Test Android Developer PT. Gerak Bersama Kita	
GERAK	Nama Lengkap : Yondika Vio Landa Posisi dilamar : Android Developer File Kode: Github File Desain: Figma Hasil Aplikasi Build: Google Drive (Question4 – Question8)

Jawaban 1 : Akai dan Franco

Pemahaman Masalah

Soal ini meminta menghitung nilai minimum yang dibutuhkan agar total tiga tugas mencapai 100. Diberikan dua nilai A dan B, kita perlu mencari nilai ketiga C sehingga `A + B + C ≥ 100`.

Alur Solusi

1. Baca jumlah test case `T`.
2. Untuk setiap test case:
   - Ambil dua nilai input A dan B.
   - Hitung kekurangan dari 100 → `100 - (A + B)`.
   - Jika hasil negatif, set ke `0` (artinya sudah cukup).
3. Cetak hasil dengan format `Case #X: Y`.

Sintaks Penting

```
val t = readlnOrNull()?.trim()?.toIntOrNull() ?: return

    for (i in 1..t) {
        val parts = readlnOrNull()?.trim()?.split("\\s+".toRegex()) ?: emptyList()

        if (parts.size < 2) {
            println("Case #$i: 100")
            continue
        }

        val (a, b) = parts.map { it.toInt() }
        val perlu = (100 - (a + b)).coerceAtLeast(0)

        println("Case #$i: $perlu")
    }
```

- `readlnOrNull()` -> Membaca input dengan aman, menghindari error jika kosong.
- `split("\\s+".toRegex())` -> Memecah input berdasarkan spasi (bisa 1 atau lebih).
- `val (a, b)` -> Destructuring list agar langsung dapat variabel A dan B.

- `coerceAtLeast(0)` -> Memastikan hasil minimal 0 (tidak negatif).

Contoh Input/Output

Contoh Input	Contoh Output
4 50 20 0 0 80 90 39 60	Case #1: 30 Case #2: 100 Case #3: 0 Case #4: 1

Catatan Teknis

- Algoritma O(1) per test case, sangat ringan.
- Code clean, readable, dan mudah di-maintain.

Jawaban 2 : Akai sang panda matematikawan

Pemahaman Masalah

Soal ini menguji evaluasi ekspresi matematika step-by-step, dengan prioritas operator. Ekspresi dapat berisi angka positif/negatif, operator `+ - * :` (tanpa spasi). Operator `:` adalah pembagian bilangan bulat.

Alur Solusi

1. Baca input ekspresi matematika sebagai string.
2. Buat objek AkaiSangPandaMatematikawan dengan ekspresi sebagai parameter.
3. Panggil metode `evaluate()` untuk memulai evaluasi step-by-step:
   - Cetak ekspresi saat ini.
   - Tentukan operator yang harus dieksekusi berikutnya (`findNextOperation()`), memprioritaskan `*` dan `:` sebelum `+` dan -.
   - Buat marker (`buildMarker()`) untuk menunjukkan operator yang sedang dihitung (`-` untuk operator, . untuk karakter lain).
   - Hitung operasi yang ditandai (`calculate()`) dan ganti bagian ekspresi tersebut dengan hasilnya.
4. Cetak ekspresi terakhir sebagai hasil akhir.

Sintaks Penting

```
val input = readlnOrNull()?.trim() ?: return
```

readlnOrNull()?.trim() ?: return → membaca input dengan aman, menghentikan program jika kosong.

```
while (expr.contains(Regex("[+\\-\\*:]"))) {
```

expr.contains(Regex("[+\\-*:]")) → mengecek apakah masih ada operator tersisa.

```
val high = Regex("(-?\\d+)([\\*:])(-?\\d+)")
```

Regex("(-?\\d+)([*:])(-?\\d+)") → menangkap operasi prioritas tinggi (* dan :).

```
val low = Regex("(-?\\d+)([+-])(-?\\d+)")
```

Regex("(-?\\d+)([+-])(-?\\d+)") → menangkap operasi prioritas rendah (+ dan -).

```
val left = match.groupValues[1].toInt()
val op = match.groupValues[2][0]
val right = match.groupValues[3].toInt()
```

match.groupValues → mengambil nilai kiri, operator, dan nilai kanan dari ekspresi.

```
private fun buildMarker(expr: String, opInfo: OperationInfo): String
```

expr.substring(start, end) → memanipulasi ekspresi untuk mengganti bagian operasi dengan hasil.

```
val marker = CharArray(expr.length) { '.' }
```

CharArray(expr.length) { '.' } → membuat marker untuk menandai operator yang sedang dihitung.

```
private fun calculate(left: Int, right: Int, op: Char) = when (op)
```

when (op) → menghitung operasi sesuai operator.

