LAPORAN PRAKTIKUM
PEMROGRAMAN PERANGKAT BERGERAK

MODUL X

DATA STORAGE (BAGIAN I)

Disusun Oleh :

Dimas Cahyo Margono / 2211104060

SE-06-02

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

2024

GUIDED

A. PUBSPEC.YAML

Sourcecode

```
name: pertemuan10
description: "A new Flutter project."
# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as
CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyRef
erence/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
  sdk: ^3.5.3

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  sqflite: ^2.4.1
  path: ^1.9.0

dev_dependencies:
  flutter_test:
    sdk: flutter
```

```
  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^4.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  # An image asset can refer to one or more resolution-specific "variants", see
  # https://flutter.dev/to/resolution-aware-images

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/to/asset-from-package

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
  #       - asset: fonts/Schyler-Italic.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  #
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/to/font-from-package
```

Deskripsi Program

Kode di atas adalah file konfigurasi proyek Flutter bernama `pertemuan10` yang menggunakan format `pubspec.yaml` untuk mengatur metadata proyek, dependensi, dan pengaturan lainnya. File ini menentukan nama proyek, deskripsi, dan versi awal aplikasi (`1.0.0+1`). Kode juga menetapkan versi SDK Dart (`^3.5.3`) untuk memastikan kompatibilitas lingkungan pengembangan. Dependensi utama yang digunakan adalah `flutter` untuk pengembangan UI, `cupertino_icons` untuk ikon bergaya iOS, `sqflite` untuk manajemen database lokal, dan `path` untuk mengelola jalur file. Selain itu, file ini mencakup dependensi pengembangan seperti `flutter_test` untuk pengujian dan `flutter_lints` untuk membantu menegakkan praktik pengkodean yang baik. Terdapat juga placeholder untuk aset seperti gambar, yang dapat diaktifkan jika dibutuhkan dalam proyek.

B. DBHELPER

Sourcecode

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

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

  // Factory constructor untuk mengembalikan instance singleton
  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!;
  }

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

  // Membuat tabel saat db pertama kali dibuat
  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)
''');
  }

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

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

  // 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]);
  }
}
```

Deskripsi Program

Program di atas adalah kelas `DatabaseHelper` untuk mengelola database SQLite di Flutter menggunakan pola singleton. Database bernama `dimascahyo.db` dibuat dengan tabel `my_table`, berisi kolom `id`, `title`, `description`, dan `createdAt`. Kelas ini menyediakan metode CRUD: `insert` untuk menambah data, `queryAllRows` untuk membaca semua data, `update` untuk memperbarui data berdasarkan `id`, dan `delete` untuk menghapus data berdasarkan `id`. Dengan ini, pengelolaan database menjadi sederhana dan terorganisir.

C. MYDBVIEW

Sourcecode

```
import 'package:flutter/material.dart';
import 'package:pertemuan10/helper/db_helper.dart';
```

```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();
  }

  // Metode untuk memperbarui data dari database
  void _refreshData() async {
    final data = await dbHelper.queryAllRows();
    setState(() {
      _dbData = data;
    });
  }

  // Metode untuk menambahkan data ke database
  void _addData() async {
    await dbHelper.insert({
      'title': _titleController.text,
      'description': _descriptionController.text,
    });
    _titleController.clear();
    _descriptionController.clear();
    _refreshData();
  }

  // Metode untuk memperbarui data dalam database
  void _updateData(int id) async {
    await dbHelper.update({
      'id': id,
      'title': _titleController.text,
      'description': _descriptionController.text,
    });
    _titleController.clear();
    _descriptionController.clear();
```

```
  _refreshData();
}

// Metode untuk menghapus data dari database
void _deleteData(int id) async {
  await dbHelper.delete(id);
  _refreshData();
}

// Menampilkan dialog untuk mengedit data
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'),
          ),
        ],
      );
    },
  );
```

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.blue,
      centerTitle: true,
      title: const Text('Praktikum Database - sqlfile'),
    ),
    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']),
                    ),
                  ],
```

```
                ),
              );
            },
          ),
        ),
      ],
    ),
  );
 }
}
```

