

**LAPORAN PRAKTIKUM  
PEMROGRAMAN MOBILE  
MODUL 2**



**ANDROID LAYOUT WITH COMPOSE**

**Oleh:**

**Devi Hafida Ariyani**

**NIM. 2310817220018**

**PROGRAM STUDI TEKNOLOGI INFORMASI  
FAKULTAS TEKNIK  
UNIVERSITAS LAMBUNG MANGKURAT  
APRIL 2025**

**LEMBAR PENGESAHAN**  
**LAPORAN PRAKTIKUM PEMROGRAMAN I**  
**MODUL 2**

Laporan Praktikum Pemrograman Mobile Modul 2: Android Layout With Compose ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Devi Hafida Ariyani  
NIM : 2310817220018

Menyetujui,  
Asisten Praktikum

Mengetahui,  
Dosen Penanggung Jawab Praktikum

Muhammad Raka Azwar  
NIM. 2210817210012

Andreyan Rizky Baskara, S.Kom., M.Kom.  
NIP. 19930703 201903 01 011

## DAFTAR ISI

LEMBAR PENGESAHAN.....	2
DAFTAR ISI .....	3
DAFTAR GAMBAR.....	4
DAFTAR TABEL .....	5
SOAL 1 .....	6
A. SOURCE CODE .....	7
B. OUTPUT PROGRAM.....	15
C. PEMBAHASAN.....	18
D. TAUTAN GIT .....	18
SOAL 2.....	19

## DAFTAR GAMBAR

Gambar 1. Tampilan Awal Aplikasi.....	6
Gambar 2. Tampilan Pilihan Persentase Tip .....	6
Gambar 3. Tampilan Aplikasi Setelah Dijalankan .....	7
Gambar 4. Soal 1 (Compose) .....	15
Gambar 5. Soal 1 (Compose) .....	16
Gambar 6. Soal 1 (Compose) .....	16
Gambar 7. Soal 1 (XML).....	17
Gambar 8. Soal 1 (XML).....	17

## DAFTAR TABEL

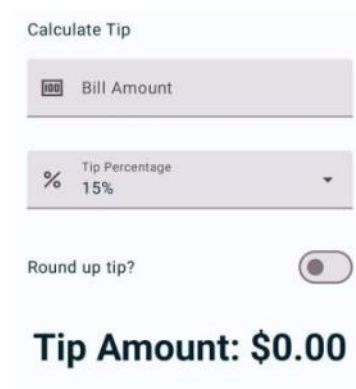
Tabel 1. Source Code Jawaban Soal 1.....	7
Tabel 2. Source Code Jawaban Soal 1.....	8
Tabel 3. Source Code Jawaban Soal 1.....	10
Tabel 4. Source Code Jawaban Soal 1.....	11

## SOAL 1

### Soal Praktikum:

Buatlah sebuah aplikasi kalkulator tip menggunakan XML dan Jetpack Compose yang dirancang untuk membantu pengguna menghitung tip yang sesuai berdasarkan total biaya layanan yang mereka terima. Fitur-fitur yang diharapkan dalam aplikasi ini mencakup:

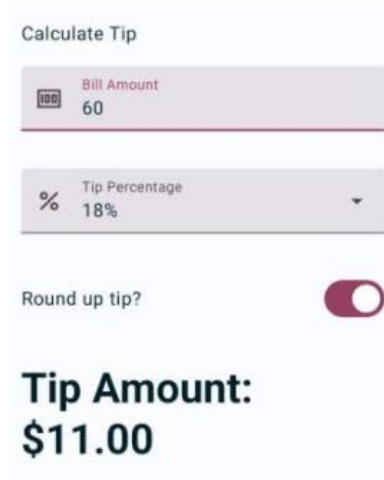
- Input biaya layanan: Pengguna dapat memasukkan total biaya layanan yang diterima dalam bentuk nominal.
- Pilihan persentase tip: Pengguna dapat memilih persentase tip yang diinginkan.
- Pengaturan pembulatan tip: Pengguna dapat memilih untuk membulatkan tip ke angka yang lebih tinggi.
- Tampilan hasil: Aplikasi akan menampilkan jumlah tip yang harus dibayar secara langsung setelah pengguna memberikan input.



