# LAPORAN AKHIR PRAKTIKUM
# PEMROGRAMAN MOBILE


**Oleh:**

**Natalie Grace Katiandagho** **NIM. 2310817120003**

**PROGRAM STUDI TEKNOLOGI INFORMASI**

**FAKULTAS TEKNIK**

**UNIVERSITAS LAMBUNG MANGKURAT**

**2025**

# LEMBAR PENGESAHAN
# LAPORAN AKHIR PRAKTIKUM PEMROGRAMAN MOBILE

Laporan Praktikum Pemrograman Mobile
Modul 1: Android Basic With Kotlin
Modul 2: Android Layout
Modul 3: Build A Scrollable List
Modul 4: View Model and Debugging
Modul 5: Connect to the Internet
ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile.
Laporan Akhir Praktikum ini dikerjakan oleh:

Nama Praktikan : Natalie Grace Katiandagho
NIM : 2310817120003

| | |
|---|---|
| Menyetujui, | Mengetahui, |
| Asisten Praktikum | Dosen Penanggung Jawab Praktikum |
| Zulfa Auliya Akbar | Muti`a Maulida S.Kom M.T.I |
| NIM. 2210817210026 | NIP. 19881027 201903 20 13 |

# DAFTAR ISI

# DAFTAR GAMBAR

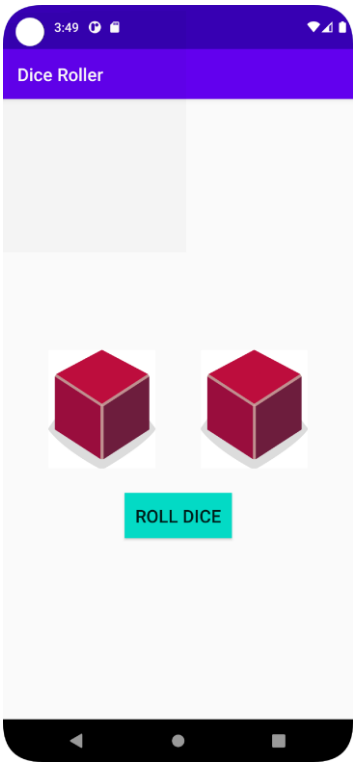
# DAFTAR TABEL

# MODUL 1 : Android Basic With Kotlin

## SOAL 1

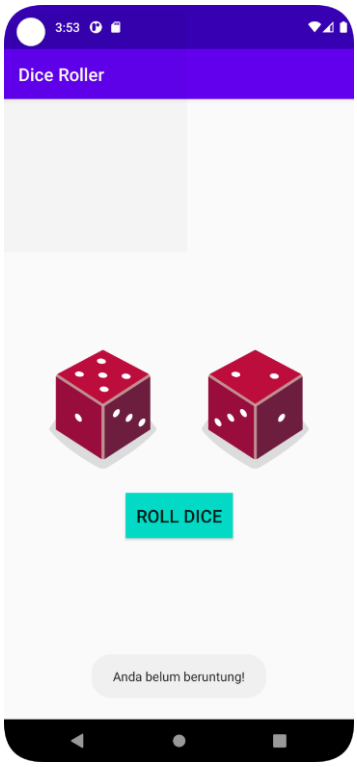
**Soal Praktikum:**
Buatlah sebuah aplikasi yang dapat menampilkan 2 (dua) buah dadu yang dapat berubah-ubah tampilannya pada saat user menekan tombol "Roll Dice". Aturan aplikasi yang akan dibangun adalah sebagaimana berikut:

1. Tampilan awal aplikasi setelah dijalankan akan menampilkan 2 buah dadu kosong seperti dapat dilihat pada Gambar 1.




*Gambar 1. Tampilan Awal Aplikasi*

2. Setelah user menekan tombol "Roll Dice" maka masing-masing dadu akan memunculkan sisi dadu masing-masing dengan angka antara 1 s/d 6. Apabila user mendapatkan nilai dadu yang berbeda antara Dadu 1 dengan Dadu 2 maka akan menampilkan pesan "Anda belum beruntung!" seperti dapat dilihat pada Gambar 2.

