TUGAS PENDAHULUAN
PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIII

NETWORKING

Disusun Oleh :

TEGAR KANG

AGENG GILANG

2311104018

SE 07 01

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

TUGAS PENDAHULUAN

SOAL

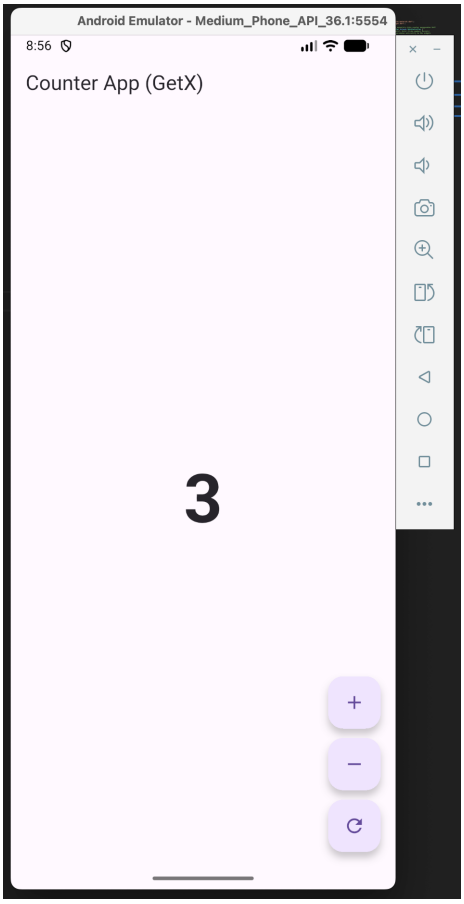
1. Apa yang dimaksud dengan state management pada Flutter?
2. Sebut dan jelaskan komponen-komponen yang ada di dalam GetX.
3. Lengkapilah code di bawah ini, dan tampilkan hasil outputnya serta jelaskan.

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

/// Controller untuk mengelola state counter menggunakan GetX
class CounterController extends GetxController {
  // 1) Variabel reaktif: RxInt (0.obs memberi Rx<int>)
  //    Keuntungan: perubahan auto-notify ke Obx widgets
  var counter = 0.obs;

  // 2) Fungsi untuk menambah nilai counter
  void increment() {
    // operator ++ di RxInt otomatis memodifikasi value
    counter++;
    // alternatif: counter.value = counter.value + 1;
  }

  // 3) Fungsi untuk mengurangi (opsional)
  void decrement() {
    if (counter.value > 0) counter--;
  }

  // 4) Fungsi untuk mereset nilai counter
  void reset() {
    counter.value = 0;
  }

  // Lifecycle hooks
  @override
  void onInit() {
    super.onInit();
    // contoh: inisialisasi data, subscribe ke service, atau fetch awal
    print("CounterController onInit - controller dibuat");
  }

  @override
  void onReady() {
    super.onReady();
    // siap setelah widget muncul di layar
    print("CounterController onReady - widget siap");
  }

  @override
  void onClose() {
```

```
    // disposal: tutup stream atau cleanup
    print("CounterController onClose - controller dihapus");
    super.onClose();
  }
}

class HomePage extends StatelessWidget {
  // Put controller ke dependency management GetX.
  // Get.put membuat instance langsung dan tersedia via Get.find()
  final CounterController controller = Get.put(CounterController());

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Counter App (GetX)")),
      body: Center(
        // Obx mendengarkan perubahan pada Rx value di dalam closure
        child: Obx(() {
          // controller.counter.value berubah -> Obx rebuild
          return Text(
            "${controller.counter.value}",
            style: TextStyle(fontSize: 72, fontWeight: FontWeight.bold),
          );
        }),
      ),
      floatingActionButton: Column(
        mainAxisAlignment: MainAxisAlignment.end,
        children: [
          // Tambah
          FloatingActionButton(
            heroTag: "add",
            onPressed: () {
              controller.increment();
            },
            child: Icon(Icons.add),
          ),
          SizedBox(height: 10),
          // Kurangi
          FloatingActionButton(
            heroTag: "minus",
            onPressed: () {
              controller.decrement();
            },
            child: Icon(Icons.remove),
          ),
          SizedBox(height: 10),
          // Reset
          FloatingActionButton(
            heroTag: "reset",
            onPressed: () {
              // Contoh penggunaan dialog konfirmasi
              Get.defaultDialog(
                title: "Reset",
                middleText: "Reset counter ke 0?",
```