Contoh Input/Output

Contoh Input	Contoh Output
23+16-8*3+4:3	23+16-8*3+4:3 .....---.... 23+16-24+4:3 ....--- 23+16-24+1 -----..... 39-24+1 -----.. 15+1 ---- 16
42+-42	42+-42 ------ 0
-5+-3	-5+-3 ----- -8

Catatan Teknis

- Regex memudahkan parsing tanpa perlu parser rumit.
- Step-by-step output memudahkan debugging.

Jawaban 3 : Akai si mahasiswa komputer

Pemahaman Masalah

Soal ini adalah problem penjadwalan multiprosesor. Diberikan K CPU dan N tugas dengan waktu eksekusi berbeda. Aturan:

1. CPU idle diprioritaskan.
2. Jika ada lebih dari satu → pilih CPU dengan tugas lebih sedikit.
3. Jika masih sama → pilih CPU dengan ID terkecil.

Output: total waktu minimum sampai semua tugas selesai.

Alur Solusi

1. Naive: simulasi detik demi detik → terlalu lambat untuk N = 1 juta.
2. Efisien: gunakan priority queue (min-heap).
   - CPU dengan `finishTime` terkecil akan dieksekusi dulu.
   - Jika sama, bandingkan `tasksCount`.
   - Jika masih sama, bandingkan `id`.
3. Track `max finishTime` sebagai hasil.

Sintaks Penting

```
data class CPU(
    val id: Int,
    var finishTime: Long = 0,
    var tasksCount: Int = 0
) : Comparable<CPU> {
    ...
}
```

- data class → otomatis menyediakan `equals`, `hashCode`, dan `toString`.
- Properti id, finishTime, tasksCount mewakili kondisi CPU.
- : Comparable<CPU> → supaya objek bisa dibandingkan saat dimasukkan ke `PriorityQueue`.

```
override fun compareTo(other: CPU): Int {
    return when {
        this.finishTime != other.finishTime ->
this.finishTime.compareTo(other.finishTime)
        this.tasksCount != other.tasksCount ->
this.tasksCount.compareTo(other.tasksCount)
        else -> this.id.compareTo(other.id)
    }
}
```

compareTo dipakai oleh PriorityQueue untuk menentukan prioritas.

Urutan prioritas:

- finishTime (paling cepat selesai lebih prioritas).
- tasksCount (paling sedikit tugas lebih prioritas).
- id (lebih kecil lebih prioritas).

```
val pq = PriorityQueue<CPU>()
```

```
for (i in 0 until K) {
    pq.add(CPU(i))
}
```

- PriorityQueue → struktur data heap (min-heap by default).
- add() → masukkan CPU ke heap.
- poll() → ambil CPU dengan prioritas tertinggi (sesuai compareTo).

```
repeat(T) { caseIndex ->
    ...
}
```

- repeat(n) → menjalankan blok kode sebanyak n kali.
- caseIndex otomatis jadi index loop (0..T-1).

```
val tasks = LongArray(N) { scanner.nextLong() }
```

- LongArray(N) → membuat array ukuran N.
- { scanner.nextLong() } → langsung isi array dengan input dari Scanner.

```
cpu.finishTime += task
cpu.tasksCount++
totalTime = maxOf(totalTime, cpu.finishTime)
```

- Update finishTime CPU dengan waktu tugas baru.
- Tambah jumlah tugas (tasksCount).
- Track totalTime untuk mencatat waktu selesai maksimum.

```
println("Case #${caseIndex + 1}: $totalTime")
```

- String template Kotlin: ${}.
- Output sesuai format soal: Case #X: Y.

