PRAKTIKUM PEMROGRAMAN PERANGKAT BERGERAK
TUGAS GUIDED & UNGUIDED

MODUL X
DATA STORAGE (BAGIAN I)

Disusun Oleh :

Nadia Putri Rahmaniar / 2211104012

S1 SE-06-01

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

Yudha Islami Sulistya, S.Kom., M.Cs.

PROGRAM STUDI S1 SOFTWARE ENGINEERING FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO 2024

GUIDED

1. Pengenalan SQLite

SQLite adalah database relasional yang dapat digunakan untuk aplikasi mobile untuk penyimpanan data offline, terutama untuk penyimpanan lokal seperti memori cache aplikasi. SQLite mendukung operasi CRUD (membuat, membaca, mengubah, dan menghapus), yang merupakan bagian yang sangat penting dari manajemen data. Struktur database SQLite mirip dengan SQL, termasuk tipe data dan variabel yang digunakan.

2. SQL Helper Dasar

Dalam Flutter, SQL Helper umumnya mengacu pada penggunaan paket seperti sqflite untuk mengelola database SQLite. Sqflite sendiri adalah plugin Flutter yang mendukung pelaksanaan operasi CRUD (Create, Read, Update, Delete) pada database SQLite. Untuk menggunakan Sqlite sebagai SQL, berikut adalah langkah-langkah dasar.

Helper flutter :

a. Menambahkan Plugin sqflite dan path di dalam file pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  sqflite: ^2.4.1
  path: ^1.9.0
```

b. Membuat class DatabseHelper baru yang dapat digunakan untuk mengelola database lalu import package sqflite dan path ke file db_helper.dart.

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

//Kelas databse untuk mengelola database
class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  static Database? _database;

```

c. Membuat factory constructor untuk mengembalikan instance singleton dan private singleton.

```dart
// factory constructor untuk mengembalikan instance singletonce
factory DatabaseHelper() {
  return _instance;
}

// Private constructor
DatabaseHelper._internal();
```

d. Membuat Getter untuk database

```dart
//Getter untuk database
Future<Database> get database async {
  if (_database != null) return _database!;
  {
    _database = await _initDatabase();
    return _database!;
  }
}
```

e. Inisialisasi database dengan nama database sesuai keinginan

```dart
//Inisialisasi database
Future<Database> _initDatabase() async {
  // mendapatkan path untuk database
  String path = join(await getDatabasesPath(), 'my_prakdatabase.db');
  // membuka database
  return await openDatabase(
    path,
    version: 1,
    onCreate: _onCreate,
  );
}
```

f. Membuat tabel untuk database-nya dengan record atau value id, title, dan description.

```dart
//Membuat tabel db dengan record dan value id, title, description
Future<void> _onCreate(Database db, int version) async {
  await db.execute('''
  CREATE TABLE my_table(
  id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  title TEXT,
  description TEXT,
  createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP)
  ''');
}
```

g. Membuat metode untuk memasukkan data ke dalam tabel.

```
61  // metode untuk mengambil semua data dari tabel
62  Future<int> insert(Map<String, dynamic> row) async {
63    Database db = await database;
64    return await db.insert('my_table', row);
65  }
```

h. Metode untuk mengambil semua data dari tabel

```
61  // metode untuk mengambil semua data dari tabel
62  Future<int> insert(Map<String, dynamic> row) async {
63    Database db = await database;
64    return await db.insert('my_table', row);
65  }
```

i. Metode untuk update data dalam tabel.

```
67  //  metode untuk memperbarui data dalam tabel
68  Future<int> update(Map<String, dynamic> row) async {
69    Database db = await database;
70    int id = row['id'];
71    return await db.update('my_table', row, where: 'id = ?', whereArgs: [id]
72  }
```

j. Terakhir metode untuk menghapus data dari tabel

```
74  // metode untuk menghapus data dari tabel
75  Future<int> delete(int id) async {
76    Database db = await database;
77    return await db.delete('my_table', where: 'id = ?', whereArgs: [id]);
78  }
```

k. Read

Dengan sqflite di Flutter, berbagai perintah seperti where, groupBy, orderBy, dan having dapat digunakan untuk membuat query. Kita dapat menggunakan metode query() dalam package sqflite untuk membaca data dari database. Kita juga dapat membaca satu atau lebih data sekaligus. Contoh kode untuk operasi read menggunakan sqflite dibawah ini:

```
80  //Membaca semua data
81  Future<List<Map<String, dynamic>>> queryAllRows() async {
82    Database db = await database;
83    return await db.query('my_table');
84  }
```

3. Source Code Praktikum:

a. main.dart

```dart
import 'package:data_storage/view/my_db_view.dart';
import 'package:flutter/material.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Database Storage',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: MyDatabaseView(),
    );
  }
}
```

b. db_helper.dart

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

//Kelas databse untuk mengelola database
class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  static Database? _database;

// factory constructor untuk mengembalikan instance singletonce
  factory DatabaseHelper() {
    return _instance;
  }

// Private constructor
  DatabaseHelper._internal();

//Getter untuk database
  Future<Database> get database async {
    if (_database != null) return _database!;
    {
      _database = await _initDatabase();
      return _database!;
    }
  }
```

```
25
26    //Inisialisasi database
27    Future<Database> _initDatabase() async {
28      // mendapatkan path untuk database
29      String path = join(await getDatabasesPath(), 'my_prakdatabase.db');
30      // membuka database
31      return await openDatabase(
32        path,
33        version: 1,
34        onCreate: _onCreate,
35      );
36    }
37
38    //Membuat tabel db dengan record dan value id, title, description
39    Future<void> _onCreate(Database db, int version) async {
40      await db.execute('''
41  CREATE TABLE my_table(
42  id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
43  title TEXT,
44  description TEXT,
45  createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP)
46  ''');
47    }
48
49  // metode untuk mengambil semua data dari tabel
50    Future<int> insert(Map<String, dynamic> row) async {
51      Database db = await database;
52      return await db.insert('my_table', row);
53    }
54
55  //  metode untuk memperbarui data dalam tabel
56    Future<int> update(Map<String, dynamic> row) async {
57      Database db = await database;
58      int id = row['id'];
59      return await db.update('my_table', row, where: 'id = ?', whereArgs: [id]);
60    }
61
62    // metode untuk menghapus data dari tabel
63    Future<int> delete(int id) async {
64      Database db = await database;
65      return await db.delete('my_table', where: 'id = ?', whereArgs: [id]);
66    }
67
68    //Membaca semua data
69    Future<List<Map<String, dynamic>>> queryAllRows() async {
70      Database db = await database;
71      return await db.query('my_table');
72    }
73  }
```

c. my_db_view.dart