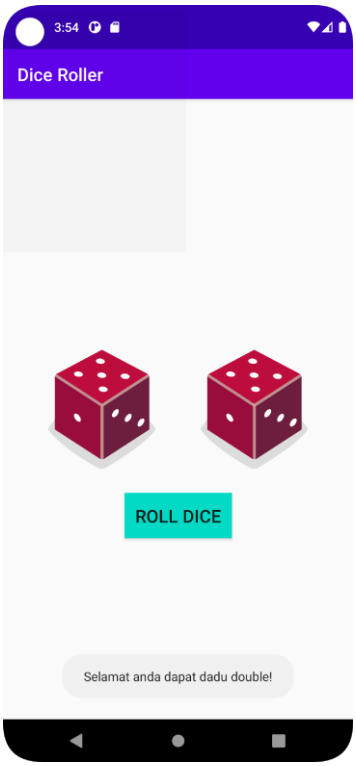



*Gambar 2. Tampilan Dadu Setelah Di Roll*

3. Apabila user mendapatkan nilai dadu yang sama antara Dadu 1 dan Dadu 2 atau nilai double, maka aplikasi akan menampilkan pesan "Selamat anda dapat dadu double!" seperti dapat dilihat pada Gambar 3.

4. Upload aplikasi yang telah anda buat kedalam repository github ke dalam **folder Module 2 dalam bentuk project.** Jangan lupa untuk melakukan **Clean Project** sebelum mengupload pekerjaan anda pada repo.

5. Untuk gambar dadu dapat didownload pada link berikut: https://drive.google.com/u/0/uc?id=147HT2lIH5qin3z5ta7H9y2N_5OMW81Ll&export= download




*Gambar 3. Tampilan Roll Dadu Double*

## A. Source Code

*Tabel 1. MainActivity Modul1*

```
1  package com.example.xml_diceroller
2
3  import android.os.Bundle
   import android.widget.Toast.LENGTH_LONG
   import androidx.activity.enableEdgeToEdge
   import androidx.appcompat.app.AppCompatActivity
   import androidx.core.view.ViewCompat
   import androidx.core.view.WindowInsetsCompat
   import
   com.example.xml_diceroller.databinding.ActivityMainBinding
   import com.google.android.material.snackbar.Snackbar

   class MainActivity : AppCompatActivity() {
       private lateinit var binding : ActivityMainBinding
       override fun onCreate(savedInstanceState: Bundle?) {
           super.onCreate(savedInstanceState)
           enableEdgeToEdge()
           binding =
```

```
ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.button.setOnClickListener {
            NgerollDadu()
        }
    }
    fun NgerollDadu() {
        val dadu = dadu(6)
        val sisi = dadu.NgerollDadu()
        val sisi2 = dadu.NgerollDadu()
        val NgerollDadu1 = when (sisi) {
            1 -> R.drawable.dadu1
            2 -> R.drawable.dadu2
            3 -> R.drawable.dadu3
            4 -> R.drawable.dadu4
            5 -> R.drawable.dadu5
            else -> R.drawable.dadu6
        }
        val NgerollDadu2 = when (sisi2) {
            1 -> R.drawable.dadu1
            2 -> R.drawable.dadu2
            3 -> R.drawable.dadu3
            4 -> R.drawable.dadu4
            5 -> R.drawable.dadu5
            else -> R.drawable.dadu6
        }

        binding.sisi.setImageResource(NgerollDadu1)
        binding.sisi2.setImageResource(NgerollDadu2)

        val pesan = if (sisi == sisi2) {
            "Selamat, anda mendapatkan dadu double"
        } else {
            "Anda belum beruntung"
        }
        Snackbar.make(binding.button, pesan,
LENGTH_LONG).show()
    }
}

class dadu(private val numSisi : Int)
{
```