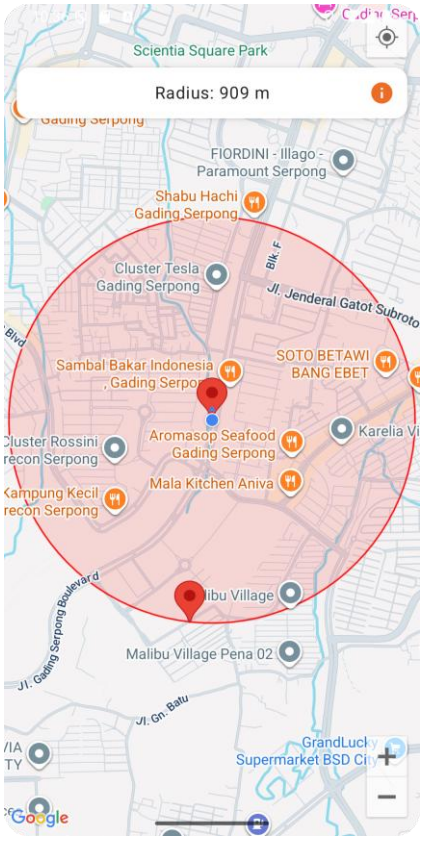
Contoh Input/Output

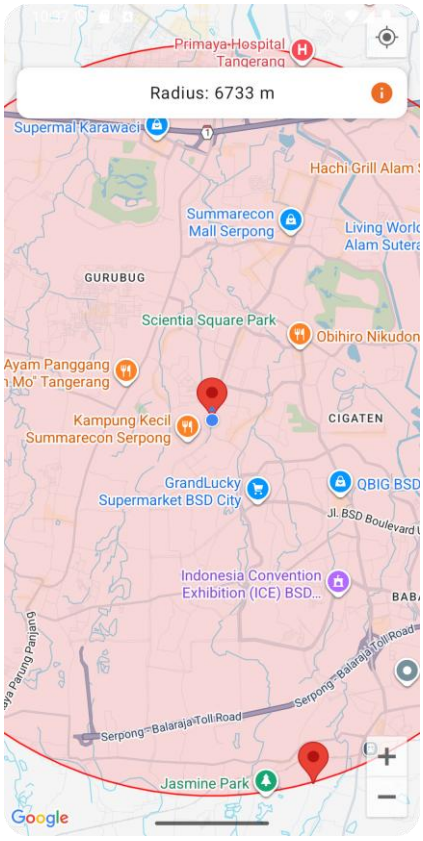
Contoh Input	Contoh Output
4 9 1 8 3 2 5 2 2 2 5 3 9 2 8 3 2 5 2 2 2 5 3 9 3 8 3 2 5 2 2 2 5 3 5 2 10 1 5 200 30	Case #1: 32 Case #2: 17 Case #3: 12 Case #4: 206

Catatan Teknis

- Solusi scalable, mampu handle hingga 1 juta tugas.
- Mirip konsep *thread pool scheduler* di sistem operasi.
- Clean code dengan `data class` membuat kode lebih maintainable.

Jawaban 4 : Akai si engineer





Pemahaman Masalah

Membuat aplikasi Android berbasis Jetpack Compose yang menampilkan peta dengan lingkaran radius dinamis. Lingkaran menggambarkan area tertentu, radius berubah sesuai input user (klik pada peta).

Komponen Utama

1. Integrasi Google Maps SDK Compose.
2. Simpan titik pusat dan radius sebagai state.
3. Render lingkaran (Circle) sesuai state.
4. Update radius/titik setiap kali user berinteraksi dengan peta.

Sintaks Penting

```
var userLocation by remember { mutableStateOf<LatLng?>(null) }
```

Menyimpan lokasi pengguna secara stateful. Nilai ini otomatis akan mengupdate UI ketika berubah.

```
LaunchedEffect(locationPermission.status.isGranted) { ... }
```

Efek samping (side effect) di Compose. Digunakan untuk mengambil lokasi terakhir pengguna setelah izin lokasi diberikan.

Dokumen ini saya susun untuk menunjukkan kemampuan teknis dalam menyelesaikan problem algoritma, membangun aplikasi Android berbasis Kotlin dan Jetpack Compose, serta melakukan perbaikan UI/UX dengan pendekatan yang konsisten dan terukur.

```
GoogleMap(...) { ... }
```
Komponen utama untuk menampilkan peta Google Maps di Jetpack Compose.
Di dalamnya bisa ditambahkan Marker, Circle, dan interaksi seperti onMapClick.

```
Marker(state = MarkerState(position = it), ...)
```
Menampilkan pin/titik pada peta di lokasi tertentu (lokasi user maupun titik terpilih).

```
Circle(center = it, radius = radius, ...)
```
Menambahkan lingkaran di peta dengan pusat (center) dan jari-jari (radius).
Digunakan untuk menggambarkan luas area berdasarkan radius.