```dart
import 'package:data_storage/helper/db_helper.dart';
import 'package:flutter/material.dart';


class MyDatabaseView extends StatefulWidget {
  const MyDatabaseView({super.key});


  @override
```

```
              State<MyDatabaseView>       createState()       =>
_MyDatabaseViewState();
}

class _MyDatabaseViewState extends State<MyDatabaseView> {
  final DatabaseHelper dbHelper = DatabaseHelper();
  List<Map<String, dynamic>> _dbData = [];
        final    TextEditingController    _titleController    =
TextEditingController();
     final   TextEditingController   _descriptionController   =
TextEditingController();

  @override
  void initState() {
    _refreshData();
    super.initState();
  }

  @override
  void dispose() {
    _titleController.dispose();
    _descriptionController.dispose();
    super.dispose();
  }

  void _refreshData() async {
    final data = await dbHelper.queryAllRows();
    setState(() {
      _dbData = data;
    });
  }

  void _addData() async {
                   if     (_titleController.text.isEmpty     ||
_descriptionController.text.isEmpty) {
      _showSnackbar('Title and Description cannot be empty!');
```

```
      return;
    }
    await dbHelper.insert({
      'title': _titleController.text,
      'description': _descriptionController.text,
    });
    _titleController.clear();
    _descriptionController.clear();
    _refreshData();
  }

  void _updateData(int id) async {
                  if     (_titleController.text.isEmpty     ||
_descriptionController.text.isEmpty) {
      _showSnackbar('Title and Description cannot be empty!');
      return;
    }
    await dbHelper.update({
      'id': id,
      'title': _titleController.text,
      'description': _descriptionController.text,
    });
    _titleController.clear();
    _descriptionController.clear();
    _refreshData();
  }

  void _deleteData(int id) async {
    await dbHelper.delete(id);
    _refreshData();
  }

  void _showEditDialog(Map<String, dynamic> item) {
    _titleController.text = item['title'];
    _descriptionController.text = item['description'];
```

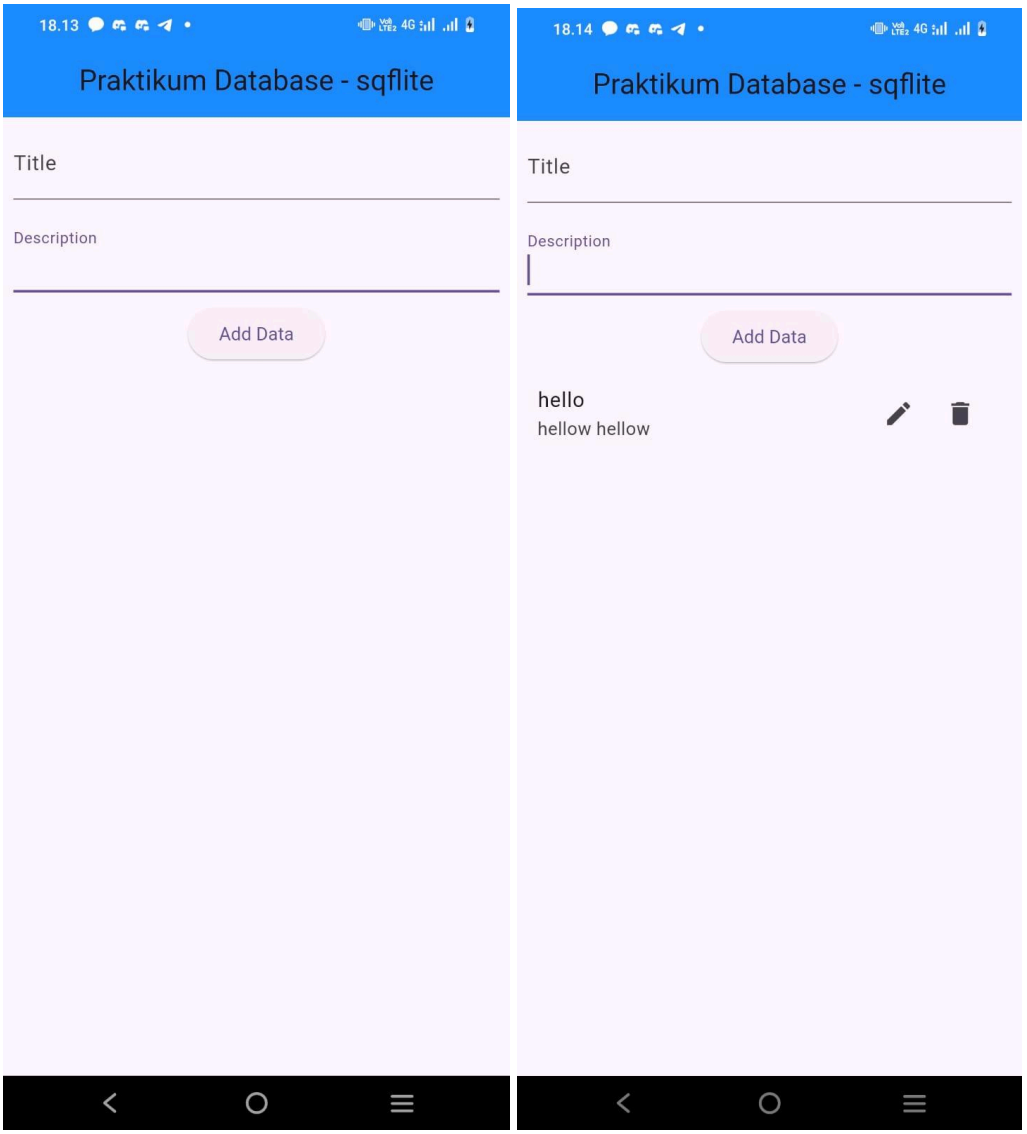
```
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: const Text('Edit Item'),
          content: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              TextField(
                controller: _titleController,
                  decoration: const InputDecoration(labelText:
'Title'),
              ),
              TextField(
                controller: _descriptionController,
                  decoration: const InputDecoration(labelText:
'Description'),
              ),
            ],
          ),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: const Text('Cancel'),
            ),
            TextButton(
              onPressed: () {
                _updateData(item['id']);
                Navigator.of(context).pop();
              },
              child: const Text('Save'),
            ),
          ],
        );
```

