

# Proyek 4 - Modul 3

## Data Modeling & Dynamic Lists



[https://github.com/dzhax499/PY4\\_2C\\_D3\\_2024\\_Modul\\_071/tree/main/MODUL\\_3](https://github.com/dzhax499/PY4_2C_D3_2024_Modul_071/tree/main/MODUL_3)

071 – Dzakir Tsabit – 24/02/26

## Daftar Isi

MODUL 3: Data Modeling & Dynamic Lists .....	4
Tujuan Pembelajaran.....	4
Materi Esensial .....	4
Target Milestones .....	4
Materi untuk Praktikum .....	4
3.1 Pendahuluan Prinsip SOLID (Liskov & Dependency Inversion).....	5
3.2 Evolusi Proyek: Dari Counter ke Logbook .....	6
3.3 Langkah-Langkah Detail Praktikum.....	6
Langkah 1: Membuat Abstraksi Data (Model).....	6
Langkah 2: Update Controller (Reactive & Persistent) .....	7
Langkah 3: Implementasi Reaktif & Dynamic UI.....	9
Langkah 4: Dialog Input & Edit (CRUD Interactions) .....	10
Langkah 5: Fitur Edit & Delete (Completing CRUD) .....	12
Langkah 6: Level Up - Memahami Reactive Programming.....	12
Analogi Reactive Programming .....	13
Troubleshooting Lab.....	14
Daemon compilation failed: null dan AssertionError .....	14
Tugas Praktikum.....	15
Task 1: Tugas Pendahuluan .....	15
Task 2: The Daily Logger (LOTS).....	16
Task 3: Reactive List Management (MOTS) .....	16
Task 4: Persistent JSON Storage (HOTS) .....	16
Homework Cosmetic & UX Enhancement (30%).....	17
HINTS .....	18
Penilaian .....	19
Kuesioner Self-Assessment.....	19
Peer Assessment (Apresiasi Rekan Sejawat 1).....	21
Peer Assessment (Apresiasi Rekan Sejawat 2).....	22
Peer Assessment (Apresiasi Rekan Sejawat 3).....	23
Template Lesson Learnt (Refleksi Akhir) .....	24
Log LLM: The Fact Check & Twist (Homework) .....	25
Contoh Cara Bertanya yang Benar (Prompting) .....	25
Rubrik Penilaian Dosen Manajer .....	26
Rubrik Penilaian Log LLM (Integritas) .....	26

Referensi .....	28
-----------------	----

# MODUL 3: Data Modeling & Dynamic Lists

Selamat! Anda telah berhasil mengamankan aplikasi. Sekarang, saatnya mengubah aplikasi hitungan sederhana menjadi aplikasi Logbook Digital yang fungsional. Di modul ini, kita akan belajar cara membungkus data ke dalam Model, mengelola kumpulan data dalam List, dan menampilkannya secara dinamis. Kita akan menerapkan prinsip Liskov Substitution dan Dependency Inversion secara sederhana melalui abstraksi data.

## Tujuan Pembelajaran

Setelah menyelesaikan modul ini, diharapkan Anda mampu:

1. Mahasiswa mampu membuat Data Model (Class) untuk standarisasi informasi.
2. Mahasiswa mampu mengimplementasikan ListView.builder untuk menampilkan data dinamis yang efisien.
3. Mahasiswa memahami cara melakukan manipulasi data (CRUD dasar: Create & Read) pada sebuah List.
4. Mahasiswa mampu menerapkan desain UI menggunakan Card dan ListTile.

## Materi Esensial

Sebelum kita mulai menulis kode, pahami dahulu dua pilar utama hari ini:

1. Data Modeling: Mengapa kita butuh Class LogModel daripada sekadar String atau Map.
2. ListView.builder: Teknik rendering daftar yang hemat memori (hanya merender yang terlihat di layar).
3. List Manipulation: Menambah (add) dan menghapus (remove) objek di dalam List.
4. Custom Widgets: Memecah UI menjadi bagian kecil agar Reusable.

## Target Milestones

Sebelum pulang dari lab hari ini, pastikan kalian sudah mencentang semua daftar di bawah ini:

TABEL 3.1: Daftar Target Milestones Modul 3 dan Indikator Keberhasilan.

Milestone	Indikator Keberhasilan
Data Standard	Berhasil membuat Class Model dan mengonversi data mentah menjadi Object.
Dynamic UI	Daftar catatan muncul secara dinamis menggunakan ListView.builder.
Interaction	User bisa menambah catatan baru melalui dialog/form sederhana.
Code Structure	Mengimplementasikan folder models dan memisahkan logika List di Controller.

## Materi untuk Praktikum

Pada bagian ini, kita tidak akan membuat proyek baru. Kita akan tetap memodifikasi proyek logbook\_app yang telah kita bangun sejak Minggu 1 dan 2. Tujuannya adalah mentransformasi

fitur Counter sederhana menjadi sistem Logbook yang sebenarnya. Kita akan menambahkan kemampuan untuk mencatat banyak aktivitas dalam bentuk daftar (List) yang terorganisir menggunakan Data Model, sehingga aplikasi tidak hanya menghitung angka, tetapi mampu menyimpan dan menampilkan riwayat informasi secara dinamis.

### 3.1 Pendahuluan Prinsip SOLID (Liskov & Dependency Inversion)

Pada Modul 3, kita melangkah dari sekadar memisahkan file (SRP) menuju pembangunan sistem yang fleksibel dan stabil menggunakan dua prinsip tambahan:

1. Liskov Substitution Principle (LSP)

“Subclass harus mendukung semua fungsi Superclass-nya sehingga pertukaran objek tidak memicu error.”