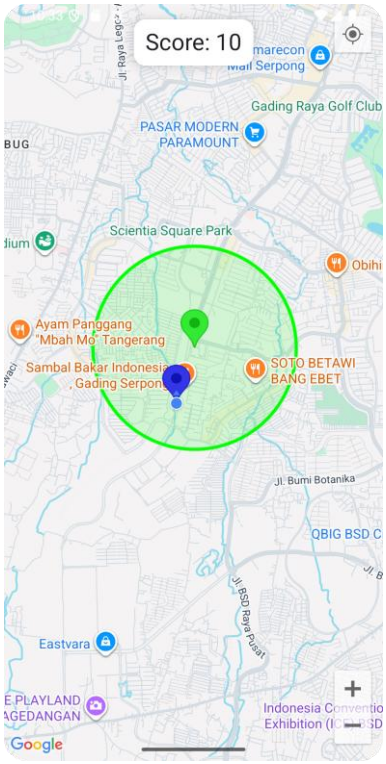
```
Location.distanceBetween(...)
```
Fungsi dari Android SDK untuk menghitung jarak (dalam meter) antara dua titik koordinat (latitude & longitude).

```
Card(...) { Column { Text(...) ... } }
```
Wadah UI berbentuk kartu (Material 3).
Digunakan untuk menampilkan informasi radius dan instruksi ke user.

Catatan Teknis

- State management dengan Compose (`remember`, `mutableStateOf`).
- Jetpack Compose membuat UI deklaratif dan lebih ringkas.
- Maps Compose mempermudah integrasi UI modern dengan peta.

Jawaban 5 : Akai si engineer 2



Pemahaman Masalah

Aplikasi “Gerak GO”: game sederhana berbasis lokasi. User mendapat poin saat memasuki radius checkpoint 1 km.

Alur Aplikasi

1. Minta izin lokasi.
2. Tampilkan peta dengan posisi user.
3. Definisikan checkpoint (marker merah + lingkaran radius).
4. Hitung jarak user ke checkpoint tiap update GPS.
5. Jika ≤ 1 km dan belum pernah dikunjungi → ubah marker hijau + tambah skor.

Tampilan Aplikasi

- Marker Biru → Lokasi pengguna.
- Marker Merah → Checkpoint yang belum dikunjungi.
- Marker Hijau → Checkpoint yang sudah dikunjungi (skor sudah didapat).
- Lingkaran Transparan → Area radius 1 km dari checkpoint.
- Card di atas Map → Menampilkan skor terbaru.

Sintaks Penting

```
GoogleMap( modifier = Modifier.fillMaxSize(), cameraPositionState =
cameraPositionState, properties = MapProperties(isMyLocationEnabled =
locationPermission.status.isGranted) )
```

Digunakan untuk menampilkan peta interaktif. Properti `isMyLocationEnabled` mengaktifkan titik lokasi pengguna di peta.

```
Marker( state = MarkerState(position = it), title = "Lokasi Kamu", icon =
BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE) )
```

`Marker` digunakan untuk menandai lokasi pada peta. Warna dan teks dapat diubah sesuai status (misalnya, checkpoint yang sudah dikunjungi menjadi hijau).

```
Circle( center = point, radius = 1000.0, strokeColor = Color.Red, fillColor =
Color(0x22FF0000) )
```

`Circle` digunakan untuk menggambar radius 1 km pada setiap checkpoint. Warna merah menunjukkan checkpoint belum dikunjungi, sedangkan hijau berarti sudah dikunjungi.

```
val results = FloatArray(1) Location.distanceBetween( uLoc.latitude,
uLoc.longitude, point.latitude, point.longitude, results ) val distance =
results[0]
```

`Location.distanceBetween()` menghitung jarak dalam meter antara dua koordinat (latitude & longitude). Hasilnya digunakan untuk menentukan apakah user berada dalam radius checkpoint.

```
if (distance <= 1000 && !isVisited) { visitedCheckpoints.add(point) score += 10
}
```

Logika utama game: jika jarak ≤ 1000 meter dan checkpoint belum pernah dikunjungi, maka checkpoint ditandai sebagai “visited” dan skor bertambah 10.