```
      },
    );
  }

  void _showSnackbar(String message) {
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text(message)));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Praktikum Database - sqflite'),
        backgroundColor: Colors.blueAccent,
        centerTitle: true,
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              controller: _titleController,
                  decoration: const InputDecoration(labelText:
'Title'),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              controller: _descriptionController,
                  decoration: const InputDecoration(labelText:
'Description'),
            ),
          ),
          ElevatedButton(
```

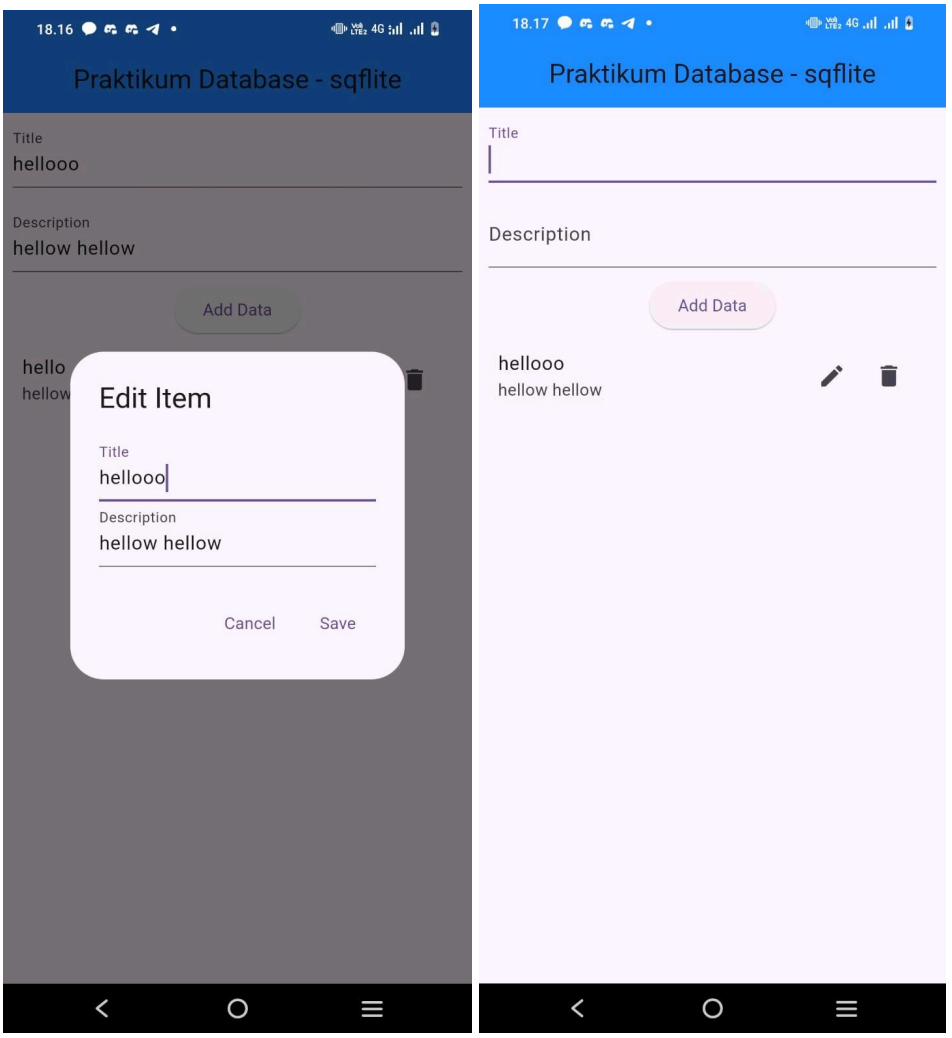
```
            onPressed: _addData,
            child: const Text('Add Data'),
          ),
          Expanded(
            child: ListView.builder(
              itemCount: _dbData.length,
              itemBuilder: (context, index) {
                final item = _dbData[index];
                return ListTile(
                  title: Text(item['title']),
                  subtitle: Text(item['description']),
                  trailing: Row(
                    mainAxisSize: MainAxisSize.min,
                    children: [
                      IconButton(
                        icon: const Icon(Icons.edit),
                        onPressed: () {
                          _showEditDialog(item);
                        },
                      ),
                      IconButton(
                        icon: const Icon(Icons.delete),
                                              onPressed:  ()  =>
_deleteData(item['id']),
                      ),
                    ],
                  ),
                );
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

Output :

