LAPORAN WORKSHOP PEMROGRAMAN PERANGKAT BERGERAK

WEEK 6
Notes App

Dosen Pengampu : Prasetyo Wibowo S.ST., M.Kom

Disusun oleh :

Nama : Muhammad Daffa Erfiansyah
NRP : 3123500006
Kelas : 2 D3 IT A

Program Studi Teknik Informatika
Departemen Teknik
Informatika dan Komputer
Politeknik Elektronika Negeri Surabaya

2024/2025

A. Dasar Teori

Flutter menyediakan berbagai widget untuk membangun tampilan antarmuka yang fleksibel dan responsif. Layout dalam Flutter didasarkan pada widget seperti Container, Row, Column, Stack, dan ListView, yang memungkinkan pengaturan elemen secara vertikal, horizontal, atau bertumpuk. Widget Expanded dan Flexible digunakan untuk mengatur proporsi ukuran elemen dalam suatu tata letak. Selain itu, penggunaan Padding, Margin, dan Align membantu dalam mengatur posisi serta jarak antar elemen agar tampilan lebih rapi dan estetis. Flutter menggunakan sistem rendering berbasis widget yang memungkinkan pembaruan UI secara efisien. Dengan konsep widget tree, setiap elemen UI dalam Flutter direpresentasikan sebagai widget yang dapat bersarang dan diperbarui secara dinamis. Penggunaan MediaQuery dan LayoutBuilder membantu dalam membuat tampilan yang adaptif terhadap berbagai ukuran layar. Oleh karena itu, pemahaman tentang widget dan cara mengaturnya dalam layout sangat penting dalam membangun aplikasi Flutter yang responsif dan menarik.

B. Pratikum

1. Inisialisasi Database

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'package:note_app/models/note.dart';

class NoteDatabase {
  static final NoteDatabase instance = NoteDatabase._init();
  NoteDatabase._init();

  static Database? _database;

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB('notes.db');
    return _database!;
  }

  Future<Database> _initDB(String filepath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filepath);

    return await openDatabase(path, version: 1, onCreate: _createDB);
  }

  Future _createDB(Database db, int version) async {
    const sql = '''CREATE TABLE $tableNotes (
      ${NoteFields.id} INTEGER PRIMARY KEY AUTOINCREMENT,
      ${NoteFields.isImportant} BOOlEAN NOT NULL,
      ${NoteFields.number} INTEGER NOT NULL,
      ${NoteFields.title} TEXT NOT NULL,
      ${NoteFields.description} TEXT NOT NULL,
      ${NoteFields.time} TIMESTAMP NOT NULL
    )
    ''';
    await db.execute(sql);
  }

  //CRUD
  Future<Note> create(Note note) async {
    final db = await instance.database;
    final id = await db.insert(tableNotes, note.toJson());
    return note.copy(id: id);
  }
```

```dart
Future<Note> getNoteById(int id) async {
    final db = await instance.database;
    final result = await db.query(
      tableNotes,
      columns: NoteFields.values,
      where: '${NoteFields.id} = ?',
      whereArgs: [id],
    );

    if (result.isNotEmpty) {
      return Note.fromJson(result.first);
    } else {
      throw Exception('ID $id not found');
    }
  }

  Future<List<Note>> getAllNotes() async {
    final db = await instance.database;
    final orderBy = '${NoteFields.number} ASC';
    final result = await db.query(tableNotes, orderBy: orderBy);
    return result.map((json) => Note.fromJson(json)).toList();
  }

  Future<int> updateNote(Note note) async {
    final db = await instance.database;
    return await db.update(
      tableNotes,
      note.toJson(),
      where: '${NoteFields.id} = ?',
      whereArgs: [note.id],
    );
  }

  Future<int> deleteNoteById(int id) async {
    final db = await instance.database;
    return await db.delete(
      tableNotes,
      where: '${NoteFields.id} = ?',
      whereArgs: [id],
    );
  }
}
```

2. Models Database

```dart
const String tableNotes = 'notes';

class NoteFields {
  static final List<String> values = [
    id,
    isImportant,
    number,
    title,
    description,
    time,
  ];

  static const String id = '_id';
  static const String isImportant = 'isImportant';
  static const String number = 'number';
  static const String title = 'title';
  static const String description = 'description';
  static const String time = 'time';
}

class Note {
  final int? id;
  final bool isImportant;
  final int number;
  final String title;
  final String description;
  final DateTime createdTime;