Gambar 1. Tampilan Awal Aplikasi



Gambar 2. Tampilan Pilihan Persentase Tip



Gambar 3. Tampilan Aplikasi Setelah Dijalankan

## A. SOURCE CODE

### XML

### XmlActivity.kt

Tabel 1. Source Code Jawaban Soal 1

```
package com.example.tipcalculation

import android.annotation.SuppressLint
import android.os.Bundle
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import java.text.NumberFormat
import kotlin.math.ceil

class XmlActivity : AppCompatActivity() {

    private lateinit var costEditText: EditText
    private lateinit var tipResult: TextView
    private lateinit var tipOptions: RadioGroup
    @SuppressLint("UseSwitchCompatOrMaterialCode")
    private lateinit var roundUpSwitch: Switch
    private lateinit var switchToCompose: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_xml)

        // Inisialisasi view
        costEditText = findViewById(R.id.costOfServiceEditText)
        tipResult = findViewById(R.id.tipResult)
        tipOptions = findViewById(R.id.tipOptions)
        roundUpSwitch = findViewById(R.id.roundUpSwitch)

        // Hitung tip saat user menekan enter
        costEditText.setOnEditorActionListener { _, _, _ ->
            calculateTip()
        }
    }
}
```

```

        true
    }
}

private fun calculateTip() {
    val cost = costEditText.text.toString().toDoubleOrNull()
    if (cost == null) {
        tipResult.text = ""
        return
    }

    val tipPercentage = when (tipOptions.checkedRadioButtonId) {
        R.id.optionTwenty -> 0.20
        R.id.optionEighteen -> 0.18
        R.id.optionFifteen -> 0.15
        else -> 0.15
    }

    var tip = cost * tipPercentage
    if (roundUpSwitch.isChecked) {
        tip = ceil(tip)
    }

    val formattedTip =
    NumberFormat.getCurrencyInstance().format(tip)
    tipResult.text = getString(R.string.tip_amount, formattedTip)
}
}

```

## activity\_xml.xml

*Tabel 2. Source Code Jawaban Soal 1*

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="24dp">

        <EditText
            android:id="@+id/costOfServiceEditText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:autofillHints=""
            android:hint="@string/enter_cost_of_service"
            android:inputType="numberDecimal"
            android:minHeight="48dp"
            android:padding="12dp" />

        <RadioGroup
            android:id="@+id/tipOptions"

```



```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_marginTop="16dp">

        <RadioButton
            android:id="@+id/optionTwenty"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/option_twenty"
            android:minHeight="48dp"
            android:padding="12dp" />

        <RadioButton
            android:id="@+id/optionEighteen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/option_eighteen"
            android:minHeight="48dp"
            android:padding="12dp" />

        <RadioButton
            android:id="@+id/optionFifteen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/option_fifteen"
            android:minHeight="48dp"
            android:padding="12dp" />
    </RadioGroup>

    <Switch
        android:id="@+id/roundUpSwitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/round_up"
        android:layout_marginTop="44dp"
        android:padding="12dp"
        tools:ignore="UseSwitchCompatOrMaterialXml" />

    <TextView
        android:id="@+id/tipResult"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/tip_default"
        android:textSize="20sp"
        android:layout_marginTop="16dp" />

    <Button
        android:id="@+id/calculateButton"
        android:layout_width="147dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Calculate" />
</LinearLayout>
</ScrollView>

```

**MainActivity.kt**

*Tabel 3. Source Code Jawaban Soal 1*

```
package com.example.tipcalculation

import android.os.Bundle
import android.view.inputmethod.EditorInfo
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import java.text.NumberFormat
import kotlin.math.ceil

class MainActivity : AppCompatActivity() {

    private lateinit var costEditText: EditText
    private lateinit var tipResult: TextView
    private lateinit var tipOptions: RadioGroup
    private lateinit var roundUpSwitch: Switch
    private lateinit var calculateButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_xml)

        costEditText = findViewById(R.id.costOfServiceEditText)
        tipResult = findViewById(R.id.tipResult)
        tipOptions = findViewById(R.id.tipOptions)
        roundUpSwitch = findViewById(R.id.roundUpSwitch)
        calculateButton = findViewById(R.id.calculateButton)