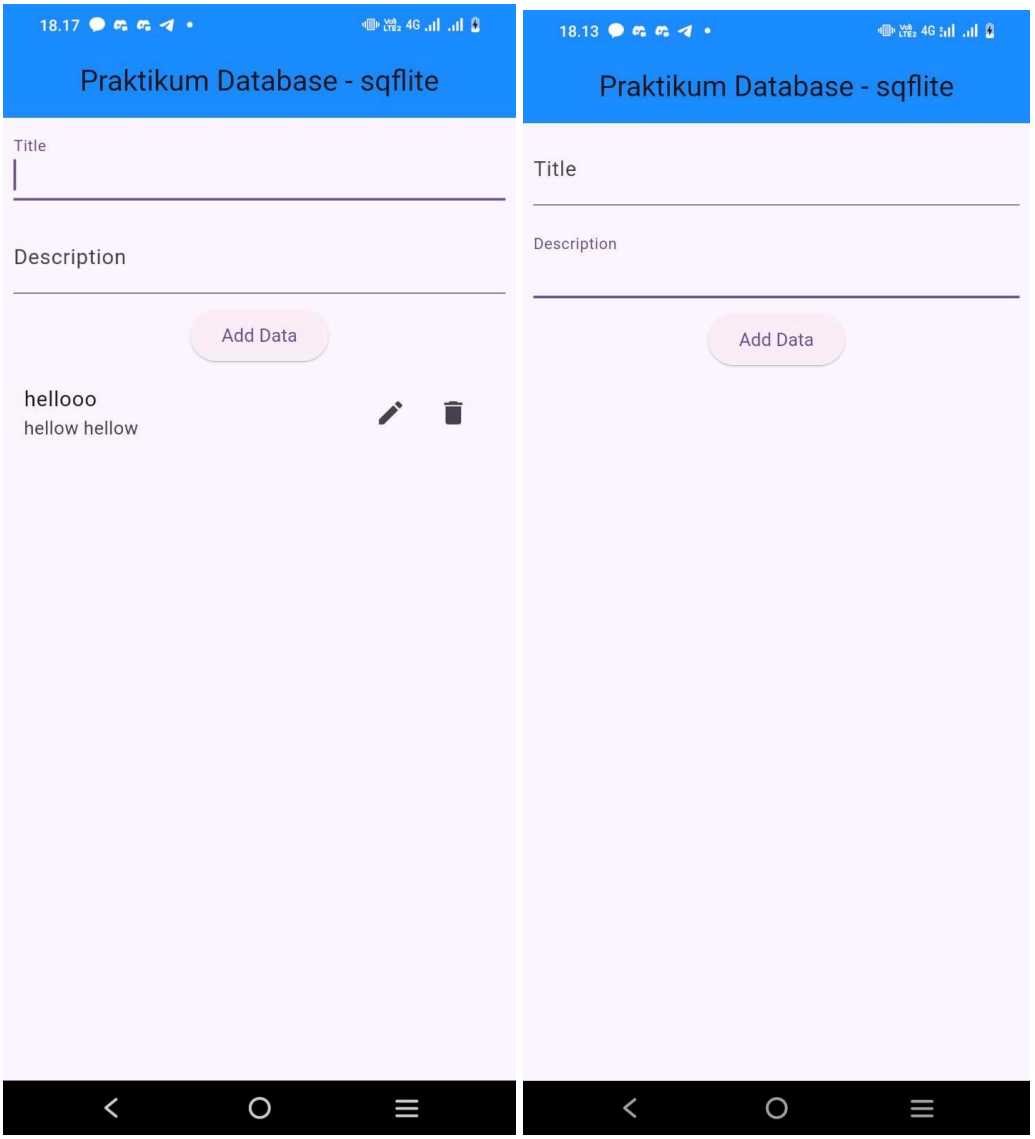
Berikut adalah halaman utama. Kemudian user dapat memasukkan informasi pada "Title" dan "Description" lalu klik button Add Data. Maka data akan ditambahkan seperti tampilan sebelah kanan.



Jika user ingin mengedit dapat klik icon "pensil" dan kemudian mengubah data. Misalnya, Mengedit title "hello" menjadi "helooo". Kemudian, klik "Save" untuk menyimpan perubahan.



Dibawah ini saya telah menambahkan 2 data, jika ingin menghapus data dapat menekan ikon tong sampah, maka data akan dihapus, misalnya. sehingga tampilannya menjadi seperti awal tidak ada data sama sekali:



UNGUIDED

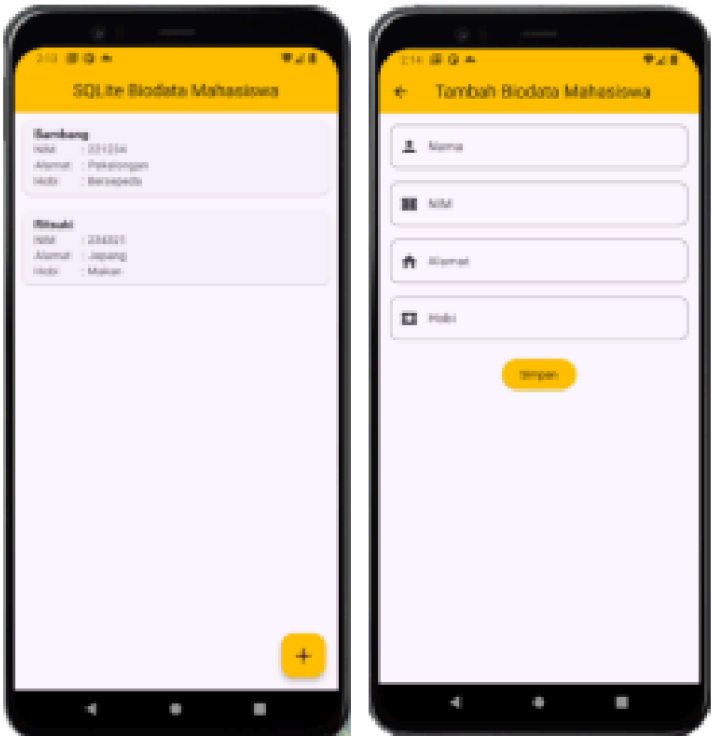
Buatlah sebuah project aplikasi Flutter dengan SQLite untuk menyimpan data biodata mahasiswa yang terdiri dari nama, NIM, domisili, dan hobi. Data yang dimasukkan melalui form akan ditampilkan dalam daftar di halaman utama. Alur Aplikasi:

a. Form Input: Buat form input untuk menambahkan biodata mahasiswa, dengan kolom:
   - Nama
   - Nim
   - Alamat
   - Hobi

b. Tampilkan Daftar Mahasiswa: Setelah data berhasil ditambahkan, tampilkan daftar semua data mahasiswa yang sudah disimpan di halaman utama.

