LEMBAR LAPORAN PRAKTIKUM PAPB

BAB: 3
NAMA: NADHIF RIF'AT RASENDRIYA
NIM: 235150201111074
ASISTEN: 1. RIFKY AKHSANUL HADI
TGL PRAKTIKUM: 24 SEPTEMBER 2025

1. ANALISIS HASIL PERCOBAAN

SOAL 1

A. **Soal**
State Dasar

○ Apa yang terjadi pada tampilan ketika nilai count diubah pada aplikasi Counter?

○ Mengapa teks pada Button dan Text bisa langsung berubah tanpa dipanggil ulang secara manual?

Jawaban

Apa yang terjadi pada tampilan ketika nilai count diubah pada aplikasi Counter?

Ketika nilai count diubah, Jetpack Compose secara otomatis mendeteksi perubahan pada state tersebut. Akibatnya, Compose akan memicu proses yang disebut recomposition (penggambaran ulang). Namun, Compose sangat cerdas tidak menggambar ulang seluruh layar, melainkan hanya Composable Text yang menampilkan nilai count saja. Tampilan teks tersebut akan langsung diperbarui untuk merefleksikan nilai count yang baru.

Mengapa teks pada Button dan Text bisa langsung berubah tanpa dipanggil ulang secara manual?

Ini adalah inti dari paradigma UI Deklaratif yang digunakan oleh Jetpack Compose. Alasannya adalah:

- UI adalah Fungsi dari State: Anda mendeskripsikan tampilan (UI) sebagai fungsi dari data (state). Contohnya, Text(text = "Anda menekan $count kali"). Teks ini secara langsung bergantung pada nilai count.
- State yang Dapat Diamati (Observable): mutableStateOf menciptakan sebuah "wadah" state yang bisa "diamati" oleh Compose.
- Recomposition Otomatis: Ketika nilai di dalam mutableStateOf berubah, Compose secara otomatis tahu bagian UI mana yang membaca state tersebut dan perlu digambar ulang.

Jadi, kita tidak perlu memanggil fungsi setText() atau sejenisnya secara manual seperti pada UI imperatif. Kita cukup mengubah datanya (state), dan UI akan mengikutinya secara otomatis.

SOAL 2

A. **Soal**

Toggle State

○ Bagaimana Compose mengetahui kapan harus mengganti teks “Follow” menjadi “Unfollow”?

○ Apa yang akan terjadi jika variabel isFollowed dideklarasikan sebagai var biasa (tanpa remember { mutableStateOf() })?

Jawaban

Bagaimana Compose mengetahui kapan harus mengganti teks “Follow” menjadi “Unfollow”?

Compose mengetahui kapan harus memperbarui teks karena UI tersebut secara deklaratif terikat pada sebuah state yang dapat diamati (*observable state*). Prosesnya sebagai berikut:

- Teks pada tombol ditentukan oleh nilai dari state isFollowed yang dibuat menggunakan mutableStateOf().
- Ketika Button diklik, nilai dari state isFollowed diubah.
- Karena isFollowed adalah state yang diamati, Compose mendeteksi perubahan ini dan secara otomatis memicu proses recomposition (penggambaran ulang) hanya pada komponen yang membaca state tersebut.
- Selama recomposition, logika kondisional (if-else) dievaluasi ulang dengan nilai state yang baru, sehingga teks yang ditampilkan di UI ikut diperbarui.

Apa yang akan terjadi jika variabel isFollowed dideklarasikan sebagai var biasa (tanpa remember { mutableStateOf() })?

Jika variabel isFollowed dideklarasikan sebagai var biasa, maka teks pada tombol tidak akan pernah berubah setelah diklik. Ini terjadi karena dua alasan fundamental:

- Tidak Memicu Recomposition: Variabel standar pada Kotlin tidak diamati oleh *runtime* Compose. Perubahan pada nilainya tidak akan memberitahu Compose untuk melakukan recomposition, sehingga UI tidak akan diperbarui.
- State Tidak Tersimpan: Tanpa remember, nilai variabel akan diinisialisasi ulang ke nilai awalnya (false) setiap kali fungsi Composable dieksekusi (selama recomposition yang mungkin dipicu oleh hal lain). remember berfungsi untuk menyimpan state agar nilainya tetap ada di antara proses recomposition.

SOAL 3

A. **Soal**
State Hoisting

○ Jelaskan perbedaan pendekatan tanpa state hoisting dan dengan state hoisting.

○ Mengapa state hoisting membuat kode lebih bersih dan mudah digunakan kembali (reusable)?

Jawaban

Jelaskan perbedaan pendekatan tanpa state hoisting dan dengan state hoisting.