Deskripsi Program

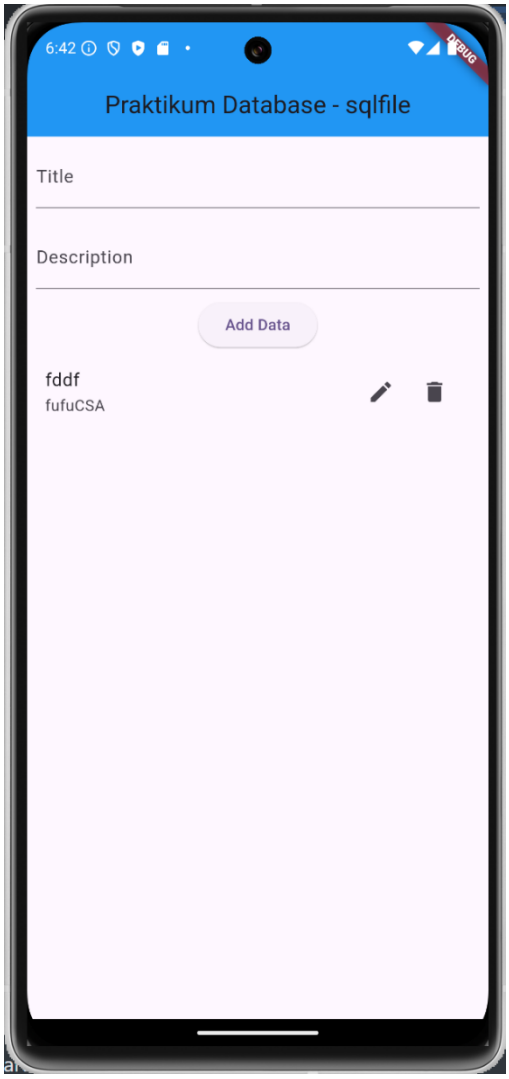
Kode di atas adalah widget Flutter untuk mengelola data SQLite dengan operasi CRUD (Create, Read, Update, Delete). Menggunakan `DatabaseHelper`, data ditampilkan dalam ListView dan dapat ditambah, diperbarui, atau dihapus. Input dilakukan melalui TextField, dan dialog digunakan untuk pengeditan. Metode `_refreshData` memastikan data selalu diperbarui secara real-time di tampilan. Widget ini mempermudah pengelolaan database langsung dari aplikasi.

D. MAIN PROGRAM

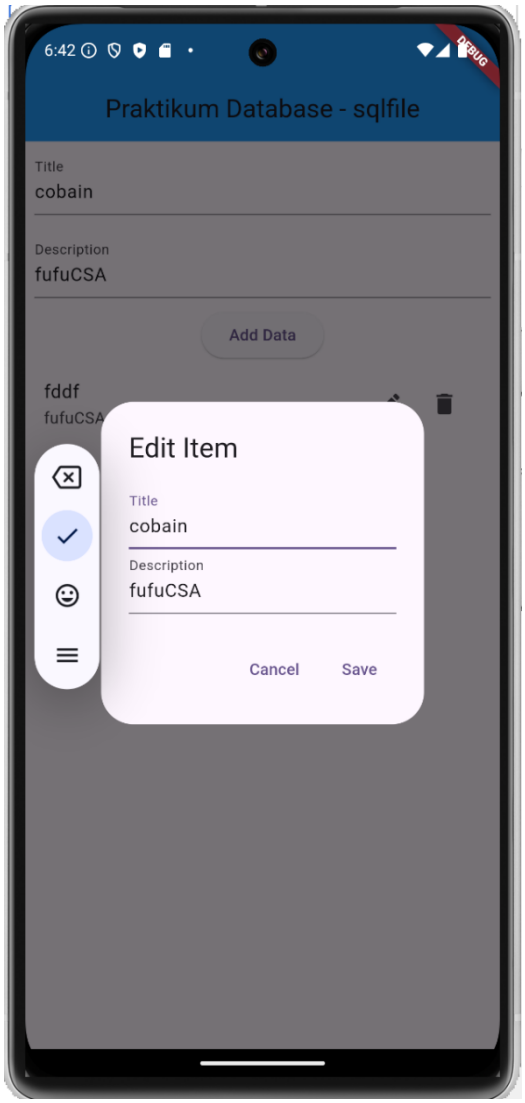
Sourcecode

```dart
import 'package:flutter/material.dart';
import 'package:pertemuan10/view/my_db_view.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyDatabaseView(), // Arahkan ke MyDatabaseView
    );
  }
}
```