c. Implementasikan fitur Create (untuk menyimpan data mahasiswa) dan Read (untuk menampilkan daftar mahasiswa yang sudah disimpan).

d. Contoh output:



Note: Jangan lupa sertakan source code, screenshoot output, dan deskripsi program. Kreatifitas menjadi nilai tambah

1. **Source Code**

main.dart

```dart
import 'package:flutter/material.dart';
import 'package:data_storage/view/my_db_view.dart';
import 'package:data_storage/view/main_view.dart';
void main() {
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Unguided Data Storage',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: MainView(),
    );
  }
}
```

db_helper.dart

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

//Kelas databse untuk mengelola database
class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  static Database? _database;

// factory constructor untuk mengembalikan instance singletonce
  factory DatabaseHelper() {
    return _instance;
  }

// Private constructor
  DatabaseHelper._internal();

//Getter untuk database
  Future<Database> get database async {
    if (_database != null) return _database!;
    {
```

```dart
      _database = await _initDatabase();
      return _database!;
    }
  }

  //Inisialisasi database
  Future<Database> _initDatabase() async {
    // mendapatkan path untuk database
    String path = join(await getDatabasesPath(), 'my_prakdatabase.db');
    // membuka database
    return await openDatabase(
      path,
      version: 2,
      onCreate: _onCreate,
      // Tambahkan ini untuk menangani upgrade
      onUpgrade: _onUpgrade,
    );
  }

  //Membuat tabel db dengan record dan value id, nama, nim
  Future<void> _onCreate(Database db, int version) async {
    await db.execute('''
    CREATE TABLE my_table(
      id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
      nama TEXT,
      nim TEXT,
      alamat TEXT,
      hobi TEXT,
      createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP)
''');
  }

  Future<void> _onUpgrade(Database db, int oldVersion, int newVersion) async {
    if (oldVersion < 2) {
      // Tambahkan kolom alamat dan hobi jika belum ada
      await db.execute('ALTER TABLE my_table ADD COLUMN alamat TEXT');
      await db.execute('ALTER TABLE my_table ADD COLUMN hobi TEXT');
    }
  }

// metode untuk mengambil semua data dari tabel
  Future<int> insert(Map<String, dynamic> row) async {
    Database db = await database;
    return await db.insert('my_table', row);
  }

//  metode untuk memperbarui data dalam tabel
  Future<int> update(Map<String, dynamic> row) async {
    Database db = await database;
    int id = row['id'];
    return await db.update('my_table', row, where: 'id = ?', whereArgs: [id]);
  }

  // metode untuk menghapus data dari tabel
  Future<int> delete(int id) async {
    Database db = await database;
    return await db.delete('my_table', where: 'id = ?', whereArgs: [id]);
  }

  //Membaca semua data
  Future<List<Map<String, dynamic>>> queryAllRows() async {
    Database db = await database;
    return await db.query('my_table');
  }
}
```

my_db_view.dart

```
import 'package:flutter/material.dart';
import 'package:data_storage/helper/db_helper.dart';

class MyDatabaseView extends StatefulWidget {
   final Map<String, dynamic>? item; // Menambahkan parameter item
untuk edit

  const MyDatabaseView({super.key, this.item});

  @override
  State<MyDatabaseView> createState() => _MyDatabaseViewState();
}

class _MyDatabaseViewState extends State<MyDatabaseView> {
  final DatabaseHelper dbHelper = DatabaseHelper();
          final     TextEditingController     _namaController     =
TextEditingController();
           final     TextEditingController     _nimController     =
TextEditingController();
         final     TextEditingController     _alamatController     =
TextEditingController();
          final     TextEditingController     _hobiController     =
TextEditingController();

  @override
  void initState() {
    super.initState();
    if (widget.item != null) {
          // Jika data ada (edit mode), masukkan data ke dalam
controller
      _namaController.text = widget.item?['nama'] ?? '';
      _nimController.text = widget.item?['nim'] ?? '';
      _alamatController.text = widget.item?['alamat'] ?? '';
      _hobiController.text = widget.item?['hobi'] ?? '';
    }
```

```
}

@override
void dispose() {
  _namaController.dispose();
  _nimController.dispose();
  _alamatController.dispose();
  _hobiController.dispose();
  super.dispose();
}

void _saveData() async {
  if (_namaController.text.isEmpty ||
      _nimController.text.isEmpty ||
      _alamatController.text.isEmpty ||
      _hobiController.text.isEmpty) {
    _showSnackbar('Tidak boleh kosong!');
    return;
  }