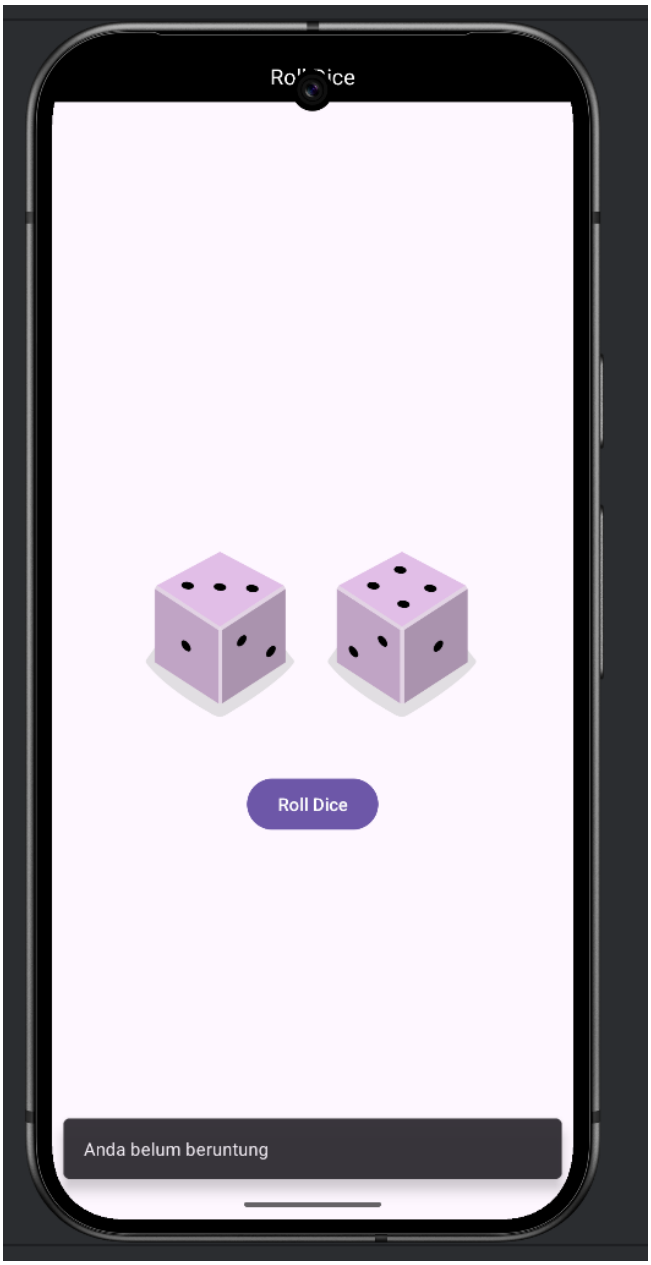
```
    fun NgerollDadu() : Int
    {
        return(1..numSisi).random()
    }
}
```

*Tabel 2. ActivityMain Modul1*

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="42dp"
        android:background="@color/black"
        android:paddingVertical="10dp"
        android:text="Roll Dice"
        android:textAlignment="center"
        android:textColor="@color/white"
        android:textSize="17dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        />

    <ImageView
        android:id="@+id/sisi"
        android:layout_width="150dp"
        android:layout_height="170dp"
        android:src="@drawable/dadu0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintHorizontal_bias="0.22"
        app:layout_constraintVertical_bias="0.5"/>

    <ImageView
        android:id="@+id/sisi2"
        android:layout_width="150dp"
        android:layout_height="170dp"
        android:src="@drawable/dadu0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.77"
        app:layout_constraintVertical_bias="0.5"  />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout_editor_absoluteX="-40dp" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="25dp"
        android:text="Roll Dice"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/sisi"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