Screenshoot Output



Tampilan aplikasi



Form edit data

Deskripsi Program

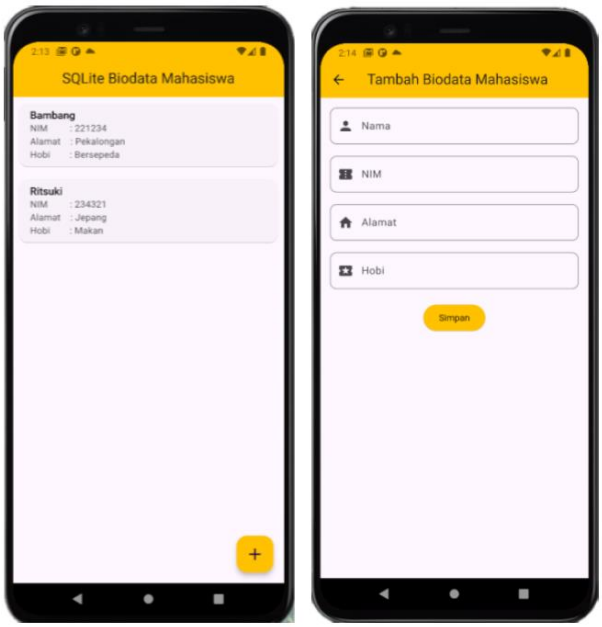
Aplikasi ini adalah Flutter CRUD sederhana menggunakan SQLite, memungkinkan pengguna menambahkan, melihat, mengedit, dan menghapus data melalui antarmuka dengan input Title dan Description. Data ditampilkan dalam ListView dan diperbarui secara real-time, memudahkan praktik pengelolaan database lokal.

UNGUIDED

1. (Soal) 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:



Berikut merupakan sourcecode dari program

A. DBHELPER

Sourcecode

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

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

  // Factory constructor untuk mengembalikan instance singleton
  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!;
  }

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

  // Membuat tabel saat database pertama kali dibuat
  Future<void> _onCreate(Database db, int version) async {
    await db.execute('''
CREATE TABLE students (
  id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  name TEXT NOT NULL,
  nim TEXT NOT NULL,
  address TEXT NOT NULL,
  hobby TEXT NOT NULL,
  createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
)
''');
  }
```

```dart
  // Metode untuk memasukkan data ke dalam tabel
  Future<int> insert(Map<String, dynamic> row) async {
    Database db = await database;
    return await db.insert('students', row);
  }

  // Metode untuk mengambil semua data dari tabel
  Future<List<Map<String, dynamic>>> queryAllRows() async {
    Database db = await database;
    return await db.query('students');
  }

  // 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('students', 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('students', where: 'id = ?', whereArgs: [id]);
  }
}
```

B. MODELS/STUDENT.DART

Sourcecode

```dart
class Student {
  int? id;
  String name;
  String nim;
  String address;
  String hobby;

  Student({
    this.id,
    required this.name,
    required this.nim,
    required this.address,
    required this.hobby,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'nim': nim,
      'address': address,
      'hobby': hobby,
    };
  }
```