Dalam konteks Logbook, kita akan memastikan bahwa berbagai jenis catatan (misal: Catatan Kuliah atau Catatan Tugas) dapat diperlakukan sama sebagai sebuah LogModel oleh UI kita.

```
// Superclass yang stabil
abstract class BaseLog {
    String get title;
}

// Subclass yang menggantikan tanpa merusak fungsi
class LectureLog extends BaseLog {
    @override
    String get title => "Log Kuliah";
}

// UI tetap bisa menampilkan keduanya tanpa tahu tipe spesifiknya
void displayLog(BaseLog log) {
    print(log.title);
}
```

2. Dependency Inversion Principle (DIP)

“Halaman UI (View) tidak boleh mengakses data mentah secara langsung, melainkan harus mengaksesnya melalui Model (Abstraksi).”

Artinya, LogView (Tampilan) tidak peduli bagaimana data disimpan (JSON/SharedPrefs), ia hanya peduli pada Abstraksi Data (Model).

```
// Abstraksi (Model) sebagai jembatan
class LogModel {
    final String content;
    LogModel(this.content);
}

// Controller (Modul Tinggi) mengelola LIST OF MODEL,
```

```
// bukan mengelola String mentah dari Storage.  
class LogController {  
    List<LogModel> data = [];  
}
```

**Analogi Programmer:** "Bayangkan Anda sedang membuat **Playlist Lagu**.

1. Liskov: Tidak peduli lagu itu berformat .mp3 atau .wav (Subclass), pemutar musik (UI) harus bisa memutar keduanya karena keduanya adalah 'Lagu' (Superclass).
2. Dependency Inversion: Pemutar musik tidak perlu tahu lagu itu disimpan di kartu memori atau streaming (Data Mentah). Pemutar musik hanya perlu tahu judul dan durasi yang disediakan oleh Model Lagu."

Sebelum mulai menulis kode, kita akan melakukan "bedah rumah". Dalam proyek skala besar, menyimpan semua file di satu folder lib adalah bencana. Kita akan menerapkan Modular Folder Structure.

Untuk mendukung prinsip di atas, kita akan menerapkan Dependency Inversion dengan memisahkan model ke folder tersendiri agar bisa diakses oleh View maupun Controller secara independen.

```
lib/features/logbook/  
└── models/  
    └── log_model.dart      <-- Abstraksi Data (Modul Tengah)  
└── widgets/  
    └── log_item_widget.dart <-- Reusable UI  
└── log_controller.dart   <-- Logika Bisnis (Modul Tinggi)  
└── log_view.dart         <-- Antarmuka (Modul Rendah)
```

### 3.2 Evolusi Proyek: Dari Counter ke Logbook

Pada Modul 2, kita telah berhasil membangun sistem keamanan login dan menyimpan satu nilai angka sederhana (Counter). Namun, aplikasi nyata membutuhkan kemampuan untuk mengelola data yang lebih kompleks dan berjumlah banyak.

Di Modul 3 ini, kita akan melakukan Refactoring (pengubahan struktur kode tanpa mengubah fungsi utama). Kita akan mengubah variabel int menjadi List<LogModel>. Teknik penyimpanan Shared Preferences dan JSON yang menjadi tantangan di minggu lalu kini kita jadikan sebagai standar dasar (Base Code) aplikasi kita. Dengan ini, aplikasi Anda kini naik kelas: dari sekadar penghitung angka menjadi sistem Persistent Logbook yang mampu mengingat seluruh riwayat catatan Anda secara permanen.

### 3.3 Langkah-Langkah Detail Praktikum

#### Langkah 1: Membuat Abstraksi Data (Model)

- Konsep: Mahasiswa membuat sebuah berkas log\_model.dart yang berisi kelas untuk membungkus tiga informasi utama: judul, tanggal, dan deskripsi catatan.

- Logika: Menggunakan kelas Dart dengan constructor untuk standarisasi data, serta menyediakan fungsi `toMap` dan `fromMap` agar data objek bisa dikonversi menjadi format JSON yang dipahami oleh media penyimpanan.
- Tujuan: Menerapkan prinsip Dependency Inversion, di mana View dan Controller bergantung pada satu standar model yang sama, bukan pada data mentah.

Buat file baru di `lib/features/logbook/models/log_model.dart`. Sesuai prinsip DIP, ini adalah standar data yang akan digunakan oleh View dan Controller.

```
class LogModel {
    final String title;
    final String date;
    final String description;

    LogModel({
        required this.title,
        required this.date,
        required this.description,
    });

    // Untuk Tugas HOTS: Konversi Map (JSON) ke Object
    factory LogModel.fromMap(Map<String, dynamic> map) {
        return LogModel(
            title: map['title'],
            date: map['date'],
            description: map['description'],
        );
    }

    // Konversi Object ke Map (JSON) untuk disimpan
    Map<String, dynamic> toMap() {
        return {
            'title': title,
            'date': date,
            'description': description,
        };
    }
}
```

## Langkah 2: Update Controller (Reactive & Persistent)

- Konsep: Memodifikasi `LogController` agar menggunakan `ValueNotifier` untuk reaktivitas dan Shared Preferences untuk penyimpanan permanen.
- Logika: Fungsi `addLog`, `updateLog`, dan `removeLog` secara otomatis memperbarui `logsNotifier` dan memicu fungsi `saveToDisk`.
- Tujuan: Memahami manajemen state reaktif dan sinkronisasi data ke penyimpanan lokal.