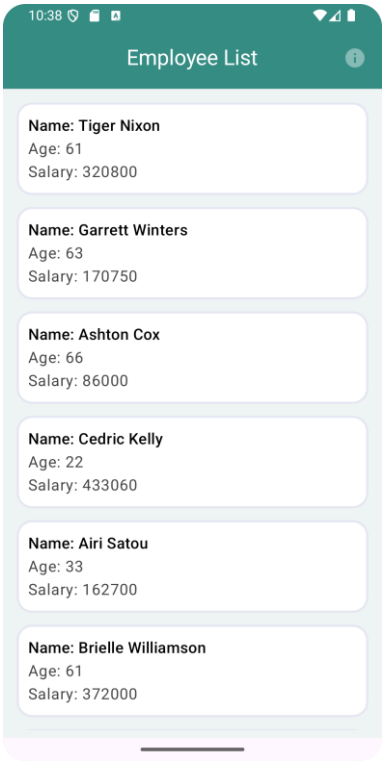
```
Card( modifier = Modifier.align(Alignment.TopCenter).padding(16.dp), colors =
CardDefaults.cardColors(containerColor = Color.White) ) { Text(text = "Score:
$score") }
```

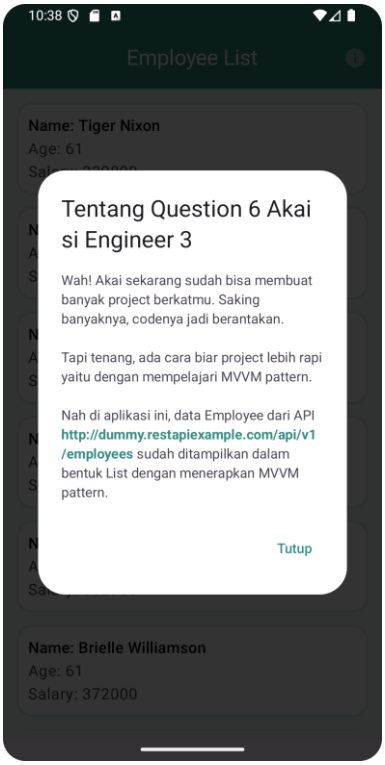
Skor ditampilkan secara realtime dalam sebuah Card di bagian atas layar.

Catatan Teknis

- Gunakan `ACCESS_FINE_LOCATION` untuk akurasi tinggi.
- Google Maps Compose untuk rendering peta.
- Material 3 untuk desain modern.

Jawaban 6 : Akai si engineer 3





Pemahaman Masalah

Aplikasi Android untuk menampilkan daftar karyawan dari API eksternal.
Arsitektur utama: **MVVM** agar project rapi dan maintainable.

Alur Solusi

1. UI (Compose) → minta data ke ViewModel.
2. ViewModel → ambil data dari Repository.
3. Repository → panggil ApiService (Retrofit).
4. API → response dikirim balik ke Repository → ViewModel → UI.
5. UI render data dengan LazyColumn.

Sintaks Penting

```
@SerializedName("employee_name") val employeeName: String
```

Fungsi: Memberi tahu Retrofit/Gson untuk memetakan field JSON (employee_name) ke property Kotlin (employeeName).
Manfaat: Data dari API bisa langsung dipakai tanpa manual parsing.

```
interface ApiService { @GET("employees") suspend fun getEmployees(): EmployeeResponse }
```

Dokumen ini saya susun untuk menunjukkan kemampuan teknis dalam menyelesaikan problem algoritma, membangun aplikasi Android berbasis Kotlin dan Jetpack Compose, serta melakukan perbaikan UI/UX dengan pendekatan yang konsisten dan terukur.

Fungsi: Mendefinisikan endpoint API dengan Retrofit.
Manfaat: Abstraksi pemanggilan HTTP → jadi cukup panggil getEmployees() untuk fetch data.

```
class EmployeeRepository { suspend fun getEmployees(): EmployeeResponse }
```

Fungsi: Repository sebagai jembatan antara ViewModel dan ApiService.
Manfaat: Memisahkan logic pengambilan data dari UI → sesuai prinsip MVVM.

```
class EmployeeViewModel : ViewModel()
```

Fungsi: Mengatur state (employees, isLoading) dan memanggil repository.
Manfaat: ViewModel tetap survive saat konfigurasi berubah (misalnya rotate screen).

```
private val _employees = MutableStateFlow<List<Employee>>(emptyList())
```