## B. Output Program




*Gambar 4. Screenshot Hasil Jawaban Soal 1 Modul 1*

## C. Pembahasan

1. MainActivity.kt

2. ActivityMain.kt

# MODUL 2 : Android Layout

## SOAL 1

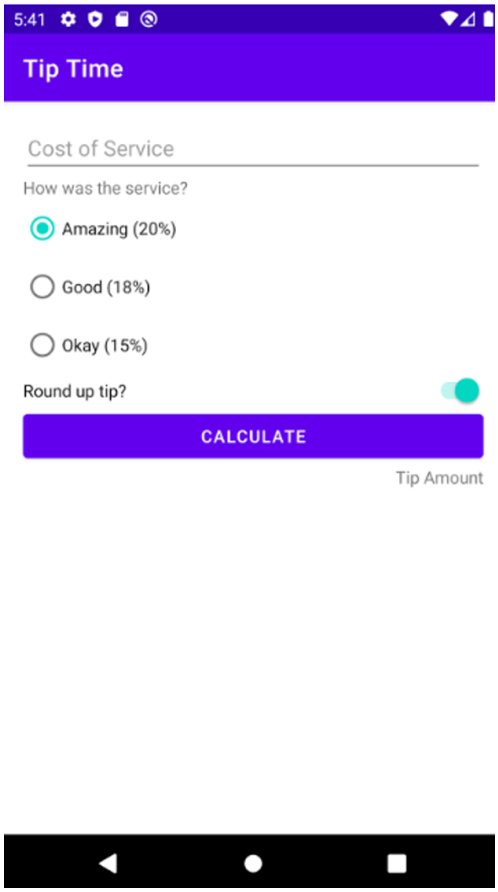
Buatlah sebuah aplikasi kalkulator tip yang dirancang untuk membantu pengguna menghitung tip yang sesuai berdasarkan total biaya layanan yang mereka terima. Fitur-fitur yang diharapkan dalam aplikasi ini mencakup:

1. Input Biaya Layanan: Pengguna dapat memasukkan total biaya layanan yang diterima dalam bentuk nominal.

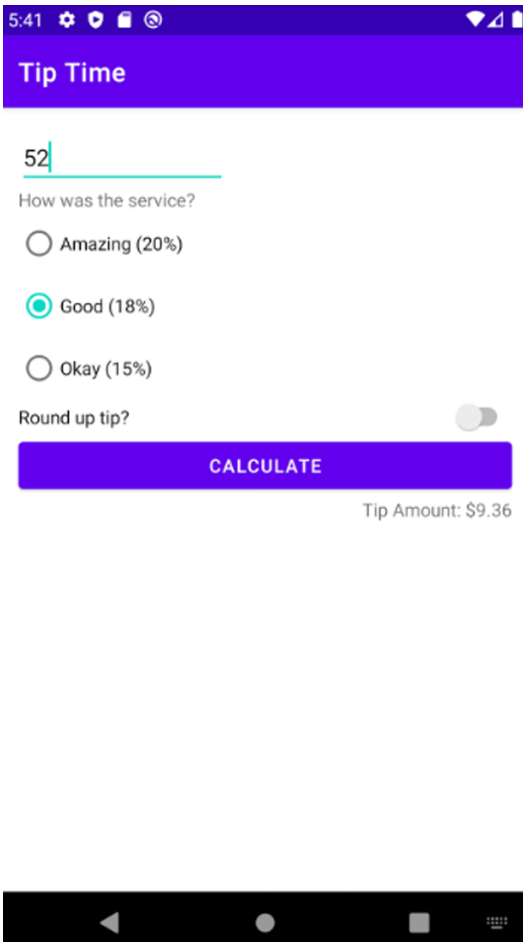
2. Pilihan Persentase Tip: Pengguna dapat memilih persentase tip yang diinginkan dari opsi yang disediakan, yaitu 15%, 18%, dan 20%.

3. Pengaturan Pembulatan Tip: Pengguna dapat memilih untuk membulatkan tip ke angka yang lebih tinggi.

4. Tampilan Hasil: Aplikasi akan menampilkan jumlah tip yang harus dibayar secara langsung setelah pengguna memberikan input.