Buka `lib/features/logbook/log_controller.dart`.

```

import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../models/log_model.dart';

class LogController {
  final ValueNotifier<List<LogModel>> logsNotifier = ValueNotifier([]);
  static const String _storageKey = 'user_logs_data';

  LogController() { loadFromDisk(); }

  void addLog(String title, String desc) {
    final newLog = LogModel(title: title, description: desc, date:
    DateTime.now().toString());
    logsNotifier.value = [...logsNotifier.value, newLog];
    saveToDisk();
  }

  void updateLog(int index, String title, String desc) {
    final currentLogs = List<LogModel>.from(logsNotifier.value);
    currentLogs[index] = LogModel(title: title, description: desc, date:
    DateTime.now().toString());
    logsNotifier.value = currentLogs;
    saveToDisk();
  }

  void removeLog(int index) {
    final currentLogs = List<LogModel>.from(logsNotifier.value);
    currentLogs.removeAt(index);
    logsNotifier.value = currentLogs;
    saveToDisk();
  }

  Future<void> saveToDisk() async {
    final prefs = await SharedPreferences.getInstance();
    final String encodedData = jsonEncode(logsNotifier.value.map((e) =>
    e.toMap()).toList());
    await prefs.setString(_storageKey, encodedData);
  }

  Future<void> loadFromDisk() async {
    final prefs = await SharedPreferences.getInstance();
    final String? data = prefs.getString(_storageKey);
    if (data != null) {
      final List decoded = jsonDecode(data);
      logsNotifier.value = decoded.map((e) => LogModel.fromMap(e)).toList();
    }
  }
}

```

```
}
```

### Langkah 3: Implementasi Reaktif & Dynamic UI

- Konsep: Menggunakan ValueListenableBuilder untuk memantau data di Controller.
- Logika: Jika data di logsNotifier berubah, UI akan merender ulang ListView.builder secara otomatis tanpa perlu setState.
- Tujuan: Memahami efisiensi rendering daftar dinamis di Flutter dan menerapkan prinsip Liskov Substitution, di mana UI dapat menampilkan objek apa pun selama ia merupakan turunan dari Model yang disepakati.

File: lib/features/logbook/log\_view.dart (Bagian Body)

```
body: ValueListenableBuilder<List<LogModel>>(
    valueListenable: _controller.logsNotifier,
    builder: (context, currentLogs, child) {
        if (currentLogs.isEmpty) return const Center(child: Text("Belum ada catatan."));
        return ListView.builder(
            itemCount: currentLogs.length,
            itemBuilder: (context, index) {
                final log = currentLogs[index];
                return Card(
                    child: ListTile(
                        leading: const Icon(Icons.note),
                        title: Text(log.title),
                        subtitle: Text(log.description),
                        trailing: Row(
                            mainAxisAlignment: MainAxisAlignment.end,
                            children: [
                                IconButton(icon: const Icon(Icons.edit), color: Colors.blue),
                                IconButton(icon: const Icon(Icons.delete), color: Colors.red),
                            ],
                        ),
                    ),
                );
            },
        ),
    ),
);
```

## Langkah 4: Dialog Input & Edit (CRUD Interactions)

- **Konsep:** Membuat dialog *overlay* untuk mengambil input atau mengubah data lama (*prefilled*).
- **Logika:** Menggunakan `TextEditingController` untuk menangkap teks dan mengirimkannya ke metode `addLog` atau `updateLog` pada Controller.
- **Tujuan:** Melatih kemampuan mengelola interaksi input pengguna dan navigasi dialog.

Tambahkan fungsi didalam kelas `_LogViewState`: Kode untuk lib/features/logbook/log\_view.dart:

```
// 1. Tambahkan Controller untuk menangkap input di dalam State
final TextEditingController _titleController = TextEditingController();
final TextEditingController _contentController = TextEditingController();

void _showAddLogDialog() {
    showDialog(
        context: context,
        builder: (context) => AlertDialog(
            title: const Text("Tambah Catatan Baru"),
            content: Column(
                mainAxisAlignment: MainAxisAlignment.min, // Agar dialog tidak memenuhi layar
                children: [
                    TextField(
                        controller: _titleController,
                        decoration: const InputDecoration(hintText: "Judul Catatan"),
                    ),
                    TextField(
                        controller: _contentController,
                        decoration: const InputDecoration(hintText: "Isi Deskripsi"),
                    ),
                ],
            ),
            actions: [
                TextButton(
                    onPressed: () => Navigator.pop(context), // Tutup tanpa simpan
                    child: const Text("Batal"),
                ),
                ElevatedButton(
                    onPressed: () {
                        // Jalankan fungsi tambah di Controller
                        _controller.addLog(
                            _titleController.text,
                            _contentController.text
                        );
                    },
                    // Trigger UI Refresh
                    setState(() {});
                ),
            ],
        ),
    );
}

// Bersihkan input dan tutup dialog
```

```
        _titleController.clear();
        _contentController.clear();
        Navigator.pop(context);
    },
    child: const Text("Simpan"),
),
],
),
);
}
}
```

```
void _showEditLogDialog(int index, LogModel log) {
    _titleController.text = log.title;
    _contentController.text = log.description;
    showDialog(
        context: context,
        builder: (context) => AlertDialog(
            title: const Text("Edit Catatan"),
            content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                    TextField(controller: _titleController),
                    TextField(controller: _contentController),
                ],
            ),
            actions: [
                TextButton(onPressed: () => Navigator.pop(context), child: const Text("Batal")),
                ElevatedButton(
                    onPressed: () {
                        _controller.updateLog(index, _titleController.text,
                        _contentController.text);
                        _titleController.clear();
                        _contentController.clear();
                        Navigator.pop(context);
                    },
                    child: const Text("Update"),
                ),
            ],
        ),
    );
}
```