  if (widget.item == null) {
    // Jika item null berarti tambah data
    await dbHelper.insert({
      'nama': _namaController.text,
      'nim': _nimController.text,
      'alamat': _alamatController.text,
      'hobi': _hobiController.text,
    });
  } else {
    // Jika item tidak null berarti edit data
    await dbHelper.update({
      'id': widget.item?['id'],
      'nama': _namaController.text,
      'nim': _nimController.text,
      'alamat': _alamatController.text,
      'hobi': _hobiController.text,
```

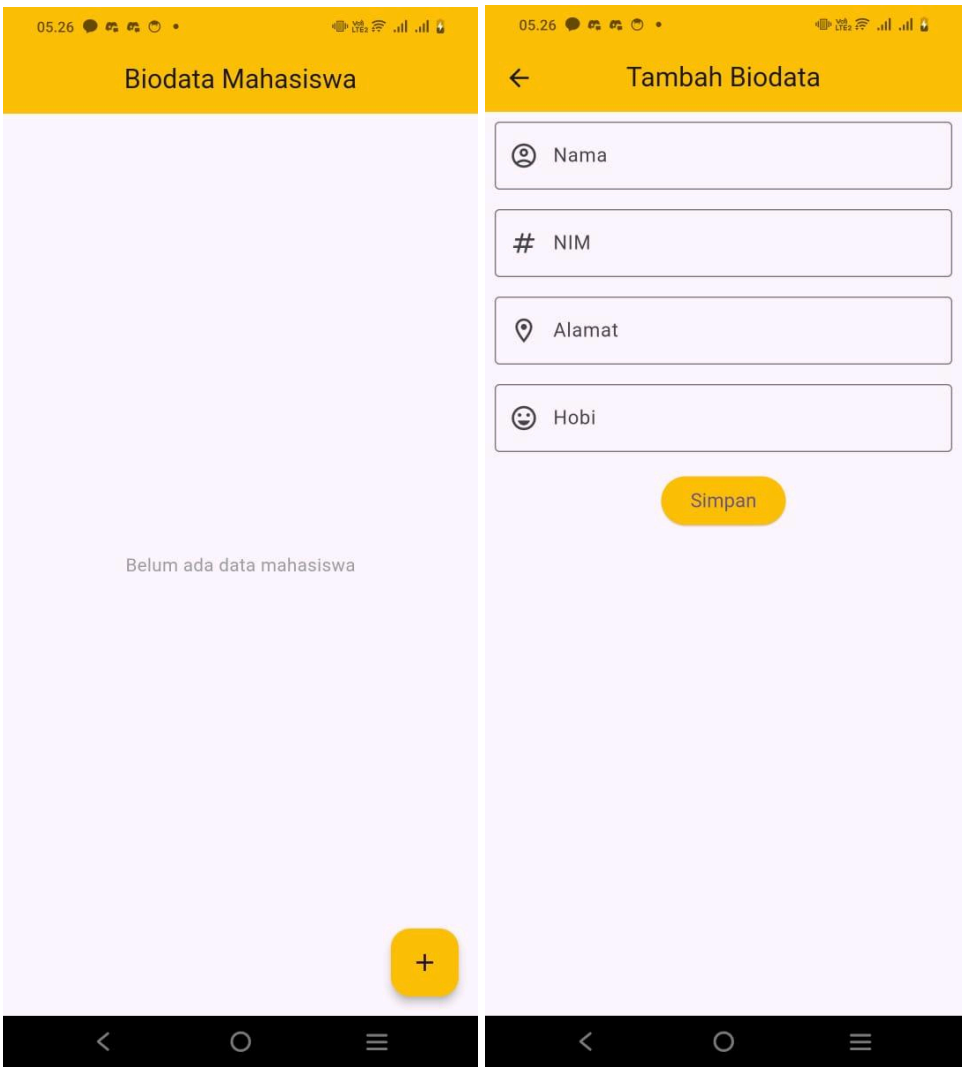
```
    });
  }

  Navigator.pop(context);
}

void _showSnackbar(String message) {
  ScaffoldMessenger.of(context)
      .showSnackBar(SnackBar(content: Text(message)));
}

//Mendefinisikan fungsi buildText
Widget buildTextField({
  required TextEditingController controller,
  required String label,
  required IconData icon,
}) {
  return TextField(
    controller: controller,
    decoration: InputDecoration(
      labelText: label,
      prefixIcon: Icon(icon),
      border: OutlineInputBorder(),
    ),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.item == null
          ? 'Tambah Biodata'
          : 'Edit Biodata'),
      backgroundColor: Colors.amber,
      centerTitle: true,
```

