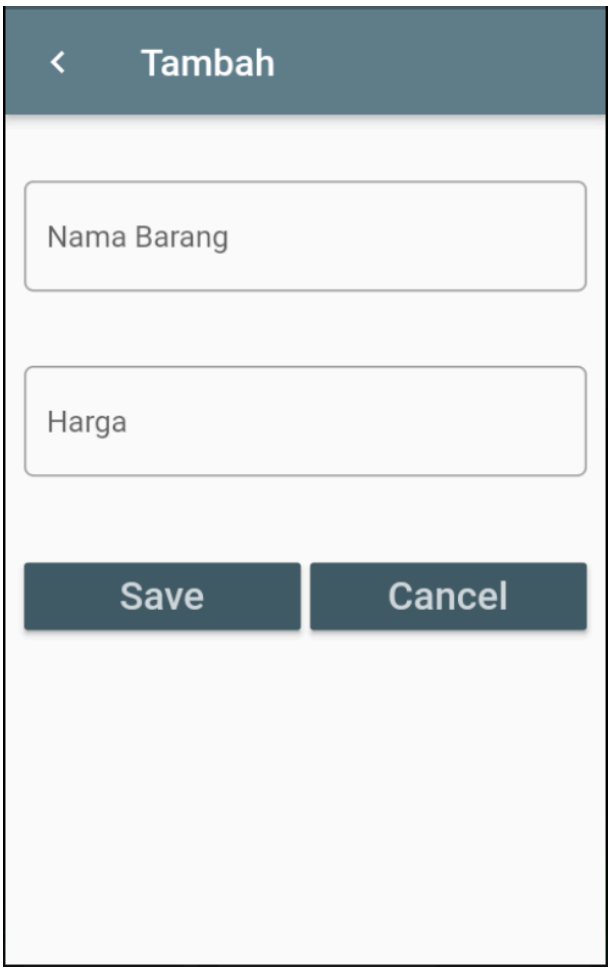
```
    var mapList = await db.query('item', orderBy: 'name');
    return mapList;
  }

//create databases
  Future<int> insert(Item object) async {
    Database db = await this.initDb();
    int count = await db.insert('item', object.toMap());
    return count;
  }
//update databases
  Future<int> update(Item object) async {
    Database db = await this.initDb();
    int count = await db.update('item', object.toMap(),
                                where: 'id=?',
                                whereArgs: [object.id]);
    return count;
  }

//delete databases
  Future<int> delete(int id) async {
    Database db = await this.initDb();
    int count = await db.delete('item',
                                where: 'id=?',
                                whereArgs: [id]);
    return count;
  }


  Future<List<Item>> getItemList() async {
    var itemMapList = await select();
    int count = itemMapList.length;
    List<Item> itemList = List<Item>();
    for (int i=0; i<count; i++) {
      itemList.add(Item.fromMap(itemMapList[i]));
    }
    return itemList;
  }


  factory DbHelper() {
    if (_dbHelper == null) {
      _dbHelper = DbHelper._createObject();
    }
    return _dbHelper;
```

```
  }
  Future<Database> get database async {
    if (_database == null) {
      _database = await initDb();
    }
    return _database;
  }
}
```

Pada tahap ini sebenarnya kita sudah selesai untuk membuat database SQLite pada flutter yang kurang hanya tampilannya saja. Untuk memanfaatkan fungsi diatas dapat dipanggil pada fungsi test di flutter.

6.3.4 Membuat Entry Form

Buatlah UI untuk menginput dan mengedit isi dari table yang kita buat seperti berikut. Tampilan tidak harus sama persis dengan screenshoot di bawah ini yang paling penting adalah variabel TextEditingControllernya gunakan nama nameController dan priceControlle sehingga mudah dalam mengikuti jobsheet ini.