        // Listener tombol Calculate
        calculateButton.setOnClickListener {
            calculateTip()
        }

        // Listener keyboard Done
        costEditText.setOnEditorActionListener { _, actionId, _ ->
            if (actionId == EditorInfo.IME_ACTION_DONE) {
                calculateTip()
                true
            } else {
                false
            }
        }
    }

    private fun calculateTip() {
        val cost = costEditText.text.toString().toDoubleOrNull()
        if (cost == null || cost == 0.0) {
            tipResult.text = getString(R.string.tip_default)
            return
        }

        val tipPercentage = when (tipOptions.checkedRadioButtonId) {
            R.id.optionTwenty -> 0.20
            R.id.optionEighteen -> 0.18
            R.id.optionFifteen -> 0.15
            else -> 0.15
        }
```

```

    }

    var tip = cost * tipPercentage
    if (roundUpSwitch.isChecked) {
        tip = ceil(tip)
    }

    val formattedTip =
        NumberFormat.getCurrencyInstance().format(tip)
    tipResult.text = getString(R.string.tip_amount, formattedTip)
}

```

## Compose

### MainActivity.kt

*Tabel 4. Source Code Jawaban Soal 1*

```

package com.example.tipcalculation

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.text.KeyboardOptions
import java.text.NumberFormat
import kotlin.math.ceil
import androidx.compose.ui.tooling.preview.Preview as Preview1
import androidx.compose.material.icons.Icons

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {

```

```

        TipCalculatorApp()

    }

}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TipCalculatorApp() {
    var amountInput by remember { mutableStateOf("") }
    var tipPercent by remember { mutableDoubleStateOf(0.18) }
    var roundUp by remember { mutableStateOf(true) }
    val amount = amountInput.toDoubleOrNull() ?: 0.0
    var tip = amount * tipPercent
    if (roundUp) tip = ceil(tip)

    val formattedTip = "$" + String.format("%.2f", tip)
    val percentageOptions = listOf(0.15, 0.18, 0.20)
    val expanded = remember { mutableStateOf(false) }

    Column(modifier = Modifier
        .fillMaxWidth()
        .padding(24.dp)) {

        Text(text = "Calculate Tip", style =
MaterialTheme.typography.titleMedium)

        Spacer(Modifier.height(8.dp))

        OutlinedTextField(
            value = amountInput,
            onValueChange = { amountInput = it },
            label = { Text("Bill Amount") },

```

```

        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),

        modifier = Modifier.fillMaxWidth()

    )

    Spacer(Modifier.height(16.dp))

    ExposedDropdownMenuBox(
        expanded = expanded.value,
        onExpandedChange = { expanded.value = !expanded.value }
    ) {
        OutlinedTextField(
            readOnly = true,
            value = "${(tipPercent * 100).toInt()}%",
            onValueChange = {},
            label = { Text("Tip Percentage") },
            trailingIcon = {
                ExposedDropdownMenuDefaults.TrailingIcon(expanded
= expanded.value)
            },
            modifier = Modifier
                .menuAnchor()
                .fillMaxWidth()
        )

        ExposedDropdownMenu(
            expanded = expanded.value,
            onDismissRequest = { expanded.value = false }
        ) {
            percentageOptions.forEach { option ->
                DropdownMenuItem(
                    text = { Text("${(option * 100).toInt()}%") },

```

```

        onClick = {
            tipPercent = option
            expanded.value = false
        }
    )
}

}

}

}

Spacer(Modifier.height(16.dp))

Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier.fillMaxWidth()
) {
    Text("Round up tip?")
    Spacer(modifier = Modifier.weight(1f))
    Switch(
        checked = roundUp,
        onCheckedChange = { roundUp = it }
    )
}

Spacer(Modifier.height(32.dp))

Text(
    text = "Tip Amount:",
    style = MaterialTheme.typography.headlineSmall
)

