GUIDED & UNGUIDED
PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIII

NETWORKING

Disusun Oleh :

Nama: Arwin Nabiel Arrofif

NIM: 2211104057

Kelas: SE 06 02

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

NETWORKING

NETWORKING/STATE MANAGEMENT

State management dalam Flutter adalah proses mengelola state atau status dari aplikasi, yaitu data atau informasi yang dapat berubah sepanjang siklus hidup aplikasi. State ini mencakup segala hal yang memengaruhi tampilan antarmuka pengguna (UI), seperti input pengguna, data dari API, dan status internal widget. Ketika aplikasi semakin kompleks dibuat, maka pasti akan ada saatnya dimana harus dibagikan state aplikasi ke berbagai halaman yang ada.

Flutter adalah deklaratif, sehingga Flutter membangun user interface berdasarkan state saat ini. Dengan menggunakan state management, dapat dilakukan sentralisasi semua state dari berbagai macam UI Control untuk mengendalikan aliran data lintas aplikasi.

State management penting karena aplikasi Flutter sering kali terdiri dari banyak *widget* yang saling terkait. Dengan mengelola state dengan baik, kita dapat memastikan :

- Sinkronisasi UI dan data, karena selalu mencerminkan data terkini.
- Organisasi kode yang baik untuk mempermudah pengembangan dan pemeliharaan.
- Pengurangan bug, karena state yang dikelola dengan benar mengurangi kemungkinan terjadinya bug.

Jenis State dalam Flutter

1. Ephemeral State (State Lokal)

State ini hanya relevan untuk widget tertentu dan tidak dibagikan ke widget lain. Contohnya adalah state untuk *TextField* atau *Checkbox*. Dan kita dapat menggunakan *StatefulWidget* untuk mengelola *ephemeral state*. Pendekatannya state menaganement-nya ada dua, yakni *StatefulWidget* (untuk ephemeral state) dan *InheritedWidget* (untuk berbagai state antar widget).

2. App State (State Global)

State ini digunakan di berbagai widget dalam aplikasi. Contohnya adalah informasi pengguna yang masuk, data keranjang belanja, atau tema aplikasi. App state biasanya membutuhkan pendekatan state management yang lebih kompleks. *Package/library* pendukung Flutter memiliki berbagai framework atau *package* untuk *state management*, seperti :

A. Provider

Provider adalah library state management yang didukung resmi oleh tim Flutter. Provider memanfaatkan kemampuan bawaan Flutter seperti *InheritedWidget*, tetapi dengan cara yang lebih sederhana dan efisien.

B. BloC/Cubit

Bloc (Business Logic Component) adalah pendekatan state management berbasis pola ***stream***. Bloc memisahkan ***business logic*** dari UI, sehingga cocok untuk aplikasi yang besar dan kompleks.

C. Riverpod

Riverpod adalah framework state management modern yang dirancang sebagai pengganti atau alternatif untuk Provider. Riverpod lebih fleksibel dan mengatasi beberapa keterbatasan Provider.

D. GetX

GetX adalah framework Flutter serbaguna yang menyediakan solusi lengkap untuk ***state management***, ***routing***, dan ***dependency injection***. GetX dirancang untuk meminimalkan *boilerplate code*, meningkatkan efisiensi, dan mempermudah pengembangan aplikasi Flutter, terutama yang memerlukan reaktivitas tinggi.

Berikut cara instalasi GetX :

1) Tambahkan GetX ke dalam proyek Flutter melalui pubspec.yaml :

```
dependencies:
flutter:
sdk: flutter
get: ^4.6.5
```

2) Konfigurasi dasar

Untuk menggunakan GetX, ubah root aplikasi dengan mengganti MaterialApp menjadi GetMaterialApp :

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


      void main() {
      runApp(MyApp());
      }
      class MyApp extends StatelessWidget {
        @override
        Widget build(BuildContext context) {
      return GetMaterialApp(       title:
      'Praktikum 13 - GetX',       home:
      HomePage(),
          );
        }
}
```

3) State Management dengan GetX

a. Membuat Controller

Buat class controller untuk mengelola state. Misalnya, untuk counter sederhana :