Lalu update bagian FloatingActionButton:

```
floatingActionButton: FloatingActionButton(
    onPressed: _showAddLogDialog, // Panggil fungsi dialog yang baru dibuat
    child: const Icon(Icons.add),
```

]),

## Langkah 5: Fitur Edit & Delete (Completing CRUD)

- **Konsep:** Mahasiswa menambahkan interaksi pada setiap item di dalam `ListView` untuk melakukan perubahan atau penghapusan catatan.
- **Logika:**
  - o **Delete:** Memanggil `_controller.removeLog(index)`. Karena menggunakan `ValueNotifier`, UI akan hilang secara otomatis tanpa `setState`.
  - o **Edit:** Memanggil `_showEditLogDialog` yang mengisi `TextField` dengan data lama, lalu menjalankan `_controller.updateLog`.
- Tujuan: Memahami manipulasi data pada koleksi (List) dan sinkronisasi antara UI dengan data di penyimpanan lokal.

Kode Tambahan pada View: cukup menambahkan trailing pada `ListTile`

```
trailing: Wrap(  
  children: [  
    IconButton(  
      icon: const Icon(Icons.edit, color: Colors.blue),  
      onPressed: () => _showEditLogDialog(index, log), // Fungsi edit  
    ),  
    IconButton(  
      icon: const Icon(Icons.delete, color: Colors.red),  
      onPressed: () {  
        setState(() => _controller.removeLog(index));  
      },  
    ),  
  ],  
)
```

## Langkah 6: Level Up - Memahami Reactive Programming

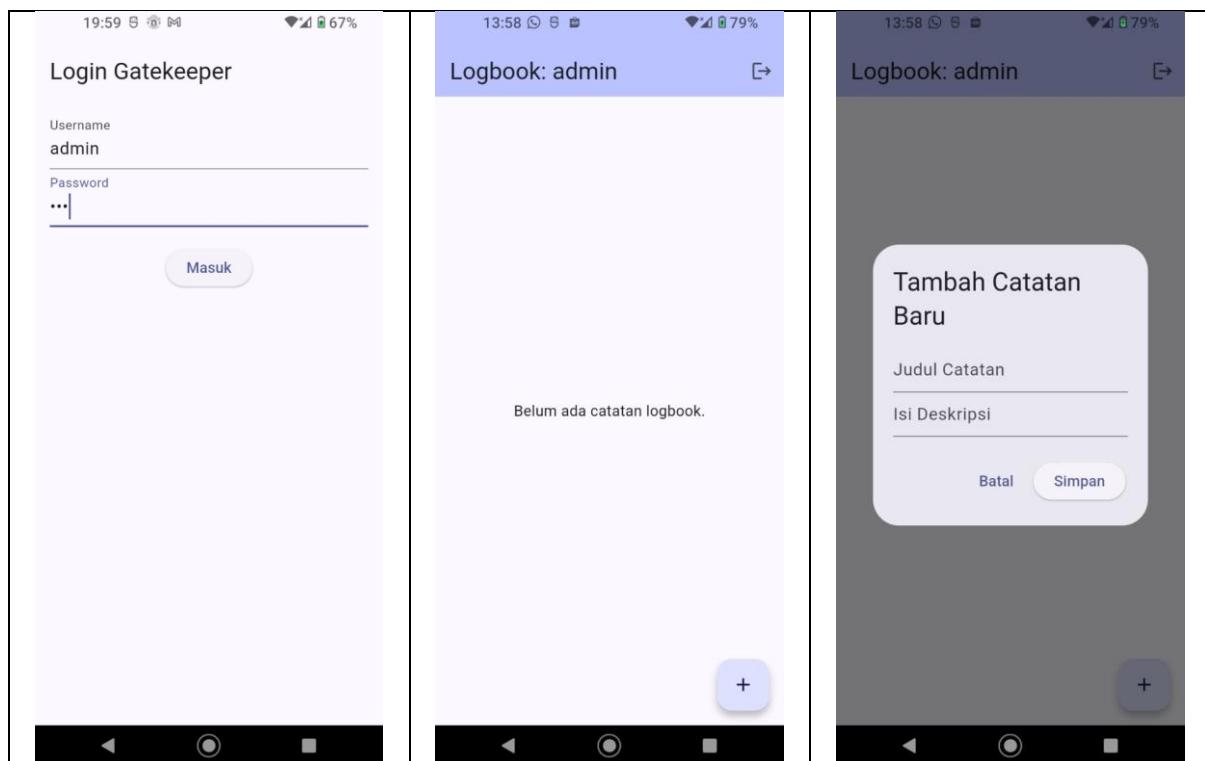
- **Konsep:** Membedakan antara *Imperative Programming* (Modul 2) dan *Reactive Programming* (Modul 3).
- Logika:
  - o Menganalisis peran `logsNotifier.value = [...]` di Controller sebagai pengirim sinyal perubahan data.
- Tujuan: Menanamkan pemahaman tentang efisiensi rendering di Flutter dan pengurangan boilerplate code.