*Gambar 5. Tampilan Awal Aplikasi*




*Gambar 6. Tampilan Aplikasi Setelah Dijalankan*

## A. Source Code

*Tabel 3. MainActivity Modul 2*

```
1   package com.example.tipcalculator
2
3   import android.os.Bundle
4   import androidx.activity.ComponentActivity
5   import androidx.activity.compose.setContent
6   import androidx.annotation.StringRes
7   import androidx.compose.foundation.layout.Arrangement
8   import androidx.compose.foundation.layout.Column
9   import androidx.compose.foundation.layout.Row
10  import androidx.compose.foundation.layout.fillMaxSize
11  import androidx.compose.foundation.layout.fillMaxWidth
12  import androidx.compose.foundation.layout.height
13  import androidx.compose.foundation.layout.padding
14  import androidx.compose.foundation.rememberScrollState
15  import androidx.compose.foundation.text.KeyboardOptions
```

```
16  import androidx.compose.foundation.verticalScroll
17  import androidx.compose.material3.MaterialTheme
18  import androidx.compose.material3.RadioButton
19  import androidx.compose.material3.Surface
20  import androidx.compose.material3.Switch
21  import androidx.compose.material3.Text
22  import androidx.compose.material3.TextField
23  import androidx.compose.runtime.Composable
24  import androidx.compose.runtime.getValue
25  import androidx.compose.runtime.mutableDoubleStateOf
26  import androidx.compose.runtime.mutableStateOf
27  import androidx.compose.runtime.remember
28  import androidx.compose.runtime.setValue
29  import androidx.compose.ui.Alignment
30  import androidx.compose.ui.Modifier
31  import androidx.compose.ui.res.stringResource
32  import androidx.compose.ui.text.input.ImeAction
33  import androidx.compose.ui.text.input.KeyboardType
34  import androidx.compose.ui.tooling.preview.Preview
35  import androidx.compose.ui.unit.dp
36  import com.example.tipcalculator.ui.theme.TipCalculatorTheme
37  import java.text.NumberFormat
38  import kotlin.math.ceil
39
40  class MainActivity : ComponentActivity() {
41      override fun onCreate(savedInstanceState: Bundle?) {
42          super.onCreate(savedInstanceState)
43          setContent {
44              TipCalculatorTheme {
45                  Surface(modifier = Modifier.fillMaxSize()) {
46                      TipCalculatorLayout()
47                  }
48              }
49          }
50      }
51  }
52
53  @Composable
54  fun TipCalculatorLayout() {
55      var amountInput by remember { mutableStateOf("") }
56      var tipPercent by remember { mutableDoubleStateOf(15.0) }
57      var roundUp by remember { mutableStateOf(false) }
58
59      val amount = amountInput.toDoubleOrNull() ?: 0.0
60      val tip = calculateTip(amount, tipPercent, roundUp)
61
62      Column(
63          modifier = Modifier
64              .padding(32.dp)
```

```
65              .verticalScroll(rememberScrollState()),
66          horizontalAlignment = Alignment.CenterHorizontally,
67          verticalArrangement = Arrangement.spacedBy(16.dp)
68      ) {
69          Text(text = stringResource(R.string.calculate_tip),
    style = MaterialTheme.typography.headlineMedium)
70
71          EditNumberField(
72              label = R.string.bill_amount,
73              value = amountInput,
74              onValueChanged = { amountInput = it },
75              keyboardOptions = KeyboardOptions(
76                  keyboardType = KeyboardType.Number,
77                  imeAction = ImeAction.Next
78              )
79          )
80
81          TipOptions(tipPercent) {
82              tipPercent = it
83          }
84
85          RoundTheTipRow(roundUp = roundUp, onRoundUpChanged = {
    roundUp = it })
86
87          Text(
88              text = stringResource(R.string.tip_amount, tip),
89              style = MaterialTheme.typography.headlineSmall
90          )
91      }
92  }
93
94  @Composable
95  fun EditNumberField(
96      value: String,
97      @StringRes label: Int,
98      onValueChanged: (String) -> Unit,
99      keyboardOptions: KeyboardOptions,
100     modifier: Modifier = Modifier
101 ) {
102     TextField(
103         value = value,
104         onValueChange = onValueChanged,
105         singleLine = true,
106         modifier = modifier.fillMaxWidth(),
107         label = { Text(stringResource(label)) },
108         keyboardOptions = keyboardOptions
109     )
110 }
111
```