```
import 'package:get/get.dart';


class CounterController extends GetxController {   var
count = 0.obs; // State yang reaktif
```

```
  void increment() => count++;
}
```

b. Menggunakan Controller di UI

- Tambahkan controller ke dalam widget menggunakan Get.put() untuk dependency injection.
- Gunakan Obx untuk memantau perubahan state.

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


class HomePage extends StatelessWidget {
  final       CounterController       controller       =
Get.put(CounterController());


  @override
  Widget build(BuildContext context) {
return Scaffold(
      appBar:     AppBar(title:     Text('GetX     State
Management')),
body: Center(
        child: Obx(() => Text(
              'Counter: ${controller.count}',
style: TextStyle(fontSize: 25),
            )),
),
      floatingActionButton:
FloatingActionButton(         onPressed:
controller.increment,         child:
Icon(Icons.add),
      ),
    );
  }
}
```

4) Routing dengan GetX

a. Definisikan Route

Gunakan *GetPage* pada main.dart untuk mendefinisikan rute aplikasi :

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

void main() {   runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
return GetMaterialApp(       initialRoute: '/',
getPages: [
        GetPage(name: '/', page: () => HomePage()),
        GetPage(name: '/details', page: () =>
DetailsPage()),
      ],
    );
  }
}
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
return Scaffold(       body: Center(
        child: ElevatedButton(
onPressed: () {
            Get.toNamed('/details'); // Navigasi ke halaman
lain           },
          child: Text('Go to Details'),
        ),
      ),
    );
  }
}
```

```
      class DetailsPage extends StatelessWidget {
        @override
        Widget build(BuildContext context)
      {     return Scaffold(       body:
      Center(
              child: Text('Details Page'),
            ),
          );
        }
}
```

b. Navigasi

- Get.to() : Navigasi ke halaman baru.
- Get.back() : Kembali ke halaman sebelumnya.
- Get.off() : Menghapus semua halaman sebelumnya.
- Get.offAll() : Menghapus semua halaman dalam stack.

5) Dependency Injection dengan GetX

1. Injeksi Sederhana

   Gunakan Get.put() untuk membuat instance controller yang tersedia di mana saja :

```
final CounterController controller =
Get.put(CounterController());
```

2. Lazy Loading

   Gunakan Get.lazyPut() jika ingin membuat instance hanya saat dibutuhkan :

```
Get.lazyPut(() => CounterController());
```

3. Mengambil Instance

   Ambil instance di mana saja dalam aplikasi :

```
final CounterController controller = Get.find();
```

6) Snackbar

```
Get.snackbar('Title', 'This is a snackbar');
```

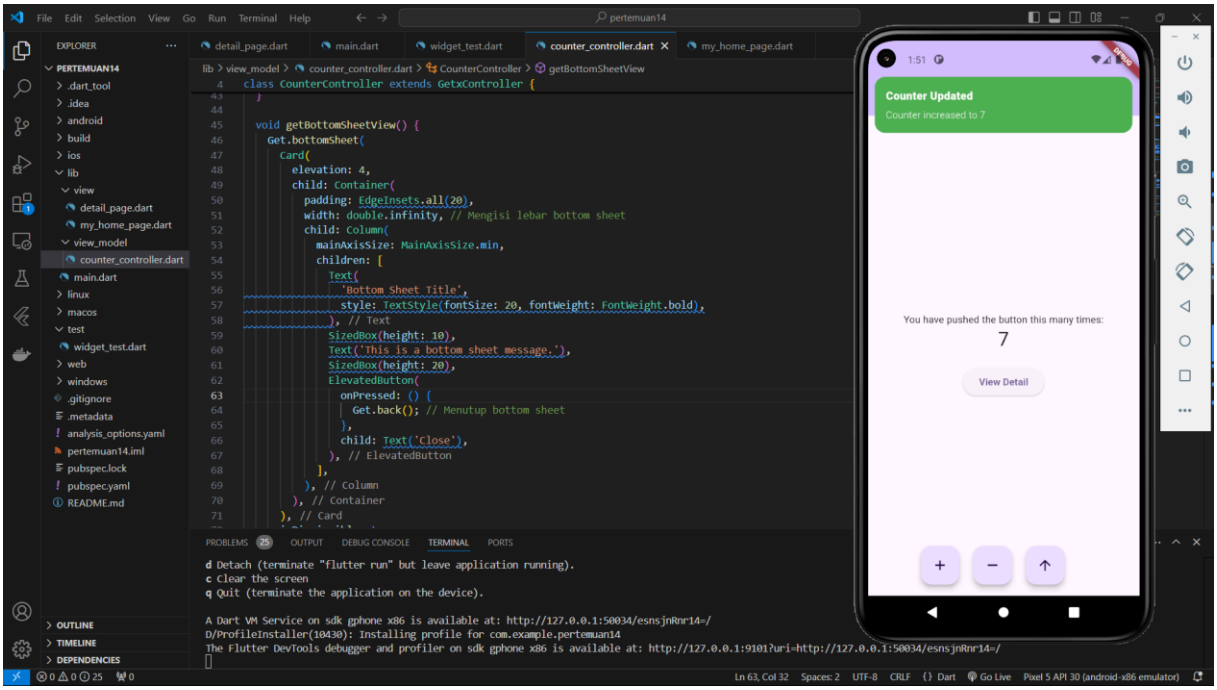
7) Dialog