Fitur	Cara Lama (Modul 2)	Cara Baru (Modul 3)
<b>Manajemen State</b>	<code>setState(() {})</code>	<code>ValueNotifier &amp; ValueListenableBuilder</code>
<b>Pemicu Refresh</b>	Manual di setiap fungsi tombol	Otomatis saat data di Controller berubah
<b>Efisiensi</b>	Merender ulang seluruh halaman	Hanya merender ulang bagian di dalam Builder

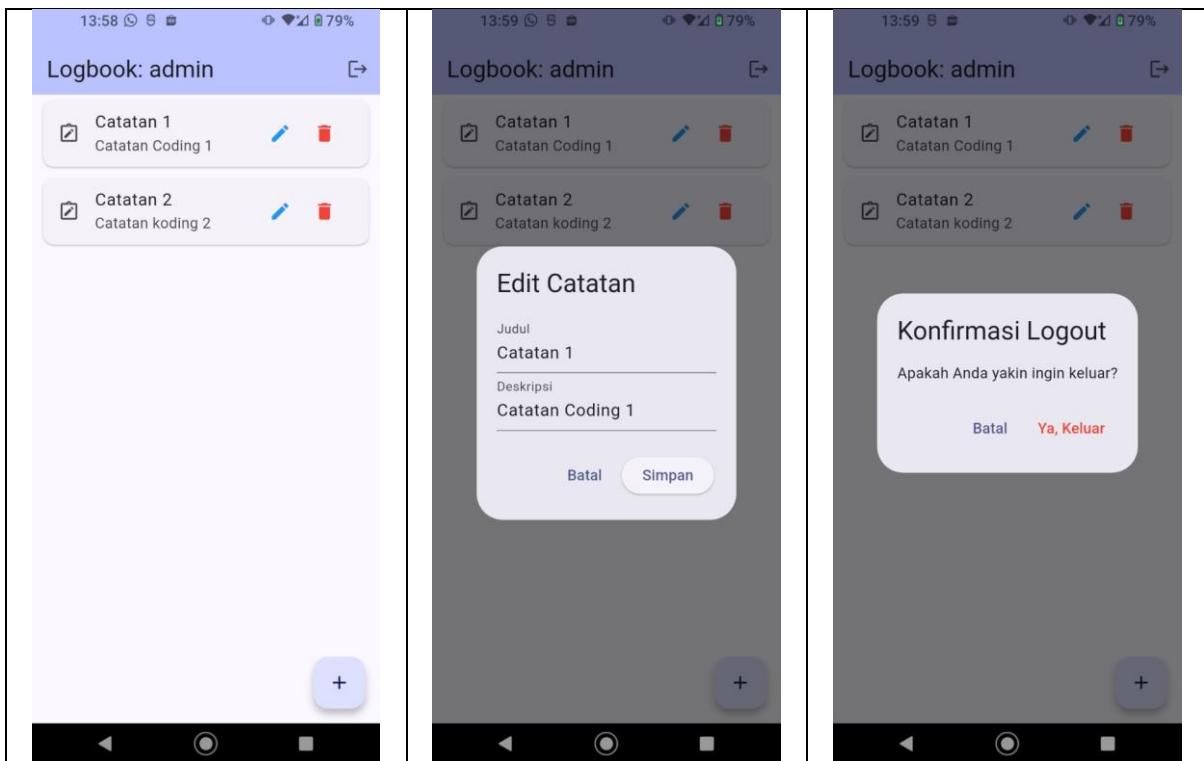
## Analogi Reactive Programming

Bayangkan Anda sedang menunggu paket.

1. Cara Manual (setState): Anda harus keluar rumah setiap 5 menit untuk mengecek pagar apakah paket sudah datang.
2. Cara Reaktif (ValueNotifier): Anda duduk santai di dalam rumah. Saat paket datang, kurir menekan Bel (Notifier). Mendengar suara bel, Anda otomatis berdiri dan mengambil paket. Lebih efisien, bukan?"



GAMBAR 1.1: Penampakan aplikasi Logbook dengan implementasi Reactive Curd Part 1.



GAMBAR 1.2: Penampakan aplikasi Logbook dengan implementasi Reactive Curd Part 2.

## Troubleshooting Lab

Gejala	Penyebab	Solusi
<b>Error: RenderBox was not laid out</b>	Biasanya terjadi jika <code>ListView.builder</code> diletakkan di dalam <code>Column</code> tanpa dibungkus <code>widget Expanded</code> .	Bungkus <code>ListView.builder</code> dengan <code>Expanded()</code> .
<b>Tampilan tidak berubah saat tambah data</b>	Ingin, fungsi <code>addLog</code> di controller tidak otomatis memicu UI.	Pastikan memanggil <code>setState(() {})</code> di dalam fungsi <code> onPressed</code> pada <code>View</code> setelah memanggil <code>_controller.addLog()</code> .
<b>Memory Leak</b>	Biasanya terjadi saat lupa untuk memanggil <code>_titleController.dispose()</code>	jika aplikasi ini sudah semakin kompleks agar tidak memakan memori
<b>Input Kosong</b>	Jika <code>_titleController.text</code> kosong	jangan jalankan fungsi simpan dan munculkan pesan peringatan.

## Daemon compilation failed: null dan AssertionError

Berikut adalah langkah-langkah untuk membersihkan dan memperbaiki error tersebut:

1. Buat variable environment `PUB_CACHE="D:\Tools\pub-cache"`
2. Clean Proyek

```
flutter clean
flutter pub get
```

### 3. Hapus Cache Gradle Secara Manual

Jika Langkah 1 belum berhasil, ada "sampah" kompilasi yang harus dibuang secara manual:

- Buka folder proyek: D:\Projects\Flutter\logbook\_app\_001
- Hapus folder bernama build (ini aman untuk dihapus, akan dibuat ulang saat dijalankan).
- Masuk ke folder android, lalu hapus folder .gradle (folder tersembunyi).

### 4. Jalankan Ulang dengan Cold Boot

Jangan gunakan tombol Hot Reload atau Hot Restart. Lakukan kompilasi dari awal:

- Pastikan emulator atau HP sudah terkoneksi.
- Tekan F5 atau jalankan Flutter run lewat terminal.