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

class EntryForm extends StatefulWidget {
  final Item item;

  EntryForm(this.item);

  @override
  EntryFormState createState() => EntryFormState(this.item);
}
//class controller
class EntryFormState extends State<EntryForm> {
  Item item;

  EntryFormState(this.item);
```

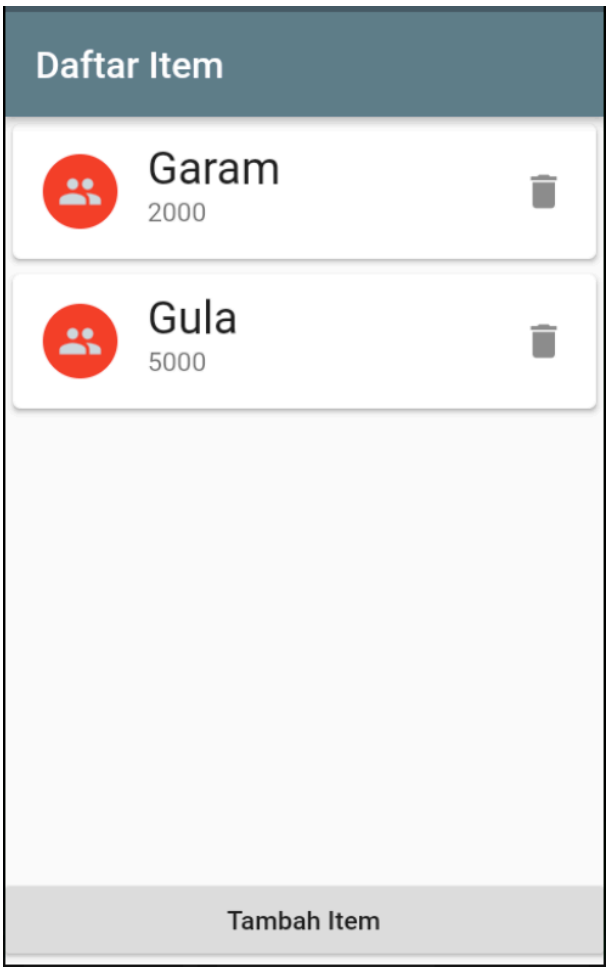
```
TextEditingController nameController = TextEditingController();
TextEditingController priceController = TextEditingController();


@override
Widget build(BuildContext context) {
  //kondisi
  if (item != null) {
    nameController.text = item.name;
    priceController.text = item.price.toString();
  }
  //rubah
  return Scaffold(
    appBar: AppBar(
      title: item == null ? Text('Tambah') : Text('Ubah'),
      leading: Icon(Icons.keyboard_arrow_left),
    ),
    body: Padding(
      padding: EdgeInsets.only(top: 15.0, left:10.0, right:10.0),
      child: ListView(
        children: <Widget> [
          // nama
          Padding (
            padding: EdgeInsets.only(top:20.0, bottom:20.0),
            child: TextField(
              controller: nameController,
              keyboardType: TextInputType.text,
              decoration: InputDecoration(
                labelText: 'Nama Barang',
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(5.0),
                ),
              ),
              onChanged: (value) {
                //
              },
            ),
          ),

          // harga
          Padding (
            padding: EdgeInsets.only(top:20.0, bottom:20.0),
            child: TextField(
```

```
                controller: priceController,
                keyboardType: TextInputType.number,
                decoration: InputDecoration(
                  labelText: 'Harga',
                  border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(5.0),
                  ),
                ),
                onChanged: (value) {
                  //
                },
              ),
            ),

            // tombol button
            Padding (
              padding: EdgeInsets.only(top:20.0, bottom:20.0),
              child: Row(
                children: <Widget> [
                  // tombol simpan
                  Expanded(
                    child: RaisedButton(
                      color: Theme.of(context).primaryColorDark,
                      textColor: Theme.of(context).primaryColorLight,
                      child: Text(
                        'Save',
                        textScaleFactor: 1.5,
                      ),
                      onPressed: () {
                        if (item == null) {
                          // tambah data
                          item            =            Item(nameController.text,
int.parse(priceController.text));
                        } else {
                          // ubah data
                          item.name = nameController.text;
                          item.price = int.parse(priceController.text);
                        }
                        // kembali ke layar sebelumnya dengan membawa objek item
                        Navigator.pop(context, item);
                      },
                    ),
                  ),
```