```
  factory Student.fromMap(Map<String, dynamic> map) {
    return Student(
      id: map['id'],
      name: map['name'],
      nim: map['nim'],
      address: map['address'],
      hobby: map['hobby'],
    );
  }
}
```

C. VIEW/FORM_PAGE.DART

Sourcecode

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

class FormPage extends StatefulWidget {
  final Map<String, dynamic>?
      item; // Data untuk edit (null jika tambah data baru)

  const FormPage({Key? key, this.item}) : super(key: key);

  @override
  State<FormPage> createState() => _FormPageState();
}

class _FormPageState extends State<FormPage> {
  final DatabaseHelper dbHelper = DatabaseHelper();

  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _nimController = TextEditingController();
  final TextEditingController _addressController = TextEditingController();
  final TextEditingController _hobbyController = TextEditingController();

  @override
  void initState() {
    super.initState();
    // Jika item tidak null, berarti ini mode edit, maka isi form dengan data
    if (widget.item != null) {
      _nameController.text = widget.item!['name'];
      _nimController.text = widget.item!['nim'];
      _addressController.text = widget.item!['address'];
      _hobbyController.text = widget.item!['hobby'];
    }
  }

  @override
  void dispose() {
    _nameController.dispose();
    _nimController.dispose();
    _addressController.dispose();
```

```
    _hobbyController.dispose();
    super.dispose();
  }

  // Simpan data ke database
  void _saveData() async {
    if (_nameController.text.isEmpty ||
        _nimController.text.isEmpty ||
        _addressController.text.isEmpty ||
        _hobbyController.text.isEmpty) {
      // Tampilkan pesan error jika ada input kosong
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Semua kolom harus diisi!')),
      );
      return;
    }

    if (widget.item == null) {
      // Tambah data baru
      await dbHelper.insert({
        'name': _nameController.text,
        'nim': _nimController.text,
        'address': _addressController.text,
        'hobby': _hobbyController.text,
      });
    } else {
      // Update data yang ada
      await dbHelper.update({
        'id': widget.item!['id'],
        'name': _nameController.text,
        'nim': _nimController.text,
        'address': _addressController.text,
        'hobby': _hobbyController.text,
      });
    }

    Navigator.pop(context, true); // Kembali ke halaman sebelumnya
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.item == null
            ? 'Tambah Biodata Mahasiswa'
            : 'Edit Biodata Mahasiswa'),
        backgroundColor: Colors.blueAccent,
        centerTitle: true,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: SingleChildScrollView(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
```

```
          children: [
            TextField(
              controller: _nameController,
              decoration: InputDecoration(
                labelText: 'Nama',
                prefixIcon: const Icon(Icons.person),
                border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(8.0)),
              ),
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _nimController,
              decoration: InputDecoration(
                labelText: 'NIM',
                prefixIcon: const Icon(Icons.numbers),
                border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(8.0)),
              ),
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _addressController,
              decoration: InputDecoration(
                labelText: 'Alamat',
                prefixIcon: const Icon(Icons.home),
                border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(8.0)),
              ),
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _hobbyController,
              decoration: InputDecoration(
                labelText: 'Hobi',
                prefixIcon: const Icon(Icons.interests),
                border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(8.0)),
              ),
            ),
            const SizedBox(height: 24),
            Center(
              child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.blueAccent,
                  foregroundColor: Colors.white,
                  shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(8.0)),
                  padding: const EdgeInsets.symmetric(
                      horizontal: 32.0, vertical: 12.0),
                ),
                onPressed: _saveData,
                child: const Text('Simpan', style: TextStyle(fontSize: 16.0)),
              ),
```

```
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

D. VIEW/LIST_PAGE.DART

Sourcecode

```dart
import 'package:flutter/material.dart';
import 'package:pertemuan101/helper/db_helper.dart';
import 'package:pertemuan101/view/form_page.dart';

class ListPage extends StatefulWidget {
  const ListPage({Key? key}) : super(key: key);

  @override
  State<ListPage> createState() => _ListPageState();
}

class _ListPageState extends State<ListPage> {
  final DatabaseHelper dbHelper = DatabaseHelper();
  List<Map<String, dynamic>> _dbData = [];

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

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('SQLite Biodata Mahasiswa'),
        backgroundColor: Colors.blueAccent,
        centerTitle: true,
      ),
```