```
                onConfirm: () {
                  controller.reset();
                  Get.back(); // tutup dialog
                },
                onCancel: () {},
              );
            },
            child: Icon(Icons.refresh),
          ),
        ],
      ),
    );
  }
}

void main() {
  // Untuk menggunakan Get features (snackbar, routing), gunakan GetMaterialApp
  runApp(GetMaterialApp(
    debugShowCheckedModeBanner: false,
    home: HomePage(),
  ));
}
```

Screenshoot Output



Deskripsi Program

Kode Flutter di atas merupakan implementasi aplikasi Counter App menggunakan GetX sebagai state management. Di dalamnya terdapat sebuah controller bernama CounterController yang mengelola nilai counter dengan menggunakan variabel reaktif `RxInt` (`0.obs`). Controller ini memiliki tiga fungsi utama: `increment()` untuk menambah nilai counter, `decrement()` untuk mengurangi nilai selama nilainya lebih besar dari nol, dan `reset()` untuk mengembalikan nilai counter ke nol. Selain itu, controller juga memanfaatkan lifecycle hooks seperti `onInit()`, `onReady()`, dan `onClose()` untuk menampilkan log ketika controller dibuat, siap digunakan, dan dihancurkan.

UI aplikasi dibangun melalui widget HomePage (StatelessWidget) yang menempatkan instance `CounterController` menggunakan `Get.put()`, sehingga controller dapat diakses di mana saja melalui GetX. Pada bagian tampilan, nilai counter ditampilkan menggunakan widget `Obx`, yang berfungsi melakukan rebuild otomatis saat nilai reaktif `counter` berubah. Aplikasi menyediakan tiga tombol FloatingActionButton: tombol tambah untuk meningkatkan nilai counter, tombol minus untuk mengurangi nilai, dan tombol reset yang menampilkan dialog konfirmasi menggunakan `Get.defaultDialog()` sebelum mereset nilai. Terakhir, fungsi `main()` menjalankan aplikasi menggunakan `GetMaterialApp`, yang memungkinkan fitur-fitur seperti snackbar, dialog, dan routing khas GetX, serta menampilkan `HomePage` sebagai halaman utama aplikasi.

1) State management di Flutter

State = data yang menentukan tampilan/tingkah laku widget (mis. nilai counter, status login, daftar item).
State management adalah teknik/arsitektur untuk **menyimpan, mengubah, dan menyebarkan** perubahan state sehingga UI dan logic tetap sinkron.

Mengapa perlu?

- Memudahkan pemisahan UI dan logic.
- Mengurangi bug (mis. UI tidak ter-update).
- Memudahkan testing dan maintenance.
- Skalabilitas: dari aplikasi kecil ke besar.

Kategori pendekatan (ringkas + kapan dipakai)

1. **Local (setState)**
   - `setState()` di `StatefulWidget`.
   - Untuk state lokal, sederhana, sedikit state.
   - Kelemahan: sulit dibagikan antar widget banyak.
2. **InheritedWidget / InheritedModel**
   - Mekanisme bawaan Flutter untuk meneruskan data ke subtree.
   - Dipakai internal Flutter (Theme, MediaQuery).
   - Boilerplate banyak → biasanya dibungkus oleh libs.
3. **Provider (dari Flutter team)**
   - Menggunakan `ChangeNotifier` atau value/stream.
   - Ringan, idiomatik, cocok untuk sebagian besar aplikasi.
4. **BLoC / Cubit (rxdart)**
   - Pattern berbasis stream (events → states).
   - Bagus untuk aplikasi besar yang butuh testability dan pemisahan yang ketat.
5. **Redux**
   - Single source of truth, immutable state, action → reducer.
   - Cocok jika model aplikasi memang cocok dengan pola Redux.
6. **GetX / Riverpod / MobX**
   - GetX: ringkas, reaktif, plus routing & DI.
   - Riverpod: modern, testable, type-safe.
   - Pilihan tergantung preferensi, ukuran tim, dan kebutuhan.

Kriteria memilih