Perbedaan fundamental antara kedua pendekatan terletak pada kepemilikan dan pengelolaan state.

- Tanpa State Hoisting: Composable (komponen UI) bersifat stateful, artinya ia membuat, menyimpan, dan memodifikasi state-nya sendiri secara internal. Komponen ini bertanggung jawab penuh atas datanya, membuatnya menjadi satu unit yang mandiri namun kurang fleksibel.
- Dengan State Hoisting: Composable bersifat stateless, artinya ia tidak memiliki state internal. State "diangkat" ke atas ke level komponen induk. Komponen anak menerima state sebagai parameter dan mengkomunikasikan event (seperti klik) kembali ke induk melalui *callback*. Dengan demikian, induk bertanggung jawab penuh atas pengelolaan state, sementara anak hanya fokus pada penampilan UI.

Mengapa state hoisting membuat kode lebih bersih dan mudah digunakan kembali (reusable)?

State hoisting meningkatkan kualitas kode karena dua alasan utama:

- Pemisahan Tanggung Jawab (Separation of Concerns): Kode menjadi lebih bersih karena adanya pemisahan yang jelas. Composable anak hanya bertanggung jawab untuk menampilkan data (UI), sementara Composable induk bertanggung jawab atas logika dan pengelolaan data (state). Hal ini membuat setiap komponen lebih sederhana, lebih fokus pada satu tugas, dan lebih mudah dipahami.
- Meningkatkan Reusability: Komponen yang stateless secara inheren lebih mudah digunakan kembali. Karena tidak terikat pada sumber data internal tertentu, komponen tersebut dapat ditempatkan di mana saja dalam aplikasi dan dikendalikan oleh sumber state yang berbeda. Contohnya, FollowButton yang stateless dapat digunakan di halaman profil, daftar teman, atau pop-up, di mana setiap instance dikontrol oleh state yang relevan dari induknya masing-masing. Ini menghilangkan duplikasi kode dan mempromosikan desain yang modular.

SOAL 4

A. **Soal**
Modifier + State

○ Bagaimana Modifier (background, clickable) bekerja bersama state untuk mengubah warna kotak?

○ Apakah UI tetap responsif jika Modifier hanya digunakan tanpa state?

Jawaban

Bagaimana Modifier (background, clickable) bekerja bersama state untuk mengubah warna kotak?

Modifier dan state bekerja sama dalam sebuah alur sebab-akibat yang dikelola oleh *runtime* Compose:

- Modifier.background: Modifier ini mengatur warna kotak dengan membaca sebuah nilai state. Warna yang ditampilkan merupakan fungsi langsung dari nilai state saat ini (misalnya, if (isClicked) Color.Red else Color.Blue).
- Modifier.clickable: Modifier ini bertindak sebagai *event listener*. Ketika kotak menerima input klik, *lambda* di dalamnya dieksekusi. Tugas utama dari *lambda* ini adalah mengubah nilai state tersebut (misalnya, isClicked = !isClicked).
- Recomposition: Perubahan nilai state yang dilakukan oleh clickable secara otomatis dideteksi oleh Compose. Deteksi ini memicu proses recomposition, di mana Modifier.background dievaluasi ulang dengan nilai state yang baru, sehingga warna kotak di layar diperbarui.

Secara singkat, clickable mengubah data (state), dan background bereaksi terhadap perubahan data tersebut untuk memperbarui UI.

Apakah UI tetap responsif jika Modifier hanya digunakan tanpa state?

Tidak, UI **tidak akan responsif secara visual** jika hanya menggunakan Modifier tanpa dihubungkan dengan state.

Meskipun Modifier.clickable akan tetap mendeteksi input sentuhan dan menjalankan kode di dalamnya, tidak ada mekanisme untuk memicu pembaruan tampilan. Perubahan visual di Jetpack Compose hanya terjadi melalui proses **recomposition**, dan recomposition hanya dipicu oleh perubahan pada State yang dapat diamati.

Tanpa state sebagai pemicu, tidak ada sinyal bagi sistem untuk menggambar ulang UI dengan properti baru (seperti warna atau ukuran yang berbeda). Akibatnya, tampilan akan tetap statis.

SOAL 5

A. **Soal**
Studi Kasus Profil

○ Bagaimana tombol Follow/Unfollow di dalam ProfileCard memanfaatkan state untuk mengubah teks?

○ Jika aplikasi ditutup dan dibuka lagi, apakah state masih tersimpan? Mengapa demikian?

Jawaban

Bagaimana tombol Follow/Unfollow di dalam ProfileCard memanfaatkan state untuk mengubah teks?