  Note({
    this.id,
    required this.isImportant,
    required this.number,
    required this.title,
    required this.description,
    required this.createdTime,
  });
```

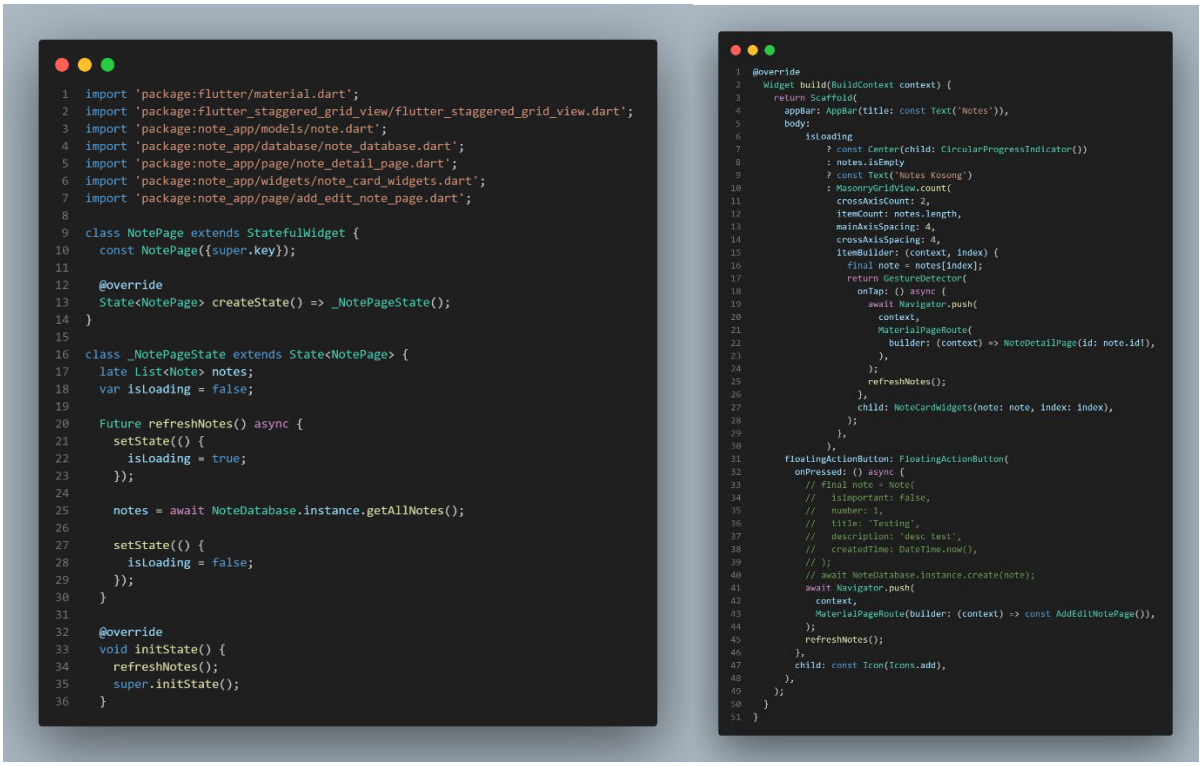
```dart
Note copy({
    int? id,
    bool? isImportant,
    int? number,
    String? title,
    String? description,
    DateTime? createdTime,
  }) => Note(
    id: id ?? this.id,
    isImportant: isImportant ?? this.isImportant,
    number: number ?? this.number,
    title: title ?? this.title,
    description: description ?? this.description,
    createdTime: createdTime ?? this.createdTime,
  );

  static Note fromJson(Map<String, Object?> json) => Note(
    id: json[NoteFields.id] as int?,
    isImportant: json[NoteFields.isImportant] == 1,
    number: json[NoteFields.number] as int,
    title: json[NoteFields.title] as String,
    description: json[NoteFields.description] as String,
    createdTime: DateTime.parse(json[NoteFields.time] as String),
  );

  Map<String, Object?> toJson() => {
    NoteFields.id: id,
    NoteFields.isImportant: isImportant ? 1 : 0,
    NoteFields.number: number,
    NoteFields.title: title,
    NoteFields.description: description,
    NoteFields.time: createdTime.toIso8601String(),
  };
}
```

Analisa:
Class NoteDatabase menerapkan pola singleton dengan lazy initialization untuk memastikan satu instance database di seluruh aplikasi. Database diinisialisasi dengan _initDB untuk penyimpanan dan _createDB untuk skema. Metode CRUD seperti create, getNoteById, getAllNotes, updateNote, dan deleteNoteById tersedia, serta metode tambahan close dan getAllNotesSortedByImportance. Model Note merepresentasikan catatan dengan field seperti id, isImportant, number, title, description, dan time. Model ini bersifat immutable dengan constructor wajib, serta menyediakan metode copy, fromJson, dan toJson untuk manipulasi data.

3. Note Page