- Ukuran proyek & tim
- Kompleksitas data sharing
- Preferensi testability vs rapid development
- Perf/overhead & learning curve

2) GetX komponen & penjelasan mendetail (dengan contoh dan best practice)

GetX sering disebut “microframework” karena menggabungkan tiga area utama: **State Management**, **Dependency Injection**, dan **Route Management** — plus utilitas lain (snackbar, dialog, internationalization, storage). Di bawah ini komponen utama + fitur lanjutan.

A. State Management di GetX

Dua cara utama:

1. **Reactive (Obx + `.obs`)** — *reactive programming*
   - Definisi: `var count = 0.obs;` → `Obx(() => Text("${c.count}"))`
   - Perubahan otomatis dipantau dan widget yang pakai `Obx` akan rebuild hanya saat value berubah.
   - Tipe reaktif: `RxInt`, `RxString`, `RxList<T>`, atau `var x = <Type>[].obs;`
   - **Akses cepat**: `c.count.value` atau operator ++/-- yang sudah override (`c.count++`).
2. **GetBuilder / GetX (non-reactive / simple state)**
   - Menggunakan `update()` di controller. `GetBuilder<Ctrl>(builder: ...)` akan rebuild hanya saat `update()` dipanggil.
   - Lebih kecil overhead, cocok bila reaktivitas granular tidak diperlukan.

Contoh perbandingan singkat:

- Reactive (`Obx`) → otomatis, granular, cocok untuk banyak perubahan sering.
- GetBuilder → manual `update()`, cocok untuk refresh berkala, lebih mudah debug.

Reactive features tambahan

- **Workers**: `ever`, `once`, `debounce`, `interval` untuk memonitor perubahan Rx dan menjalankan side-effect.
  - Contoh: `ever(username, (_) => print('changed'))`
- **RxList / RxMap**: operasi list reactive (add/remove otomatis notify).

B. Dependency Injection (DI)

- `Get.put(MyController())` → menaruh instance di dependency tree (biasanya pada saat widget dibuat).
- `Get.find<MyController>()` → mengambil instance di mana saja tanpa `BuildContext`.
- `Get.lazyPut(() => MyController())` → hanya dibuat saat dipanggil pertama kali (hemat memori).
- `Get.putAsync(...)`, `Get.create(...)`, `permanent: true` — opsi lifecycle.
- **Bindings**: cara terstruktur untuk mengikat dependency pada route (rekomendasi untuk proyek besar).
  - Contoh: `class HomeBinding extends Bindings { @override void dependencies() { Get.lazyPut(() => HomeController()); } }`

C. Route Management (Navigasi)

- `Get.to(NextPage())` — push (tanpa context).
- `Get.off(NextPage())` — replace current.
- `Get.offAll(Home())` — clear stack.

- Named routes: `Get.toNamed('/detail')` + `GetPage` dengan `binding`.
- **Middleware**: intercept route (mis. auth check).
- **Transitions & animations** mudah diatur via param.

D. Lain-lain (utilitas)

- Snackbars: `Get.snackbar('title', 'message')`
- Dialog: `Get.defaultDialog(...)`
- BottomSheet: `Get.bottomSheet(...)`
- Internationalization: `Get.updateLocale(...)` / `GetMaterialApp` support
- Storage: GetStorage (key-value lightweight)
- Easy testing: controller bisa diuji terpisah.

Lifecycle di `GetxController`

- `onInit()` — dipanggil saat controller dibuat (init async, listeners).
- `onReady()` — setelah widget siap/first frame.
- `onClose()` — saat controller dihapus; cocok untuk dispose stream/close resource.

Best practices GetX

- Gunakan `Bindings` untuk DI di routing besar agar lifecycle jelas.
- Jangan menaruh semua logic di Controller; gunakan service/repository untuk I/O.
- Gunakan `.obs` untuk nilai yang sering berubah; GetBuilder untuk kasus update jarang.
- Hati-hati memory leak: unregister controller jika tidak permanen.
- Testing: simpan logic di controller/service agar mudah unit test.