```
Get.defaultDialog(   title: 'Dialog Title',
middleText: 'This is a dialog',
);
```

8) BottomSheet

```
Get.bottomSheet(
Container(
    child: Text('This is a bottom sheet'),
  ),
);
```

TUGAS MANDIRI (GUIDED)



TUGAS MANDIRI (UNGUIDED)

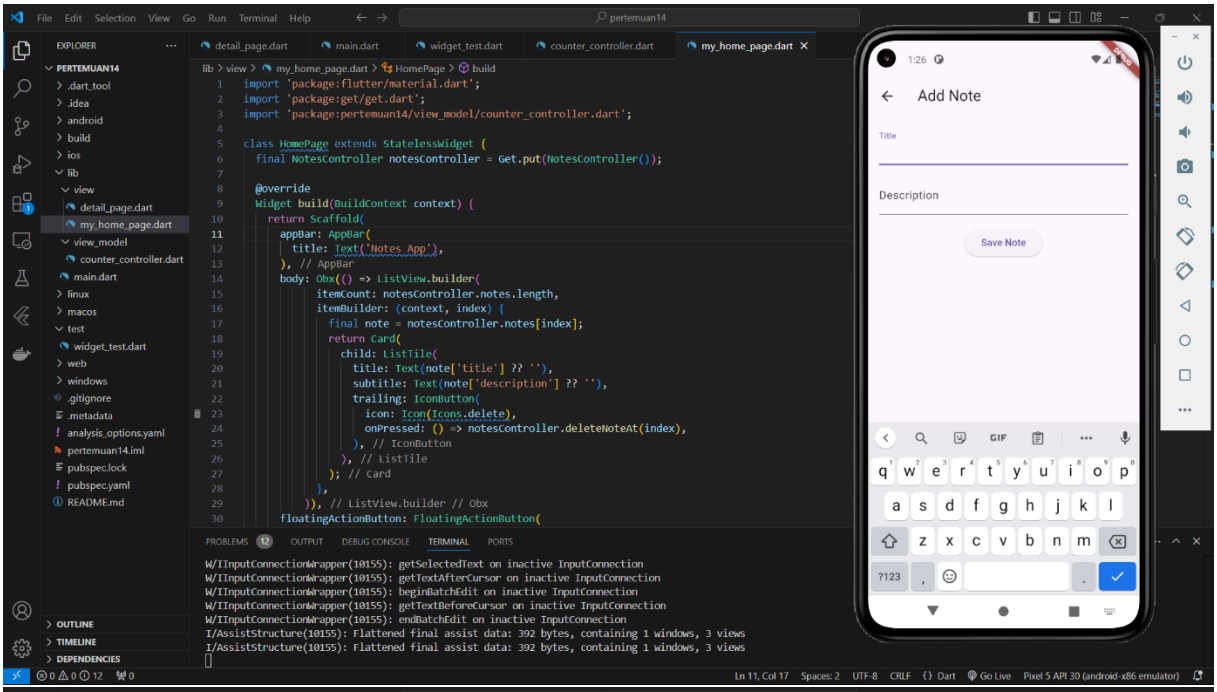
SOAL

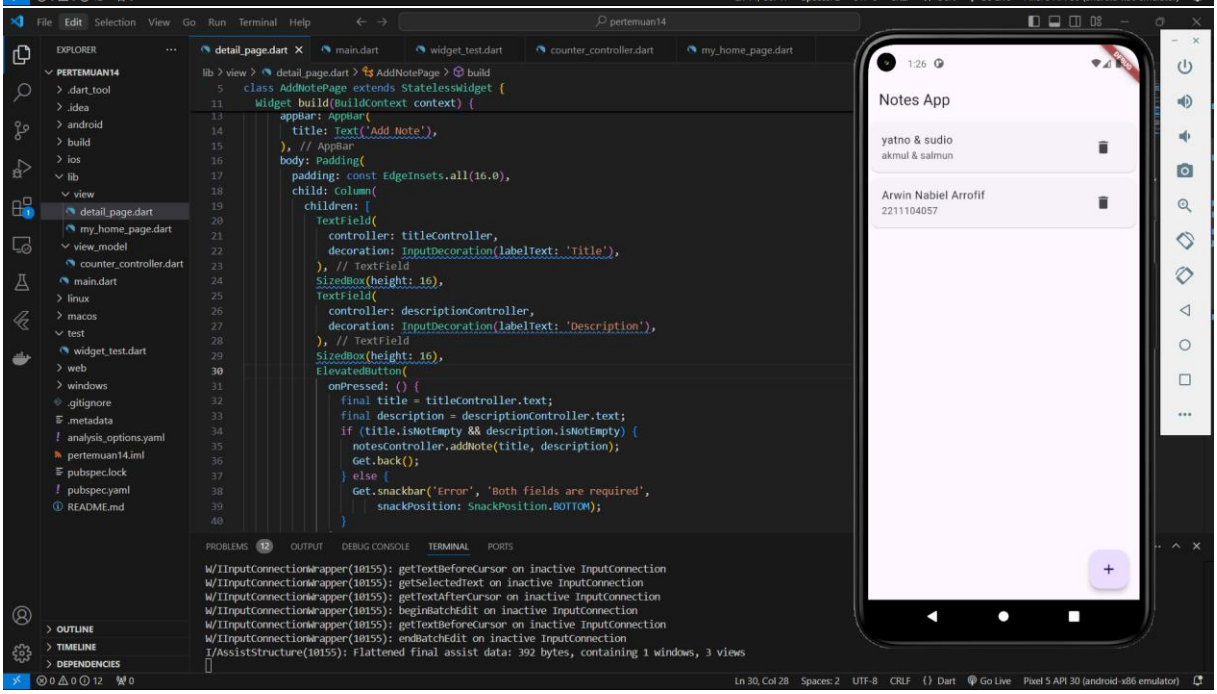
Buatlah Aplikasi Catatan Sederhana menggunakan GetX, dengan ketentuan sebagai berikut :

1. Halaman utama atau Homepage untuk menampilkan daftar catatan yang telah ditambahkan. Setiap catatan terdiri dari judul dan deskripsi singkat, serta terdapat tombol untuk menghapus catatan dari daftar.
2. Halaman kedua untuk menambah catatan baru, berisi : form untuk memasukkan judul dan deskripsi catatan, serta tombol untuk menyimpan catatan ke daftar (Homepage).
3. Menggunakan getx controller.
4. Menggunakan getx routing untuk navigasi halaman.

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

JAWAB


my_home_page.dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:pertemuan14/view_model/counter_controller.dart';
class HomePage extends StatelessWidget {
final NotesController notesController = Get.put(NotesController());
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text('Notes App'),
), // AppBar
body: Obx(() => ListView.builder(
itemCount: notesController.notes.length,
itemBuilder: (context, index) {
final note = notesController.notes[index];
return Card(
child: ListTile(
title: Text(note['title'] ?? ''),
subtitle: Text(note['description'] ?? ''),
trailing: IconButton(
icon: Icon(Icons.delete),
onPressed: () => notesController.deleteNoteAt(index),
), // IconButton
), // ListTile
); // Card