```
    ),
    body: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
        children: [
          buildTextField(
            controller: _namaController,
            label: 'Nama',
            icon: Icons.account_circle_outlined,
          ),
          const SizedBox(height: 16),
          buildTextField(
            controller: _nimController,
            label: 'NIM',
            icon: Icons.numbers,
          ),
          const SizedBox(height: 16),
          buildTextField(
            controller: _alamatController,
            label: 'Alamat',
            icon: Icons.location_on_outlined,
          ),
          const SizedBox(height: 16),
          buildTextField(
            controller: _hobiController,
            label: 'Hobi',
            icon: Icons.emoji_emotions_outlined,
          ), // icon
          const SizedBox(height: 16),
          ElevatedButton(
            onPressed: _saveData,
            child: const Text(
              'Simpan',
              style: TextStyle(fontSize: 16),
            ),
                  style: ElevatedButton.styleFrom(backgroundColor:
```

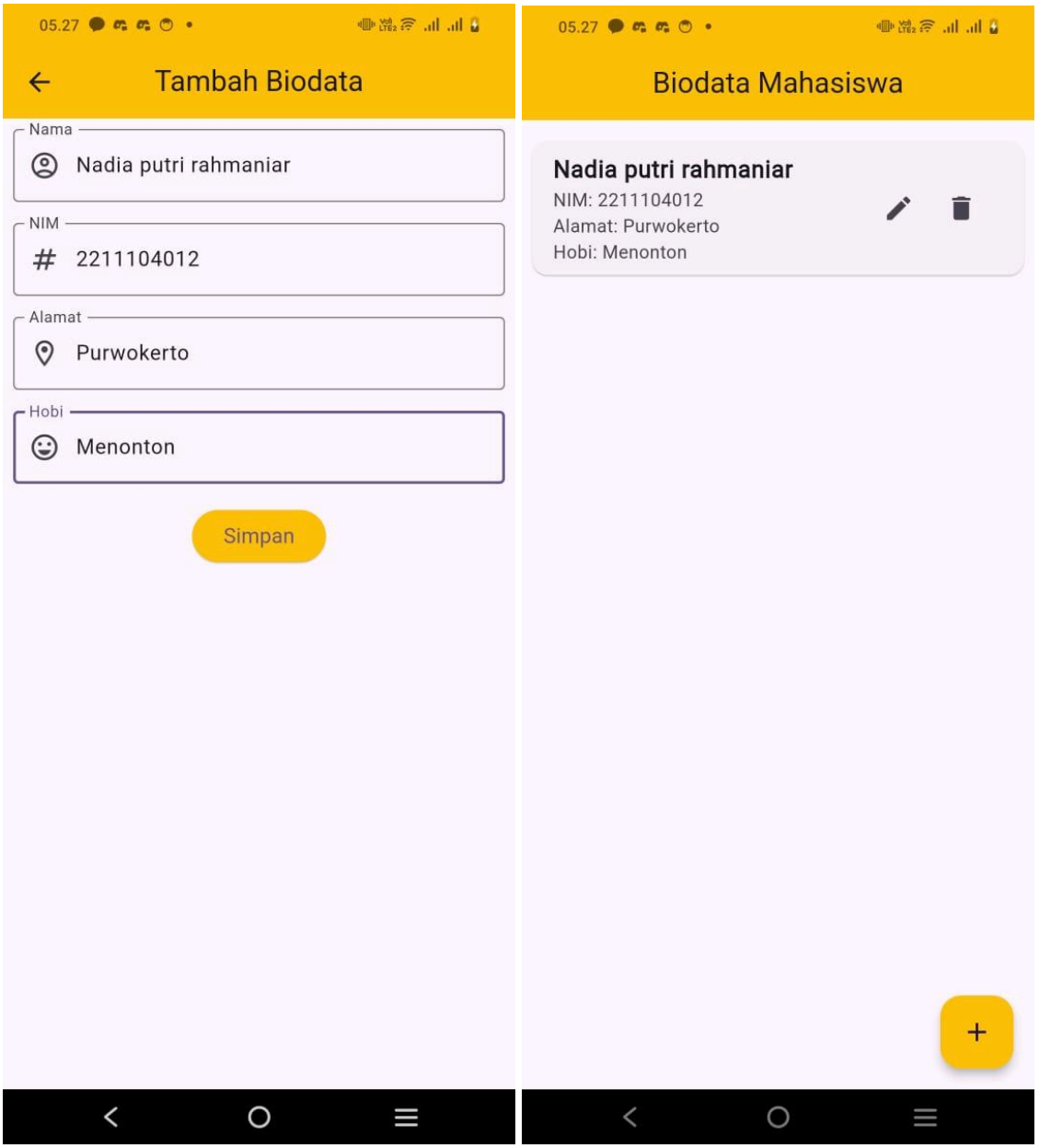
```
Colors.amber),
              ),
            ],
          ),
        ),
      );
    }
}
```

2. **Screenshot Output**

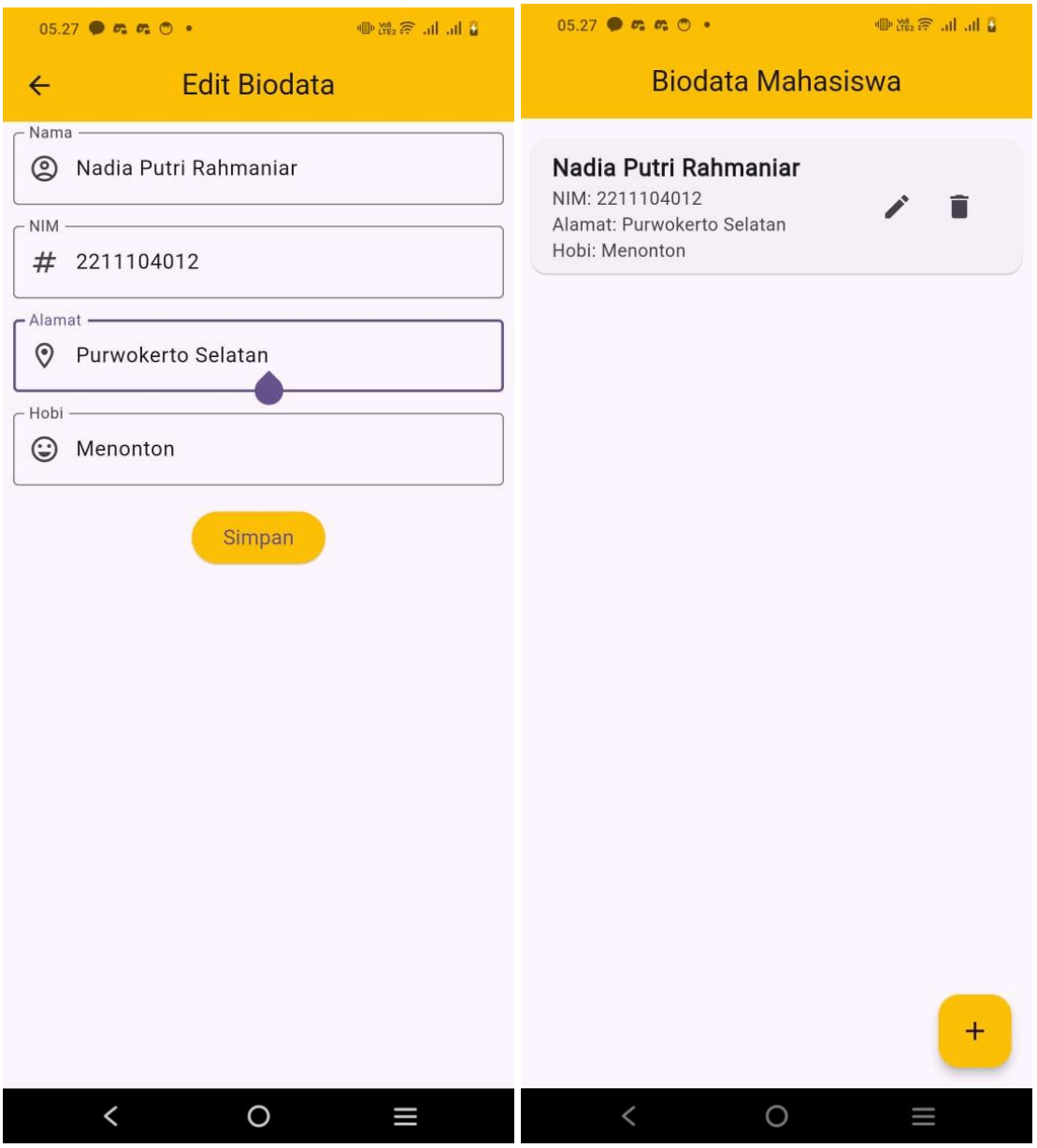
Ketika belum ada data yang dimasukkan, ini adalah **tampilan utama program**. Jika user ingin mengisi biodata mahasiswa, mereka harus menekan tombol plus di pojok kanan bawah untuk melihat form "Tambah Biodata".



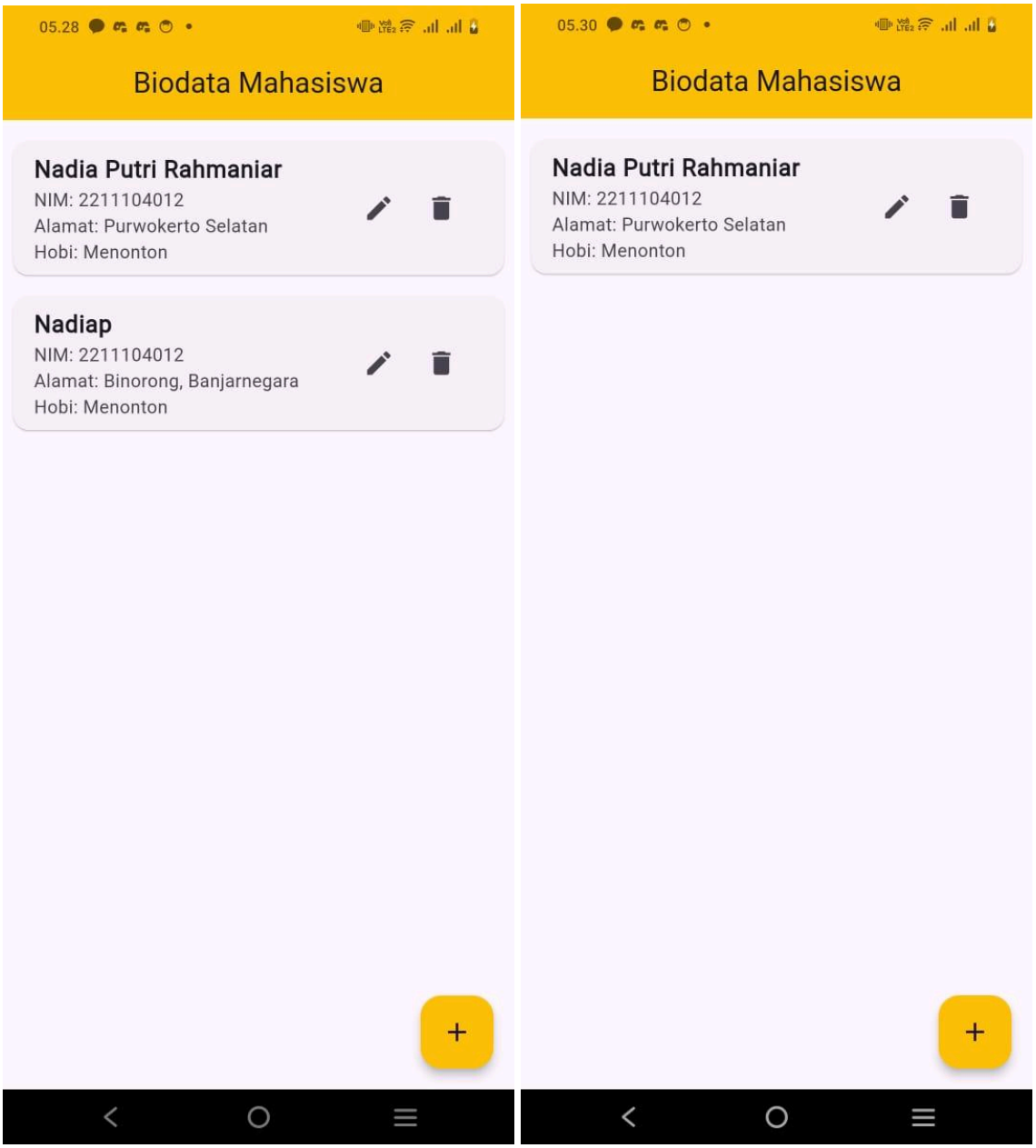
Ini adalah tampilan yang muncul saat user mengisi form tambah biodata. Setelah klik tombol "simpan", halaman utama akan menampilkan semua data yang ada.