Tombol Follow/Unfollow di dalam ProfileCard memanfaatkan sebuah state lokal (isFollowed) yang dideklarasikan dengan remember { mutableStateOf(false) }. Cara kerjanya adalah sebagai berikut:

- Teks yang ditampilkan pada Button ("Follow" atau "Unfollow") ditentukan secara kondisional dengan membaca nilai dari state isFollowed.
- Ketika Button diklik, event onClick dieksekusi untuk mengubah nilai boolean dari state isFollowed.
- Perubahan pada state ini secara otomatis memicu recomposition pada Button, sehingga teksnya dievaluasi ulang dan diperbarui di UI sesuai dengan nilai state yang baru.

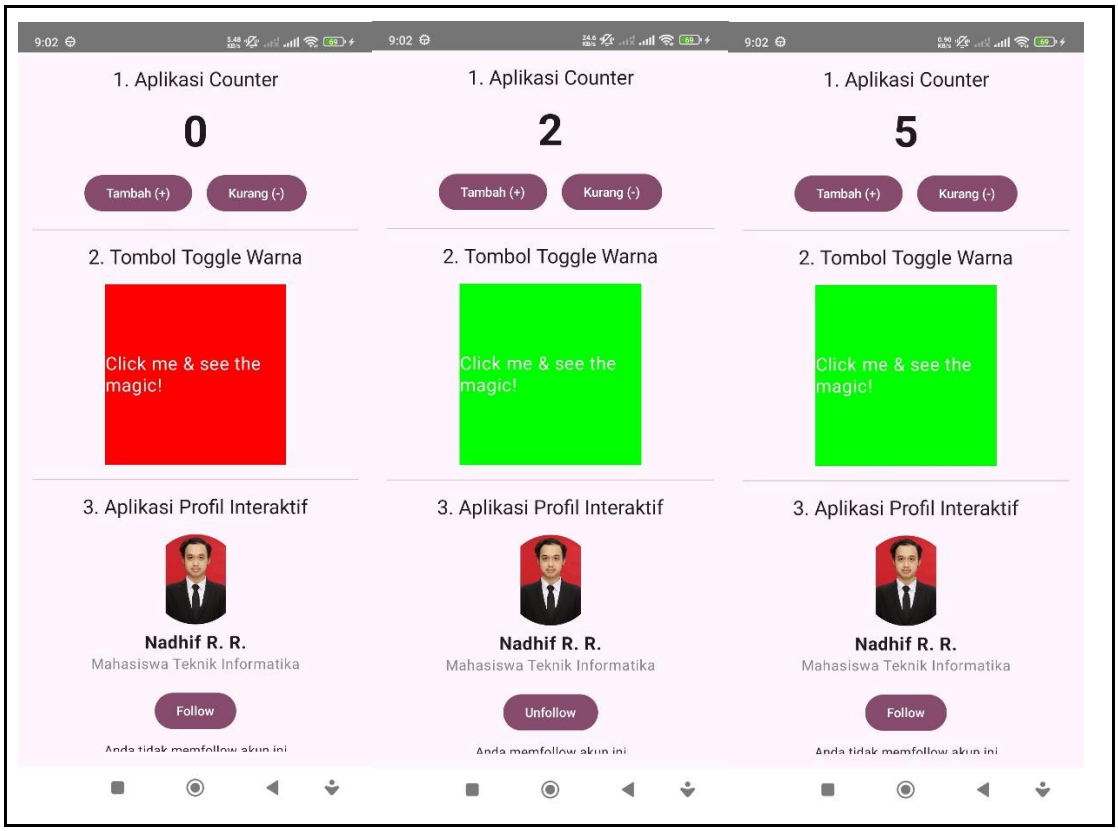
Jika aplikasi ditutup dan dibuka lagi, apakah state masih tersimpan? Mengapa demikian?

Tidak, state tersebut tidak akan tersimpan jika aplikasi ditutup (prosesnya dihentikan) dan dibuka kembali.

Alasannya adalah karena state dalam contoh ini hanya disimpan dalam memori menggunakan fungsi remember. Fungsi remember bertujuan untuk mempertahankan state selama recomposition terjadi saat aplikasi berjalan. Namun, remember tidak dirancang untuk bertahan dari penghentian proses aplikasi (process death). Ketika aplikasi ditutup, seluruh datanya yang ada di memori akan dihapus.
Untuk menyimpan state secara permanen agar tetap ada setelah aplikasi ditutup, diperlukan mekanisme persistensi data seperti SharedPreferences, DataStore, atau database lokal seperti Room.

2. TUGAS PRAKTIKUM

1. Screenshot



2. Link kode program dan README.md

Link : drive.google.com / github.com (pastikan folder atau repo public)

https://github.com/nadhif-royal/PAPB-Bab3