},
)), // ListView.builder // Obx
floatingActionButton: FloatingActionButton(
Add Note
Title
Description
Save Note
W/IInputConnectionWrapper(10155): getSelectedText on inactive InputConnection
W/IInputConnectionWrapper(10155): getTextAfterCursor on inactive InputConnection
W/IInputConnectionWrapper(10155): beginBatchEdit on inactive InputConnection
W/IInputConnectionWrapper(10155): getTextBeforeCursor on inactive InputConnection
W/IInputConnectionWrapper(10155): endBatchEdit on inactive InputConnection
I/AssistStructure(10155): Flattened final assist data: 392 bytes, containing 1 windows, 3 views
I/AssistStructure(10155): Flattened final assist data: 392 bytes, containing 1 windows, 3 views


detail_page.dart
class AddNotePage extends StatelessWidget {
Widget build(BuildContext context) {
appBar: AppBar(
title: Text('Add Note'),
), // AppBar
body: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
TextField(
controller: titleController,
decoration: InputDecoration(labelText: 'Title'),
), // TextField
SizedBox(height: 16),
TextField(
controller: descriptionController,
decoration: InputDecoration(labelText: 'Description'),
), // TextField
SizedBox(height: 16),
ElevatedButton(
onPressed: () {
final title = titleController.text;
final description = descriptionController.text;
if (title.isNotEmpty && description.isNotEmpty) {
notesController.addNote(title, description);
Get.back();
} else {
Get.snackbar('Error', 'Both fields are required',
snackPosition: SnackPosition.BOTTOM);
}
Notes App
yatno & sudio
akmul & salmun
Arwin Nabiel Arrofif
2211104057
W/IInputConnectionWrapper(10155): getTextBeforeCursor on inactive InputConnection