```dart
import 'package:flutter/material.dart';
import 'package:flutter_staggered_grid_view/flutter_staggered_grid_view.dart';
import 'package:note_app/models/note.dart';
import 'package:note_app/database/note_database.dart';
import 'package:note_app/page/note_detail_page.dart';
import 'package:note_app/widgets/note_card_widgets.dart';
import 'package:note_app/page/add_edit_note_page.dart';

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

  @override
  State<NotePage> createState() => _NotePageState();
}

class _NotePageState extends State<NotePage> {
  late List<Note> notes;
  var isLoading = false;

  Future refreshNotes() async {
    setState(() {
      isLoading = true;
    });

    notes = await NoteDatabase.instance.getAllNotes();

    setState(() {
      isLoading = false;
    });
  }

  @override
  void initState() {
    refreshNotes();
    super.initState();
  }
```

```dart
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Notes')),
      body:
          isLoading
              ? const Center(child: CircularProgressIndicator())
              : notes.isEmpty
              ? const Text('Notes Kosong')
              : MasonryGridView.count(
                crossAxisCount: 2,
                itemCount: notes.length,
                mainAxisSpacing: 4,
                crossAxisSpacing: 4,
                itemBuilder: (context, index) {
                  final note = notes[index];
                  return GestureDetector(
                    onTap: () async {
                      await Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => NoteDetailPage(id: note.id!),
                        ),
                      );
                      refreshNotes();
                    },
                    child: NoteCardWidgets(note: note, index: index),
                  );
                },
              ),
      floatingActionButton: FloatingActionButton(
        onPressed: () async {
          // final note = Note(
          //   isimportant: false,
          //   number: 1,
          //   title: 'Testing',
          //   description: 'desc test',
          //   createdTime: DateTime.now(),
          // );
          // await NoteDatabase.instance.create(note);
          await Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => const AddEditNotePage()),
          );
          refreshNotes();
        },
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Analisa:
Class NotePage menampilkan daftar catatan dalam grid dinamis menggunakan MasonryGridView. Fitur yang tersedia mencakup pencarian, animasi transisi, indikator loading, tampilan kosong saat tidak ada catatan, serta tombol tambah catatan. State dikelola dengan baik menggunakan AnimationController, dengan pembersihan resource di dispose(). Class NoteCardWidgets menampilkan setiap catatan dalam kartu dengan judul, deskripsi, waktu, dan indikator kepentingan. Tampilan diperindah dengan warna latar dari _lightColors, variasi tinggi kartu melalui getMinHeight(), serta format tanggal yang mudah dibaca menggunakan DateFormat.

4. Add & Edit Page

```dart
import 'package:flutter/material.dart';
import 'package:note_app/database/note_database.dart';
import 'package:note_app/models/note.dart';
import 'package:note_app/widgets/note_form_widgets.dart';

class AddEditNotePage extends StatefulWidget {
  final Note? note;
  const AddEditNotePage({super.key, this.note});

  @override
  State<AddEditNotePage> createState() => _AddEditNotePageState();
}

class _AddEditNotePageState extends State<AddEditNotePage> {
  late bool isImportant;
  late int number;
  late String title;
  late String description;
  final _formKey = GlobalKey<FormState>();
  var isUpdateForm = false;

  @override
  void initState() {
    super.initState();
    isImportant = widget.note?.isImportant ?? false;
    number = widget.note?.number ?? 0;
    title = widget.note?.title ?? '';
    description = widget.note?.description ?? '';
    isUpdateForm = widget.note != null;
  }
```

```dart
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Add'), actions: [buildButtonSave()]),
      body: Form(
        key: _formKey,
        child: NoteFormWidget(
          isImportant: isImportant,
          number: number,
          title: title,
          description: description,
          onChangeIsImportant: (value) {
            setState(() {
              isImportant = value;
            });
          },
          onChangeNumber: (value) {
            setState(() {
              number = value;
            });
          },
          onChangeTitle: (value) {
            title = value;
          },
          onChangeDescription: (value) {
            description = value;
          },
        ),
      ),
    );
  }
```

```dart
buildButtonSave() {
    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 8, horizontal: 12),
      child: ElevatedButton(
        onPressed: () async {
          final isValid = _formKey.currentState!.validate();
          if (isValid) {
            if (isUpdateForm) {
              //Update Data
              await updateNote();
            } else {
              //Tambah Data
              await addNote();
            }
            //Tutup halaman
            Navigator.pop(context);
          }
        },
        child: const Text('Save'),
      ),
    );
  }

  Future addNote() async {
    final note = Note(
      isImportant: isImportant,
      number: number,
      title: title,
      description: description,
      createdTime: DateTime.now(),
    );
    await NoteDatabase.instance.create(note);
  }

  Future updateNote() async {
    final updateNote = widget.note?.copy(
      isImportant: isImportant,
      number: number,
      title: title,
      description: description,
    );
    await NoteDatabase.instance.updateNote(updateNote!);
  }
}
```