Fungsi: Menyimpan state daftar employee secara reactive.
Manfaat: Setiap kali datanya berubah, UI otomatis update.

```
val employees by viewModel.employees.collectAsState()
```

Fungsi: Menghubungkan StateFlow dari ViewModel ke UI Compose.
Manfaat: Jetpack Compose bisa “mendengar” perubahan data tanpa manual refresh.

```
LazyColumn { items(employees) { employee -> EmployeeItem(employee) } }
```

Fungsi: Menampilkan list data secara efisien (hanya render item yang terlihat di layar).
Manfaat: Performant & cocok untuk dataset panjang.

```
Card(modifier = Modifier.fillMaxWidth().padding(6.dp), colors = …)
```

Fungsi: Membuat container UI untuk tiap employee.
Manfaat: Biar tampilannya rapih dengan background putih & border abu-abu.

```
Scaffold(topBar = { CenterAlignedTopAppBar(...) })
```

Fungsi: Menyediakan struktur UI standar (TopBar, FloatingActionButton, body).
Manfaat: Memudahkan konsistensi layout aplikasi.

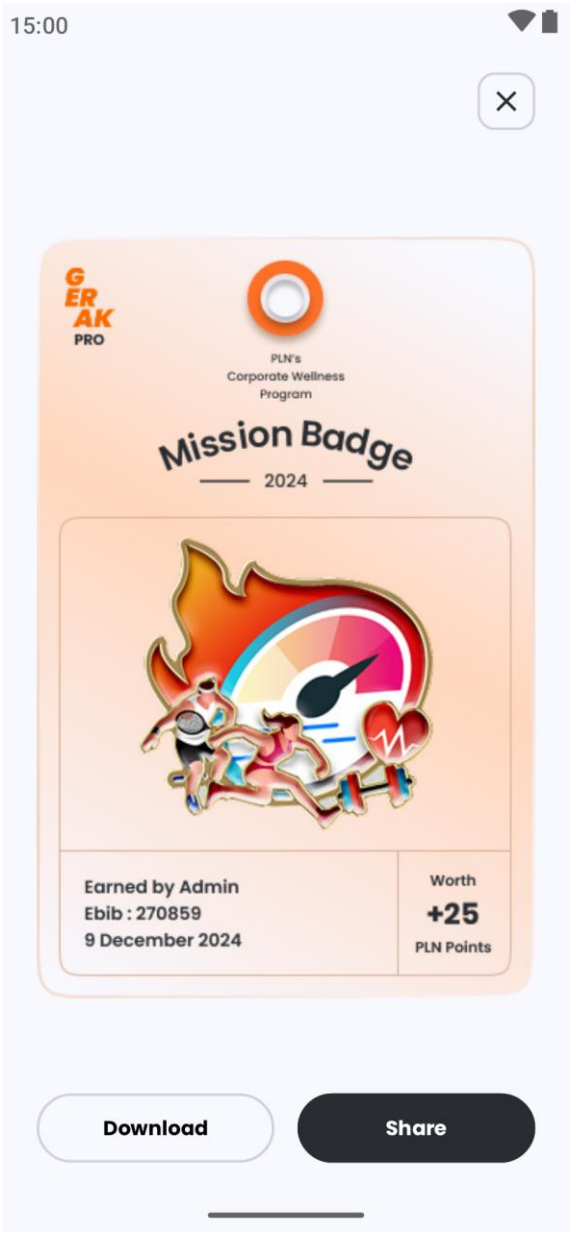
```
AlertDialog(onDismissRequest = { ... }, title = { Text("Info") }, … )
```

Fungsi: Menampilkan popup dialog informasi.
Manfaat: Memberi user context tentang aplikasi & API yang digunakan.

Catatan Teknis

- MVVM → separation of concern.
- Retrofit + OkHttp untuk networking.
- StateFlow untuk observable data → Compose UI lebih reaktif..

Jawaban 7 : Memperbaiki UI

Sebelum Perbaikan	Setelah Perbaikan
	

Pemahaman Masalah

UI awal kurang kontras, branding tidak konsisten, dan button terlihat tidak harmonis.

Perbaikan yang dilakukan

1. Visual Hierarchy lebih baik

- Versi kanan (after) punya kontras yang lebih baik: background putih dengan aksen oranye → badge jadi lebih menonjol, tidak "tenggelam" seperti versi kiri yang abu-abu kusam.