W/IInputConnectionWrapper(10155): getSelectedText on inactive InputConnection
W/IInputConnectionWrapper(10155): getTextAfterCursor on inactive InputConnection
W/IInputConnectionWrapper(10155): beginBatchEdit on inactive InputConnection
W/IInputConnectionWrapper(10155): getTextBeforeCursor on inactive InputConnection
W/IInputConnectionWrapper(10155): endBatchEdit on inactive InputConnection
I/AssistStructure(10155): Flattened final assist data: 392 bytes, containing 1 windows, 3 views

Deskripsi Program:

Aplikasi Catatan Sederhana, aplikasi berbasis Flutter yang memanfaatkan GetX untuk pengelolaan state dan navigasi antar halaman. Aplikasi ini dirancang untuk membantu pengguna mencatat dan mengelola daftar catatan dengan mudah. Berikut fitur-fitur utama yang tersedia

1. **Halaman Utama**
    - Menampilkan daftar catatan yang telah ditambahkan pengguna.
    - Setiap catatan mencakup judul dan deskripsi singkat.
    - Pengguna dapat menghapus catatan dari daftar menggunakan tombol hapus yang tersedia.
2. **Halaman Tambah Catatan**
    - Berisi form untuk menambahkan catatan baru dengan input berupa judul dan deskripsi.
    - Terdapat tombol Simpan Catatan untuk menambahkan catatan ke daftar di halaman utama.
3. **Pengelolaan Data dengan GetX Controller**
    - Aplikasi menggunakan NotesController untuk mengelola daftar catatan secara reaktif dengan RxList, sehingga perubahan data langsung tercermin di tampilan.
4. **Navigasi Halaman dengan GetX Routing:**
    - Perpindahan antara halaman utama dan halaman tambah catatan dilakukan dengan mudah menggunakan fitur GetX Routing.

Note: Kodingan sudah di apload di folder unguided.