Text(
    text = formattedTip,
    style = MaterialTheme.typography.headlineSmall,

```

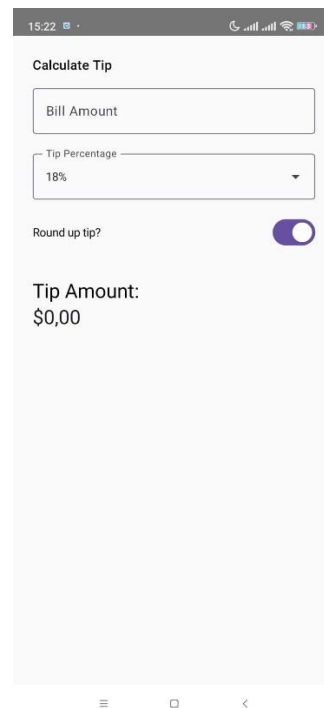
```

        color = MaterialTheme.colorScheme.onBackground
    )
}

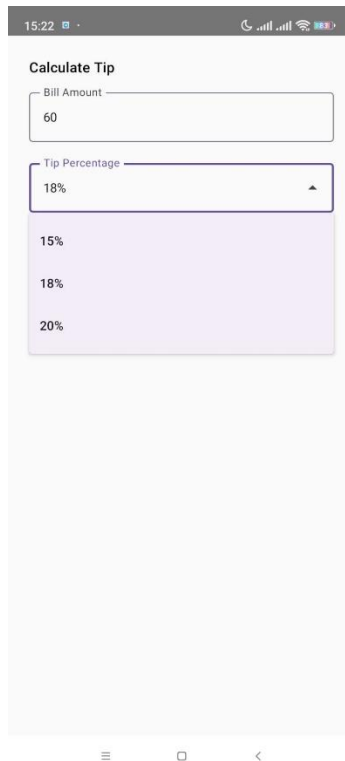
@Preview1(showBackground = true)
@Composable
fun TipCalculatorAppPreview() {
    TipCalculatorApp()
}

```

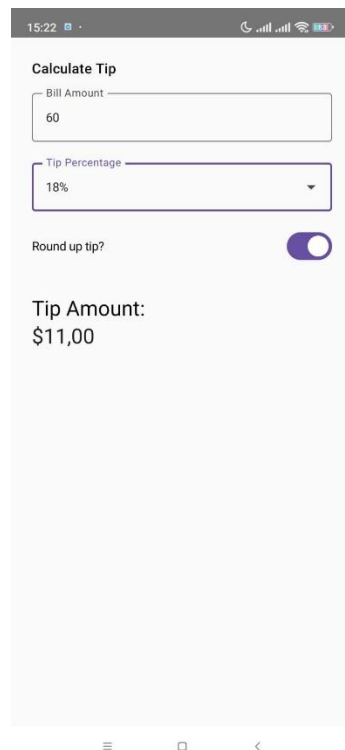
## B. OUTPUT PROGRAM



*Gambar 4. Soal 1 (Compose)*



*Gambar 5. Soal 1 (Compose)*



*Gambar 6. Soal 1 (Compose)*



2:27

Enter cost of service

☐ 20%

☐ 18%

☐ 15%

Round up tip? ☐

Tip Amount:

CALCULATE

*Gambar 7. Soal 1 (XML)*

2:27

60

☐ 20%

☒ 18%

☐ 15%

Round up tip? ☒

Tip Amount: Rp11,00

CALCULATE

*Gambar 8. Soal 1 (XML)*

### C. PEMBAHASAN

XmlActivity.kt adalah activity berbasis XML layout yang berfungsi sebagai kalkulator tip sederhana. Di sini, komponen UI seperti EditText untuk input biaya layanan, RadioGroup untuk memilih persentase tip, Switch untuk opsi pembulatan tip, dan TextView untuk menampilkan hasil diinisialisasi menggunakan `findViewById`. Kalkulasi tip dilakukan saat pengguna menekan tombol "Done" pada keyboard dengan memanggil fungsi `calculateTip()`, yang mengambil nilai input, menentukan persentase tip berdasarkan RadioButton yang dipilih, menghitung tip, melakukan pembulatan jika diaktifkan, dan menampilkan hasil dalam format mata uang lokal.

`activity_xml.xml` adalah file layout XML yang mendefinisikan tampilan UI untuk kalkulator tip di aplikasi. Layout ini menggunakan ScrollView dengan LinearLayout vertikal berisi EditText untuk input biaya, RadioGroup dengan tiga RadioButton untuk memilih persentase tip (15%, 18%, 20%), sebuah Switch untuk memilih opsi pembulatan tip, TextView untuk menampilkan hasil perhitungan tip, serta sebuah tombol "Calculate" untuk memicu kalkulasi secara manual. Elemen-elemen diberi padding dan margin agar tampilan lebih rapi dan nyaman digunakan.

MainActivity.kt (XML) adalah activity utama yang menggunakan layout `activity_xml.xml` dan menginisialisasi semua view dengan `findViewById` seperti di XmlActivity, tetapi memiliki tambahan tombol `calculateButton` yang jika diklik akan memicu kalkulasi tip. Selain itu, tombol keyboard "Done" juga di-handle agar memanggil kalkulasi tip. Fungsi `calculateTip()` mirip dengan XmlActivity tapi dengan validasi input tambahan yang memeriksa jika input kosong atau nol, maka menampilkan teks default.

MainActivity.kt (Compose) adalah implementasi kalkulator tip yang sepenuhnya menggunakan Jetpack Compose tanpa layout XML. Aktivitas ini menampilkan UI berbasis composable function yang menggunakan state untuk menangani input biaya, persentase tip yang dipilih dari dropdown menu, dan switch untuk opsi pembulatan. Perhitungan tip dilakukan secara reaktif sesuai state saat ini, hasilnya diformat ke mata uang dolar dan ditampilkan langsung. UI terdiri dari field input teks, dropdown untuk persentase, switch pembulatan, serta tampilan hasil yang diatur dalam Column dengan padding dan spacing agar mudah dibaca dan digunakan.