## Tugas Praktikum

Untuk menguji sejauh mana pemahaman kalian tentang SRP dan Dart Dasar, selesaikan dua tantangan berikut. Ingat aturan mainnya: 70% harus selesai di Lab (hingga fungsi berjalan), dan 30% sisanya (perapian UI/UX) bisa kalian jadikan homework untuk mempercantik portofolio kalian.

[https://github.com/dzhax499/PY4\\_2C\\_D3\\_2024\\_Modul\\_071/tree/main/MODUL\\_3](https://github.com/dzhax499/PY4_2C_D3_2024_Modul_071/tree/main/MODUL_3)

**Dikerjakan secara mandiri sebelum sesi praktikum dimulai.**

1. Konsep Model: Pelajari cara membuat Class di Dart yang memiliki Constructor dengan Named Parameters.
2. Mapping Data: Cari tahu apa kegunaan fungsi jsonEncode() dan jsonDecode() dari library dart:convert.
3. Analisis: Mengapa menampilkan daftar data menggunakan ListView.builder lebih disarankan daripada menggunakan Column di dalam SingleChildScrollView?

**Jawab:**

1. **Named parameters** adalah parameter yang dipanggil dengan menyebut nama argumennya secara eksplisit menggunakan sintaks namaParameter: nilai. Di Dart, ditulis dengan kurung kurawal {} pada constructor/fungsi. Keunggulannya adalah kode lebih **readable** (mudah dibaca), urutan pengisian **bebas**, dan bisa ditandai required agar wajib diisi sehingga mencegah kesalahan saat objek dibuat.
2. Keduanya dari library dart:convert digunakan untuk mengubah data Dart ke format JSON (teks) dan sebaliknya, karena SharedPreferences hanya bisa menyimpan tipe primitif seperti String.
3. ListView.builder lebih disarankan karena menggunakan teknik lazy rendering hanya membuat widget yang sedang tampak di layar, bukan semuanya sekaligus. Data nya dinamis, memakan memory lebih sedikit karena jika ada 1000 data = yang ditampilkan hanya yang aktif

## Task 2: The Daily Logger (LOTS)

**Fokus:** Mengimplementasikan CRUD dasar dan menampilkan data dinamis.

Spesifikasi Tugas:

1. Data Modeling: Buat file log\_model.dart dengan minimal 3 properti: title, description, dan timestamp.
2. List Interaction: Implementasikan fitur Tambah (via dialog), Edit (mengubah konten yang sudah ada), dan Delete (menghapus item berdasarkan index).
3. Dynamic Rendering: Gunakan ListView.builder untuk merender setiap item ke dalam widget Card atau ListTile.

Kriteria Selesai di Lab (70%):

- [x] Berhasil menambah, mengedit, dan menghapus item dari daftar secara real-time.
- [x] Daftar catatan tampil rapi menggunakan widget ListView.builder.
- [x] Menggunakan Class Model sebagai standar data (bukan String mentah).

## Task 3: Reactive List Management (MOTS)

**Fokus:** Menerapkan pemrograman reaktif agar UI tersinkronisasi otomatis.

Spesifikasi Tugas:

1. Reactive Logic: Ubah pengelolaan List di LogController menggunakan ValueNotifier atau ChangeNotifier.
2. Auto-Update UI: Gunakan ValueListenableBuilder di LogView sehingga pemanggilan setState() di dalam View bisa diminimalisir atau dihapus.
3. Data Integrity: Pastikan saat data diubah melalui fungsi edit, UI langsung mencerminkan perubahan tanpa perlu pindah halaman.

Kriteria Selesai di Lab (70%):

- [x] UI terupdate secara otomatis tanpa penggunaan setState() manual pada fungsi CRUD.
- [x] Logika reaktif terpusat di dalam file Controller.

## Task 4: Persistent JSON Storage (HOTS)

**Fokus:** Menyimpan kumpulan objek ke penyimpanan lokal menggunakan format JSON.

Spesifikasi Tugas:

1. Serialization: Tambahkan fungsi di LogController untuk mengubah List<LogModel> menjadi format JSON String.
2. Local Save: Simpan String JSON tersebut ke dalam Shared Preferences setiap kali terjadi perubahan data (tambah/edit/hapus).
3. Data Restoration: Saat aplikasi dibuka kembali, lakukan parsing dari JSON kembali menjadi List<LogModel> dan tampilkan di layar.

Kriteria Selesai di Lab (70%):

- [x] Seluruh daftar catatan tidak hilang meskipun aplikasi dilakukan Hot Restart atau ditutup.

- [x] Proses encoding (Object to JSON) dan decoding (JSON to Object) berjalan tanpa error.

## Homework Cosmetic & UX Enhancement (30%)

Setelah logika di atas berjalan lancar di lab, gunakan waktu di rumah untuk:

1. Search Feature: Tambahkan sebuah TextField di atas list untuk memfilter catatan berdasarkan judul secara real-time.
2. Empty State: Tampilkan ilustrasi atau gambar menarik jika daftar catatan masih kosong (jangan hanya layar putih).
3. Categories: Tambahkan dropdown kategori (misal: "Pekerjaan", "Pribadi", "Urgent") pada dialog input, dan berikan warna yang berbeda pada setiap kartu sesuai kategorinya.

# HINTS

## 1. Implementasi Fitur Search (Real-time Filtering)

Agar fitur ini berjalan, mahasiswa perlu menambahkan satu variabel list tambahan untuk menampung hasil filter agar data asli tidak hilang.