```
                    Container(width: 5.0,),
                    // tombol batal
                    Expanded(
                      child: RaisedButton(
                        color: Theme.of(context).primaryColorDark,
                        textColor: Theme.of(context).primaryColorLight,
                        child: Text(
                          'Cancel',
                          textScaleFactor: 1.5,
                        ),
                        onPressed: () {
                          Navigator.pop(context);
                        },
                      ),
                    ),
                  ],
                ),
              ),
            ],
          ),
        )
      );
  }
}
```

6.3.5 Membuat Home

Buatlah UI untuk menampilan record yang ada di SQLite pada class home.dart untuk melakukan aksi delete, edit dan menambah data seperti berikut.




```dart
import 'package:flutter/material.dart';
import 'package:sqflite/sqflite.dart';
import 'dart:async';
import 'package:sqlite/dbhelper.dart';
import 'package:sqlite/entryform.dart';

import 'item.dart';
//pendukung program asinkron

class Home extends StatefulWidget {
  @override
  HomeState createState() => HomeState();
}

class HomeState extends State<Home> {

  DbHelper dbHelper = DbHelper();
  int count = 0;
```

```
List<Item> itemList;

@override
Widget build(BuildContext context) {
  if (itemList == null) {
    itemList = List<Item>();
  }

  return Scaffold(
    appBar: AppBar(
      title: Text('Daftar Item'),
    ),
    body: Column(children : [
      Expanded(child: createListView(),),
      Container(
        alignment: Alignment.bottomCenter,
        child: SizedBox(
                width: double.infinity,
                child: RaisedButton(
                  child: Text("Tambah Item"),
                  onPressed: () async {
                      var item = await navigateToEntryForm(context, null);
                      if (item != null) {
                        //TODO 2 Panggil Fungsi untuk Insert ke DB
                        int result = await dbHelper.insert(item);
                        if (result > 0) {
                          updateListView();
                        }
                      }
                    },
                  ),
        ),
      ),
    ]),
  );
}

Future<Item> navigateToEntryForm(BuildContext context, Item item) async {
  var result = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (BuildContext context) {
        return EntryForm(item);
```

```
        }
      )
    );
    return result;
  }

  ListView createListView() {
    TextStyle textStyle = Theme.of(context).textTheme.headline5;
    return ListView.builder(
      itemCount: count,
      itemBuilder: (BuildContext context, int index) {
        return Card(
          color: Colors.white,
          elevation: 2.0,
          child: ListTile(
            leading: CircleAvatar(
              backgroundColor: Colors.red,
              child: Icon(Icons.ad_units),
            ),
            title: Text(this.itemList[index].name, style: textStyle,),
            subtitle: Text(this.itemList[index].price.toString()),
            trailing: GestureDetector(
              child: Icon(Icons.delete),
              onTap: ()async {
                    //TODO 3 Panggil Fungsi untuk Delete dari DB berdasarkan Item
                  },
            ),
            onTap: () async {
              var item = await navigateToEntryForm(context, this.itemList[index]);
              //TODO 4 Panggil Fungsi untuk Edit data
            },
          ),
        );
      },
    );
  }

  //update List item
  void updateListView() {
    final Future<Database> dbFuture = dbHelper.initDb();
    dbFuture.then((database) {
      //TODO 1 Select data dari DB
      Future<List<Item>> itemListFuture = dbHelper.getItemList();
```

```
      itemListFuture.then((itemList) {
        setState(() {
          this.itemList = itemList;
          this.count = itemList.length;
        });
      });
    });
  }


}
```

6.3.6 Membuat Main

```
//kode utama Aplikasi tampilan awal
import 'package:flutter/material.dart';
import 'package:sqlite/home.dart';

//package letak folder Anda
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Tambahkan Item',
      theme: ThemeData(
        primarySwatch: Colors.blueGrey,
      ),
      home: Home(),
    );
  }
}
```

6.3.7 Tugas

1. Lengkapi TODO 3 dan 4.
2. Tambahkan variabel stok dan kode barang untuk operasi CRUD