Form.dart

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

class NoteFormWidget extends StatelessWidget {
  const NoteFormWidget({
    Key? key,
    required this.isImportant,
    required this.number,
    required this.title,
    required this.description,
    required this.onChangeIsImportant,
    required this.onChangeNumber,
    required this.onChangeTitle,
    required this.onChangeDescription,
  }) : super(key: key);

  final bool isImportant;
  final int number;
  final String title;
  final String description;
  final ValueChanged<bool> onChangeIsImportant;
  final ValueChanged<int> onChangeNumber;
  final ValueChanged<String> onChangeTitle;
  final ValueChanged<String> onChangeDescription;

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            Row(
              children: [
                Switch(value: isImportant, onChanged: onChangeIsImportant),
                Expanded(
                  child: Slider(
                    value: number.toDouble(),
                    min: 0,
                    max: 5,
                    divisions: 5,
                    onChanged: (value) => onChangeNumber(value.toInt()),
                  ),
                ),
              ],
            ),
            buildTitleField(),
            const SizedBox(height: 8),
            buildDescriptionField(),
          ],
        ),
      ),
    );
  }
```

```dart
buildTitleField() {
    return TextFormField(
      maxLines: 1,
      initialValue: title,
      style: const TextStyle(
        // color: Colors.white70,
        fontWeight: FontWeight.bold,
        fontSize: 24,
      ),
      decoration: const InputDecoration(
        border: InputBorder.none,
        hintText: 'Title',
        hintStyle: TextStyle(color: Colors.grey),
      ),
      validator: (title) {
        return title != null && title.isEmpty
            ? 'The title cannot be empty'
            : null;
      },
      onChanged: onChangeTitle,
    );
  }

  buildDescriptionField() {
    return TextFormField(
      maxLines: null,
      initialValue: description,
      style: const TextStyle(
        // color: Colors.white70,
        fontWeight: FontWeight.bold,
        fontSize: 24,
      ),
      decoration: const InputDecoration(
        border: InputBorder.none,
        hintText: 'Type something...',
        hintStyle: TextStyle(color: Colors.grey),
      ),
      validator: (desc) {
        return desc != null && desc.isEmpty
            ? 'The description cannot be empty'
            : null;
      },
      onChanged: onChangeDescription,
    );
  }
}
```

Analisa:
Class AddEditNotePage berfungsi sebagai halaman formulir untuk menambah atau mengedit catatan, dengan parameter note yang opsional. State dikelola untuk field catatan menggunakan Form dan GlobalKey untuk validasi. Metode addNote dan updateNote dipisahkan untuk menangani penyimpanan dan pembaruan catatan, serta Snackbar digunakan untuk menampilkan notifikasi keberhasilan atau kegagalan. NoteFormWidget memisahkan UI formulir dari logika dengan menerima nilai dan callback. Formulir mencakup switch untuk menandai catatan penting, slider prioritas (0-5), serta field judul dan deskripsi. Desainnya diperhatikan dengan Card, border radius, padding yang konsisten, serta fokus border ungu saat field aktif. Validasi memastikan judul dan deskripsi tidak kosong.

5. DetailPage

```dart
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:note_app/models/note.dart';
import 'package:note_app/database/note_database.dart';
import 'package:note_app/page/add_edit_note_page.dart';

class NoteDetailPage extends StatefulWidget {
  final int id;
  const NoteDetailPage({super.key, required this.id});

  @override
  State<NoteDetailPage> createState() => _NoteDetailPageState();
}

class _NoteDetailPageState extends State<NoteDetailPage> {
  late Note note;
  var isLoading = false;