### D. TAUTAN GIT

Berikut adalah tautan untuk source code yang telah dibuat.

<https://github.com/DeviHafida/Mobile>

## SOAL 2

Jelaskan perbedaan dari implementasi XML dan Jetpack Compose beserta kelebihan dan kekurangan dari masing-masing implementasi.

### 1. XML (Traditional View-based UI)

XML adalah cara tradisional untuk mendesain antarmuka pengguna di Android. UI didefinisikan menggunakan file XML yang diproses oleh sistem Android untuk membuat tampilan aplikasi.

Cara Kerja:

- Layout UI dibuat di dalam file XML yang terpisah (misalnya `activity_main.xml`).
- Elemen-elemen UI seperti `TextView`, `EditText`, `Button`, dan `Spinner` ditentukan dengan tag XML, dan kemudian dikaitkan dengan kode Kotlin atau Java menggunakan `findViewById()`.

Kelebihan:

- Deklaratif dan Terpisah: Penggunaan XML memisahkan logika aplikasi (kode Kotlin) dan antarmuka pengguna (UI), membuat kode lebih terstruktur dan mudah dipahami.
- Pemakaian Library Pihak Ketiga: XML lebih mudah digunakan bersama dengan berbagai library UI pihak ketiga, yang banyak tersedia di Android.
- Kompatibilitas yang Lebih Luas: XML adalah standar yang sudah ada sejak lama, sehingga hampir semua perangkat Android mendukungnya.

Kekurangan:

- Verbosity: Menggunakan XML bisa memerlukan banyak baris kode, terutama ketika UI menjadi kompleks. Menambahkan banyak elemen bisa membuat kode tampak berantakan.
- Pengelolaan Proyek yang Lebih Sulit: Pengelolaan antarmuka pengguna (UI) yang melibatkan banyak file XML dan pengaturan `findViewById()` bisa menjadi lebih sulit, terutama ketika aplikasi berkembang.
- Kurang Fleksibel: Membuat perubahan dinamis pada UI (seperti mengganti tampilan berdasarkan data) bisa lebih rumit dan memerlukan pembaruan elemen UI satu per satu.

### 2. Jetpack Compose (Declarative UI Framework)

Jetpack Compose adalah framework UI deklaratif modern untuk Android, yang memungkinkan pengembang membuat antarmuka pengguna dengan kode Kotlin, tanpa memerlukan XML. Compose memungkinkan pembuatan UI dengan cara yang lebih langsung dan reaktif.

### Cara Kerja:

- UI didefinisikan langsung di dalam kode Kotlin menggunakan fungsi-fungsi deklaratif (misalnya `Text()`, `Button()`, `Column()`, dll.).
- Pengguna tidak perlu lagi menggunakan `findViewById()`, karena elemen-elemen UI langsung dipetakan ke komponen Kotlin.
- UI dapat diperbarui secara reaktif berdasarkan perubahan data menggunakan state management.