2. Branding lebih konsisten
   - Warna oranye yang dipakai di lingkaran gantungan sesuai dengan logo "GERAK PRO" → ada sense konsistensi.
   - Secara psikologi warna, oranye juga memberi energi & semangat (cocok untuk konsep *wellness program*).

3. Typography lebih bersih
   - Tulisan lebih rapi, terutama bagian detail "Earned by, Ebib, Date, Worth" → spacing lebih lega, lebih mudah dibaca.

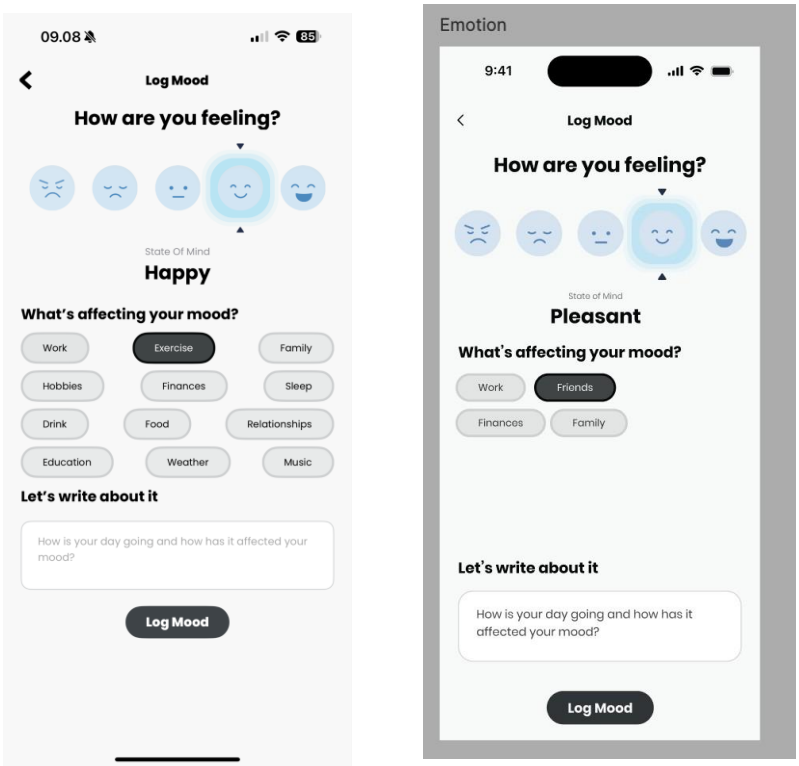
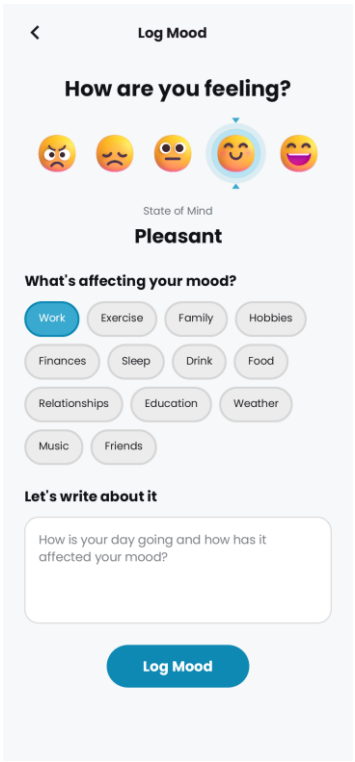
4. Button lebih harmonis
   - Di versi kiri, tombol Download/Share terlalu kontras dengan background gelap, ditambah dengan shadow putih yang membuat warna bertabrakan.
   - Versi kanan tombolnya pakai *outlined* putih & hitam solid → lebih modern, sesuai dengan tren desain saat ini.

5. Kesan premium & modern
   - Overall versi kanan terlihat lebih fresh, lebih premium, dan sesuai dengan standar aplikasi kesehatan/achievement modern.

Catatan Teknis

- Perbaikan alignment dengan Modifier.padding dan Modifier.fillMaxWidth().
- Gunakan Material3 ButtonStyle agar konsisten.