Di Controller (log\_controller.dart):

```
// List cadangan untuk hasil pencarian
ValueNotifier<List<LogModel>> filteredLogs = ValueNotifier([]);

void searchLog(String query) {
    if (query.isEmpty) {
        filteredLogs.value = logsNotifier.value;
    } else {
        filteredLogs.value = logsNotifier.value
            .where((log) =>
log.title.toLowerCase().contains(query.toLowerCase()))
            .toList();
    }
}
```

Di View (log\_view.dart): Letakkan TextField di atas ListView (dalam sebuah Column):

```
TextField(
    onChanged: (value) => _controller.searchLog(value),
    decoration: const InputDecoration(
        labelText: "Cari Catatan...",
        prefixIcon: Icon(Icons.search),
    ),
),
```

## 2. Implementasi Dismissible (Swipe to Delete)

Ini akan memberikan efek visual di mana item bergeser keluar layar saat dihapus.

Di dalam ListView.builder:

```
return Dismissible(
    key: Key(log.date), // Gunakan identitas unik (timestamp)
    direction: DismissDirection.endToStart, // Swipe dari kanan ke kiri
    background: Container(
        color: Colors.red,
        alignment: Alignment.centerRight,
        padding: const EdgeInsets.only(right: 20),
        child: const Icon(Icons.delete, color: Colors.white),
    ),
    onDismissed: (direction) {
        _controller.removeLog(index);
    }
);
```

```

ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text("Catatan dihapus")),
);
},
child: Card(
    child: ListTile(
        title: Text(log.title),
        subtitle: Text(log.description),
    ),
),
);

```

### 3. JSON Persistence

Logika Loading JSON di Controller:

```

Future<void> loadLogs() async {
    final prefs = await SharedPreferences.getInstance();
    String? rawJson = prefs.getString('saved_logs');

    if (rawJson != null) {
        // 1. Decode String ke List<Map>
        Iterable decoded = jsonDecode(rawJson);

        // 2. Map kembali ke List<LogModel>
        logsNotifier.value = decoded.map((item) =>
LogModel.fromMap(item)).toList();
    }
}

```

## Penilaian

### Kuesioner Self-Assessment

Jawablah dengan jujur sesuai dengan apa yang kalian rasakan setelah menyelesaikan Modul 1. Berikan tanda centang (✓) pada kolom angka yang paling sesuai:

Keterangan Skala: 1: Sangat Tidak Setuju (STS) | 2: Tidak Setuju (TS) | 3: Netral (N) | 4: Setuju (S) | 5: Sangat Setuju (SS)

No	Pernyataan Refleksi	1	2	3	4	5
1	Saya mampu menjelaskan keuntungan menggunakan Class Model dibandingkan variabel biasa.					(✓)
2	Saya memahami cara kerja ListView.builder dalam menghemat penggunaan RAM HP.					(✓)
3	Saya mampu mengubah data objek menjadi format JSON untuk disimpan secara permanen.					(✓)

**4**

Saya mandiri dalam memecahkan error "Unbounded Height" pada Layout Flutter.



Apa bagian yang paling menantang atau membingungkan bagi kalian di modul ini?

Bagian yang paling menantang di Modul 3 adalah memahami konsep Reactive Programming menggunakan ValueNotifier dan ValueListenableBuilder. Awalnya saya terbiasa menggunakan setState() untuk memperbarui UI, sehingga butuh waktu untuk memahami mengapa setState() tidak lagi diperlukan setelah menggunakan ValueNotifier di Controller. Selain itu, memahami struktur widget yang benar ketika menggabungkan beberapa widget dalam satu builder (seperti TextField dan ListView dalam Column + Expanded) juga cukup membingungkan karena menyebabkan error Unbounded Height jika tidak menggunakan Expanded dengan tepat.

## Peer Assessment (Apresiasi Rekan Sejawat 1)

Setelah selesai, tukarkan HP kalian dengan teman sebangku. Cobalah aplikasi mereka dan berikan penilaian sejajar mungkin. Ingat, tujuannya bukan menjatuhkan, tapi saling memperbaiki!

**Nama Penilai:** \_\_\_\_\_ **Nama Pemilik Aplikasi:** \_\_\_\_\_

No	Kriteria Pengalaman Pengguna (UX)	Skor (1-5)	Catatan "Hal Keren/Saran"
1	Kerapihan: Apakah daftar catatan terlihat rapi dan menggunakan Card yang menarik?		
2	Fungsional: Apakah fitur tambah data dan hapus data (swipe) berfungsi tanpa error?		
3	Persistensi: Apakah daftar catatan tetap ada saat aplikasi dibuka kembali?		

Print Screen Aplikasi Teman Anda:

## Peer Assessment (Apresiasi Rekan Sejawat 2)

Setelah selesai, tukarkan HP kalian dengan teman sebangku. Cobalah aplikasi mereka dan berikan penilaian sejajar mungkin. Ingat, tujuannya bukan menjatuhkan, tapi saling memperbaiki!

**Nama Penilai:** \_\_\_\_\_ **Nama Pemilik Aplikasi:** \_\_\_\_\_

No	Kriteria Pengalaman Pengguna (UX)	Skor (1-5)	Catatan "Hal Keren/Saran"
1	Kerapihan: Apakah daftar catatan terlihat rapi dan menggunakan Card yang menarik?		
2	Fungsional: Apakah fitur tambah data dan hapus data (swipe) berfungsi tanpa error?		
3	Persistensi: Apakah daftar catatan tetap ada saat aplikasi dibuka kembali?		

Print Screen Aplikasi Teman Anda:

## Peer Assessment (Apresiasi Rekan Sejawat 3)

Setelah selesai, tukarkan HP kalian dengan teman sebangku. Cobalah aplikasi mereka dan berikan penilaian sejajar mungkin. Ingat, tujuannya bukan menjatuhkan, tapi saling memperbaiki!