### Kelebihan:

- Deklaratif dan Mudah Dibaca: Karena Compose menggunakan paradigma deklaratif, UI didefinisikan dalam satu tempat, membuatnya lebih mudah untuk dipahami dan diubah.
- Pengelolaan State yang Lebih Baik: Compose memungkinkan pengelolaan status UI secara lebih efisien menggunakan state dan remember, memungkinkan aplikasi bereaksi terhadap perubahan data secara otomatis.
- Fleksibilitas dan Kinerja: Compose memungkinkan penyesuaian UI dengan lebih bebas, dan lebih efisien dalam hal kinerja dibandingkan dengan XML, terutama untuk animasi dan pembaruan tampilan yang dinamis.
- Integrasi dengan Kotlin: Compose sepenuhnya terintegrasi dengan Kotlin, sehingga memudahkan pengembangan dan pemeliharaan aplikasi.

### Kekurangan:

- Kurva Pembelajaran: Meskipun Compose lebih intuitif, ada kurva pembelajaran yang lebih curam bagi pengembang yang belum terbiasa dengan pendekatan deklaratif.
- Keterbatasan Kompatibilitas dengan Library Lama: Karena Jetpack Compose adalah framework baru, beberapa library pihak ketiga atau elemen UI yang lebih tua mungkin belum sepenuhnya kompatibel dengan Compose.
- Masih dalam Pengembangan: Meskipun Jetpack Compose sudah stabil, beberapa fitur dan optimasi masih terus berkembang, yang mungkin memengaruhi keputusan untuk mengadopsinya dalam aplikasi besar yang sudah ada.

### Perbandingan dalam Konteks Aplikasi Tip Calculation

- Dengan XML:
  - Pendekatan: UI aplikasi kalkulator tip (misalnya, `EditText` untuk biaya, `Spinner` untuk persentase tip, dan `Button` untuk menghitung) didefinisikan di file XML, dan interaksi dengan elemen-elemen UI dilakukan dalam kode Kotlin menggunakan `findViewById()`.
  - Kelebihan: Lebih mudah digunakan bagi pengembang yang sudah berpengalaman dengan Android SDK tradisional dan XML. Penggunaan library pihak ketiga yang lebih banyak tersedia.

- Kekurangan: UI menjadi lebih sulit untuk dikelola seiring dengan berkembangnya aplikasi. Perubahan dinamis di UI (misalnya pembaruan hasil kalkulasi) memerlukan manipulasi manual elemen UI.
- Dengan Jetpack Compose:
  - Pendekatan: UI aplikasi kalkulator tip bisa langsung didefinisikan dalam kode Kotlin, menggunakan elemen seperti `Text()`, `Button()`, dan `Slider()`. Dengan Compose, pembaruan hasil tip dapat dilakukan lebih efisien dengan mengubah nilai state dan secara otomatis memperbarui tampilan.
  - Kelebihan: UI lebih fleksibel dan reaktif. Pengelolaan status (seperti hasil kalkulasi tip) lebih mudah dilakukan, dan UI dapat diperbarui secara otomatis saat data berubah.
  - Kekurangan: Penggunaan Jetpack Compose memerlukan waktu belajar dan penyesuaian, terutama bagi pengembang yang belum terbiasa dengan pendekatan deklaratif. Keterbatasan kompatibilitas dengan beberapa library pihak ketiga bisa menjadi tantangan.