Jawaban 8 : Memperbaiki UI Part 2

Sebelum Perbaikan	Setelah Perbaikan
	

Pada **UI bagian kiri** merupakan hasil implementasi awal dari kode yang sudah dibuat.
Pada **UI bagian tengah** adalah desain referensi dari **Figma.**
Pada **UI bagian kanan** adalah hasil **perbaikan UI** yang saya lakukan untuk menyelaraskan tampilan dengan desain Figma sekaligus melakukan beberapa improvisasi agar lebih konsisten, mudah digunakan, dan estetis.

Catatan: Karena saya tidak memiliki asset icon yang ada pada Figma, saya memutuskan untuk menggantinya, tapi ini bukan bagian dari perbaikan, karena desain emoji sebenarnya sudah oke (Perbaikan yang bisa dilakukan adalah perubahan warna agar lebih kontras(merah, orange, kuning, hijau muda, hijau tua)).

Pemahaman Masalah
UI awal tidak sepenuhnya sesuai desain Figma, hierarki visual lemah, dan usability kurang optimal.

Perbaikan yang Dilakukan
1. Judul Pertanyaan "How are you feeling?"

Sebelum	Setelah	Alasan

Sudah sesuai desain, namun posisi dan proporsi kurang seimbang dengan elemen lain	Saya menyesuaikan spasi (padding/margin) agar lebih seimbang dengan emoji dan konten di bawahnya	Membuat hierarki visual lebih jelas, memudahkan pengguna memindai layar

2. State of Mind (Happy / Pleasant, dll.)

Sebelum	Setelah	Alasan
Tulisan state of mind tampil agak kecil dan tidak terlalu menonjol	Saya menyesuaikan ukuran font agar lebih konsisten dan mudah terbaca	Status emosi adalah informasi utama, sehingga harus cukup terlihat jelas

3. Tag Mood Influences (Work, Friends, Family, dll.)

Sebelum	Setelah	Alasan
Tampil rata horizontal tanpa terlalu menonjol, spacing kurang rapi	Saya menambahkan spacing yang lebih konsisten, padding dalam setiap tag diperbesar agar lebih mudah diklik, serta memastikan alignment sesuai desain Figma	Perbaikan usability, mempermudah interaksi pengguna dengan tombol tag

4. Text Area ("Let's write about it")

Sebelum	Setelah	Alasan
Kotak input terlalu kecil secara visual, kurang proporsional dengan elemen lain	Kotak input diperbesar, mengikuti proporsi yang lebih baik dengan tombol di bawahnya	Memberikan ruang lebih untuk menulis, mendukung pengalaman pengguna yang lebih nyaman
Posisi berdasarkan desain, berada di paling bawah, berdasarkan code tepat dibawah tag mood influences	Posisinya berada tepat dibawah tag mood influences	karena lebih proporsional mengingat tiap smartphone berbeda dimensi, bisa saya overflow vertical dan posisi text area jadi tertutup

5. Button "Log Mood"

Sebelum	Setelah	Alasan
Sudah ada, namun tidak terlalu menonjol	Button diberi ukuran lebih proporsional, teks lebih kontras agar mudah dikenali sebagai CTA (Call To Action), dan warna dibuat lebih cerah menyesuaikan dengan warna chip dan mood.	CTA harus jelas terlihat agar alur logging mood tidak membingungkan pengguna

---

Hasil Akhir (UI Kanan)

Perbaikan UI ini menghasilkan tampilan yang:

1. Lebih dekat dengan desain Figma.
2. Lebih jelas dan komunikatif dengan penggunaan emoji berwarna.
3. Lebih konsisten dalam penggunaan spacing, padding, dan ukuran font.

4. Lebih user-friendly dalam hal interaksi tombol, input text, dan hierarki visual.

Catatan Teknis

- Menggunakan `Column/ConstraintLayout` untuk menjaga alignment dan hierarki antar elemen.
- Menambahkan modifier padding, spacing, dan weight pada komponen (`Chip, TextField, Button`) agar konsisten.
- Mengoptimalkan scalable font size dan komponen responsif agar UI tetap proporsional di berbagai ukuran layar.
- Menggunakan state management sederhana (`remember`, `mutableStateOf`) untuk mengelola mood, tags, dan catatan.
- Menyesuaikan styling dengan Material3 (`Typography, Shapes, ColorScheme`) untuk konsistensi desain.

Sekian,

Nama Lengkap : Yondika Vio Landa
Posisi yang dilamar : Android Developer

Kontak:
Email : yondikaviolanda@gmail.com
No. Telp/WA : +62895605086353