**Nama Penilai:** \_\_\_\_\_ **Nama Pemilik Aplikasi:** \_\_\_\_\_

No	Kriteria Pengalaman Pengguna (UX)	Skor (1-5)	Catatan "Hal Keren/Saran"
1	Kerapihan: Apakah daftar catatan terlihat rapi dan menggunakan Card yang menarik?		
2	Fungsional: Apakah fitur tambah data dan hapus data (swipe) berfungsi tanpa error?		
3	Persistensi: Apakah daftar catatan tetap ada saat aplikasi dibuka kembali?		

Print Screen Aplikasi Teman Anda:



## Template Lesson Learnt (Refleksi Akhir)

Jangan biarkan ilmu hari ini menguap begitu saja. Tuliskan 3 poin utama yang baru kalian sadari atau pahami hari ini.

1. Konsep Baru: (Contoh: Baru tahu kalau \_ itu buat bikin variabel jadi rahasia/private).
2. Kemenangan Kecil: (Contoh: Berhasil benerin error Path yang bikin pusing selama 1 jam).
3. Target Berikutnya: (Contoh: Pengen tahu gimana cara bikin tampilan yang lebih berwarna di bab selanjutnya).

## Log LLM: The Fact Check & Twist (Homework)

Menggunakan AI itu boleh, yang tidak boleh adalah menjadi "Zombi AI" (Copy-Paste tanpa mikir). Gunakan template ini setiap kali kalian meminta bantuan ChatGPT/Gemini/Claude.

### Contoh Cara Bertanya yang Benar (Prompting)

-  **Salah:** "Buatin kode list di Flutter." (Terlalu umum, kalian akan bingung bacanya).
-  **Benar:** "Saya punya class CounterController di Dart. Saya ingin menambah `List<String>` untuk simpan riwayat tiap tombol diklik. Tolong berikan logika murni Dart-nya saja tanpa UI."

Komponen	Isian Mahasiswa
Pertanyaan (Prompt)	"Gimana cara nambahin data ke List tapi dibatasi cuma 5 data terbaru aja di Dart?"
Jawaban AI (Intisari)	AI menyarankan pakai fungsi <code>insert(0, data)</code> lalu pakai <code>removeLast()</code> kalau panjang list > 5.
The Fact Check	Saya coba di DartPad, ternyata <code>removeLast()</code> bakal error kalau List-nya masih kosong. Jadi saya tambahin pengecekan if ( <code>list.isNotEmpty</code> ).
The Twist (Modifikasi)	Saya nggak cuma simpan teks biasa, tapi saya tambahin jam otomatis pakai <code>DateTime.now()</code> supaya lebih informatif seperti aplikasi logbook asli.

## Rubrik Penilaian Dosen Manajer

Tabel 2.2: Rubrik Penilaian Praktikum Modul 1.

Kriteria	Skor 0-50 (Kurang)	Skor 51-80 (Cukup)	Skor 81-100 (Sangat Baik)	Bobot
<b>Data Modeling &amp; Arsitektur (SOLID)</b>	Tidak menggunakan Class Model; data masih dikelola sebagai variabel mentah. Struktur folder berantakan.	Menggunakan Class Model, namun logika CRUD masih tercampur di dalam file View (Melanggar SRP).	Penerapan <b>DIP</b> sempurna; Model dipisahkan, dan View hanya bergantung pada abstraksi Model.	25%
<b>Task 1: Full CRUD &amp; Dynamic UI</b>	Hanya bisa menambah data. Edit atau Delete tidak berfungsi. Tampilan list statis.	CRUD berfungsi lengkap (Tambah, Edit, Delete), namun UI belum menggunakan ListView.builder.	CRUD berfungsi sempurna dengan UI dinamis yang efisien menggunakan ListView.builder dan Card.	25%
<b>Task 2: Reactive Programming</b>	Masih bergantung sepenuhnya pada setState() manual di setiap fungsi.	Sudah mencoba ValueNotifier, namun sinkronisasi data masih sering tertinggal atau error.	Implementasi <b>Reactive</b> sukses menggunakan ValueListenableBuilder; UI terupdate otomatis tanpa setState manual.	25%
<b>Task 3: JSON Persistence (HOTS)</b>	Data hilang saat aplikasi di-restart. Tidak ada upaya implementasi JSON.	Berhasil menyimpan data ke Shared Preferences, namun hanya tipe data String sederhana (bukan List Object).	Sukses melakukan <b>Serialization</b> (Object ↔ JSON); seluruh daftar logbook tersimpan permanen secara utuh.	25%



## Rubrik Penilaian Log LLM (Integritas)

Tabel 2.3: Rubrik Penilaian Integritas Log LLM.

Kriteria	Skor 1	Skor 3	Skor 5
<b>Kejujuran</b>	Mengaku tidak pakai AI padahal kode sangat kompleks.	Mencantumkan prompt tapi jawaban AI disalin bulat-bulat.	Terbuka soal bantuan AI dan mencantumkan prompt dengan jelas.
<b>Analisa (Fact Check)</b>	Tidak ada pengecekan ulang.	Mengecek apakah kode jalan atau tidak saja.	Mampu menemukan celah atau kelemahan dari saran AI.

<b>Kreativitas (Twist)</b>	Kode 100% sama dengan saran AI.	Mengubah nama variabel saja.	Mengintegrasikan saran AI ke dalam struktur SRP yang sudah dibuat.
--------------------------------	---------------------------------	------------------------------	--

## Referensi