  Future refreshNote() async {
    setState(() {
      isLoading = true;
    });

    note = await NoteDatabase.instance.getNoteById(widget.id);

    setState(() {
      isLoading = false;
    });
  }

  @override
  void initState() {
    super.initState();
    refreshNote();
  }
```

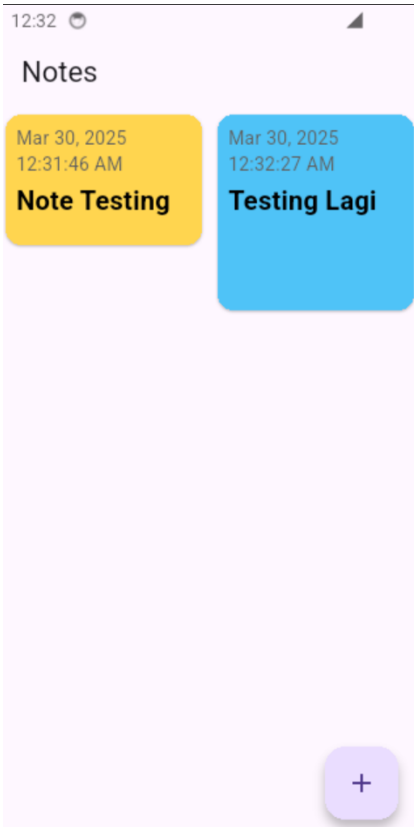
```dart
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Detail Page'),
        actions: [editButton(), deleteButton()],
      ),
      body:
          isLoading
              ? const Center(child: CircularProgressIndicator())
              : ListView(
                padding: const EdgeInsets.all(8),
                children: [
                  Text(
                    note.title,
                    style: const TextStyle(
                      // color: Colors.white,
                      fontSize: 22,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 8),
                  Text(
                    DateFormat.yMMMd().format(note.createdTime),
                    // style: const TextStyle(color: Colors.white38),
                  ),
                  const SizedBox(height: 8),
                  Text(
                    note.description,
                    style: const TextStyle(
                      // color: Colors.white70,
                      fontSize: 18,
                    ),
                  ),
                ],
              ),
    );
  }
```

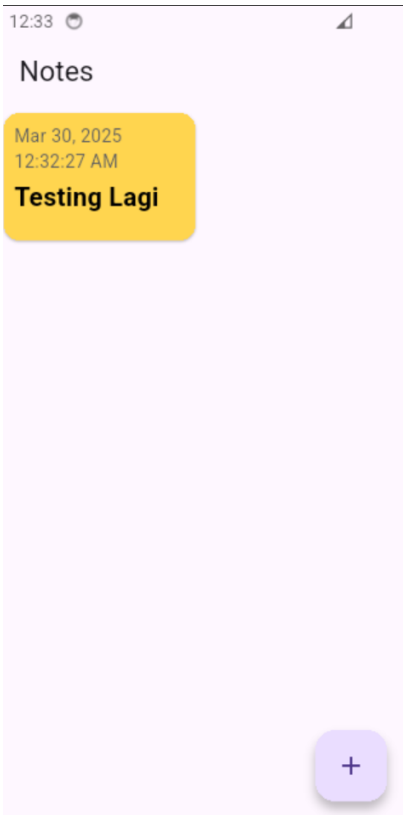
```dart
Widget editButton() {
    return IconButton(
      icon: const Icon(Icons.edit_outlined),
      onPressed: () async {
        if (isLoading) return;
        await Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => AddEditNotePage(note: note)),
        );
        refreshNote();
      },
    );
  }

  Widget deleteButton() {
    return IconButton(
      icon: const Icon(Icons.delete),
      onPressed: () async {
        if (isLoading) return;
        await NoteDatabase.instance.deleteNoteById(widget.id);
        Navigator.pop(context);
      },
    );
  }
}
```

Analisa :
Kelas NoteDetailPage dirancang untuk menampilkan informasi lengkap dari sebuah catatan yang dipilih berdasarkan ID. Halaman ini menggunakan StatefulWidget karena perlu memuat data catatan secara asinkron dari database. Metode refreshNote() mengambil data catatan berdasarkan ID dan mengelola state loading dengan baik.

6. Output :





7. Kesimpulan

   Dalam praktikum ini untuk menampilkan informasi tempat wisata dapat dilakukan dengan pendekatan yang responsif dan estetis. Dengan memanfaatkan ListView, Card, dan Stack, aplikasi mampu menyajikan daftar tempat wisata serta detail informasi dengan tampilan yang menarik dan mudah diakses oleh pengguna. Selain itu, penggunaan Navigator.push memungkinkan navigasi antar halaman dengan lancar, sementara elemen visual seperti ClipRRect, BoxShadow, dan LinearGradient memberikan tampilan yang lebih profesional. Praktikum ini juga menunjukkan pentingnya pengorganisasian kode dengan metode terpisah, seperti _buildListItem dan _buildInfoCard, untuk meningkatkan keterbacaan dan pemeliharaan kode.