Ketika ingin mengedit, user dapat klik icon "pensil". Kemudian user dapat mengubah data, seperti mengubah alamat “Purwokerto” menjadi “Purwokerto Selatan”.



Disini sudah menambahkan menjadi dua data lalu jika ingin menghapus data, bisa dengan menekan ikon tong sampah, maka data akan dihapus.



3. Deskripsi Program

Aplikasi ini memanfaatkan SQLite untuk mengelola informasi biodata mahasiswa secara lokal. Fitur utamanya memungkinkan pengguna untuk menambahkan, memperbarui, dan menghapus data mahasiswa. Semua data yang diolah akan disimpan secara offline menggunakan database SQLite. Berikut adalah penjelasan tiap bagian program:

a. File main.dart : Berfungsi sebagai titik awal aplikasi, file ini menampilkan halaman utama (mainView), yang berupa daftar mahasiswa yang telah tersimpan di database.
b. File db_helper.dart : File ini bertugas menangani operasi database SQLite. Tugasnya meliputi inisialisasi database, pembuatan tabel, serta pengelolaan data seperti penambahan (insert), pembaruan (update), dan penghapusan (delete). Tabel ini menyimpan informasi seperti ID, nama, NIM, alamat, dan hobi mahasiswa.
c. Halaman Utama (MainView) : Halaman ini berfungsi menampilkan daftar data mahasiswa yang tersimpan. Jika tidak ada data, pesan "Belum ada data mahasiswa" akan muncul. Pengguna dapat menambahkan data baru melalui tombol tambah (FloatingActionButton) atau mengubah dan menghapus data yang sudah ada dengan ikon khusus di daftar.
d. Form Input Data (MyDatabaseView) : Bagian ini menyediakan form untuk memasukkan atau mengedit informasi mahasiswa. Form mencakup kolom untuk nama, NIM, alamat, dan hobi. Jika pengguna mengedit data, nilai yang ada secara otomatis ditampilkan pada kolom input. Data akan disimpan ke database setelah pengguna menekan tombol "Simpan".