```
      body: _dbData.isEmpty
          ? const Center(child: Text('Belum ada data'))
          : ListView.builder(
              itemCount: _dbData.length,
              itemBuilder: (context, index) {
                final item = _dbData[index];
                return Card(
                  margin:
                      const EdgeInsets.symmetric(vertical: 8, horizontal: 16),
                  child: ListTile(
                    title: Text(item['name']),
                    subtitle: Text(
                      'NIM: ${item['nim']}\nAlamat: ${item['address']}\nHobi:
${item['hobby']}',
                    ),
                    trailing: Row(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        IconButton(
                          icon: const Icon(Icons.edit),
                          onPressed: () {
                            Navigator.push(
                              context,
                              MaterialPageRoute(
                                builder: (context) => FormPage(item: item),
                              ),
                            ).then((_) => _refreshData());
                          },
                        ),
                        IconButton(
                          icon: const Icon(Icons.delete),
                          onPressed: () => _deleteData(item['id']),
                        ),
                      ],
                    ),
                  ),
                );
              },
            ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => const FormPage(),
            ),
          ).then((_) => _refreshData());
        },
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

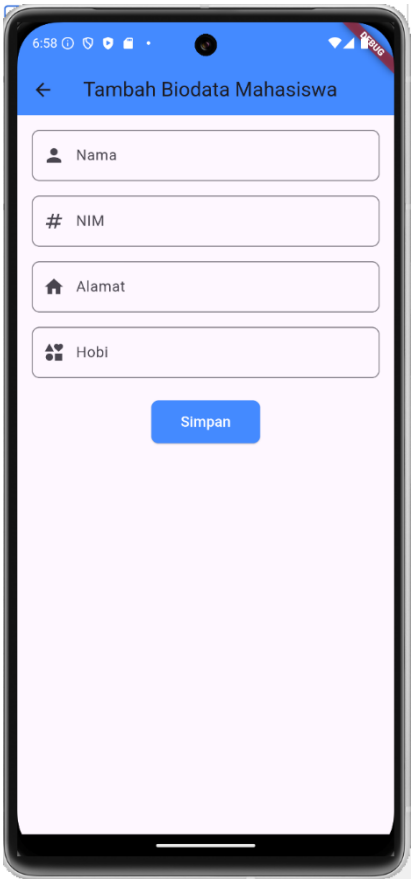
E. MAIN PROGRAM

Sourcecode

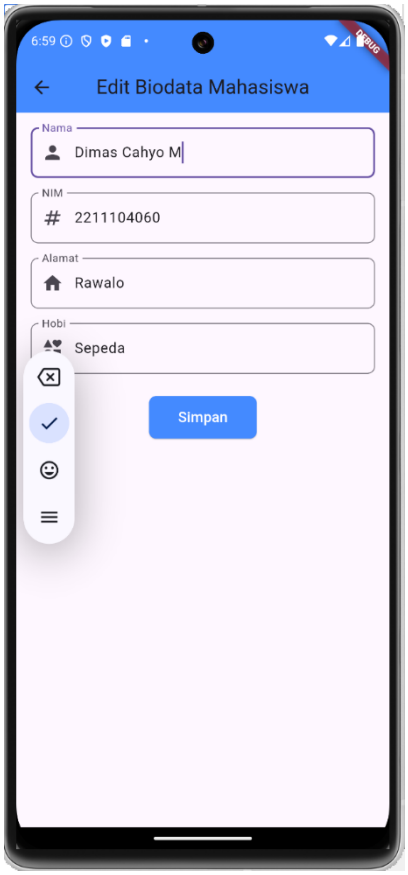
```dart
import 'package:flutter/material.dart';
import 'view/list_page.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'SQLite Biodata Mahasiswa',
      theme: ThemeData(primarySwatch: Colors.yellow),
      home: const ListPage(),
    );
  }
}
```

Screenshoot Output

Tampilan halaman utama



Form tambah data mahasiswa



Form edit data mahasiswa

Deskripsi Program

Aplikasi ini adalah Flutter CRUD untuk mengelola data mahasiswa menggunakan SQLite. Halaman utama menampilkan daftar mahasiswa dengan tombol untuk menambah, mengedit, dan menghapus data. Struktur foldernya terorganisir, terdiri dari helper/db_helper.dart untuk logika database, model/student.dart untuk model data, serta view/form_page.dart dan list_page.dart untuk form input dan tampilan daftar. Aplikasi ini dirancang agar mudah dipahami dan memisahkan tanggung jawab antar komponen.