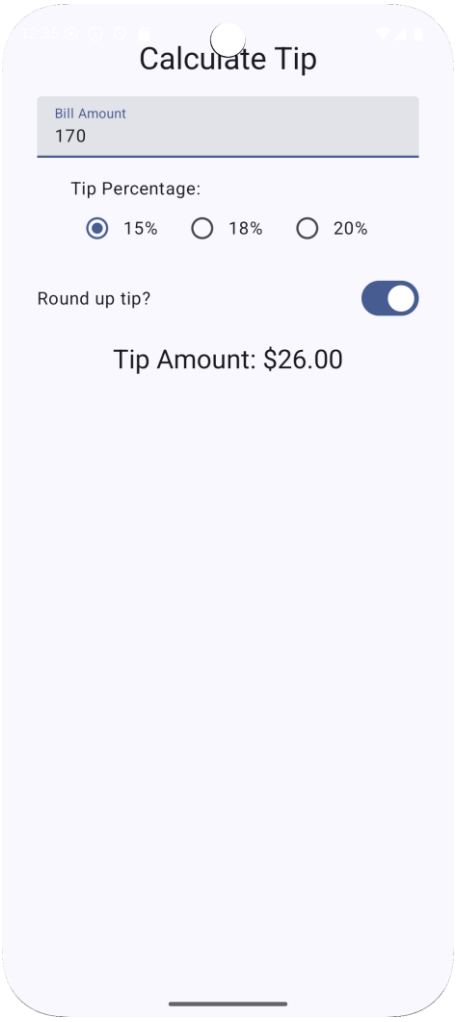
```
@Composable
fun TipOptions(selectedTip: Double, onTipSelected: (Double) ->
Unit) {
    Column {
        Text(text = "Tip Percentage:")
        Row(verticalAlignment = Alignment.CenterVertically) {
            listOf(15.0, 18.0, 20.0).forEach { percent ->
                Row(
                    verticalAlignment =
Alignment.CenterVertically,
                    modifier = Modifier.padding(end = 16.dp)
                ) {
                    RadioButton(
                        selected = selectedTip == percent,
                        onClick = { onTipSelected(percent) }
                    )
                    Text(text = "${percent.toInt()}%")
                }
            }
        }
    }
}

@Composable
fun RoundTheTipRow(
    roundUp: Boolean,
    onRoundUpChanged: (Boolean) -> Unit
) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .height(48.dp),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        Text(
            text = stringResource(R.string.round_up_tip),
            modifier = Modifier.weight(1f)
        )
        Switch(
            checked = roundUp,
            onCheckedChange = onRoundUpChanged
        )
    }
}

private fun calculateTip(amount: Double, tipPercent: Double =
15.0, roundUp: Boolean): String {
    var tip = tipPercent / 100 * amount
```

```
158     if (roundUp) {
159         tip = ceil(tip)
160     }
161     return NumberFormat.getCurrencyInstance().format(tip)
162 }
163 
164 @Preview(showBackground = true)
165 @Composable
166 fun TipCalculatorPreview() {
167     TipCalculatorTheme {
168         TipCalculatorLayout()
169     }
170 }
```

*Tabel 4. String.xml Modul 2*

```
1   <resources>
2       <string name="app_name">Tip Calculator</string>
3       <string name="calculate_tip">Calculate Tip</string>
4       <string name="bill_amount">Bill Amount</string>
5       <string name="tip">Tip (%)</string>
6       <string name="round_up_tip">Round up tip?</string>
7       <string name="tip_amount">Tip Amount: %1$s</string>
8   </resources>
```

## B. Output Program




*Gambar 7. Screenshoot Hasil Modul 2*

## C. Pembahasan

### 1. MainActivity.kt:

Pada line 1, dideklarasikan nama package file Kotlin yang pada praktikum kali ini adalah com.example.tipcalculcator

**Dari line 3-38 fungsinya untuk mengimport :**

Pada line 3, Bundle buat komunikasiin data antar aktivitas

Pada line 4, ComponentActivity sebagai superclass untuk activity yang menggunakan Jetpack Compose
Pada line 5, setContent untuk menampilkan UI berbasis Compose pada activity
Pada line 6, @StringRes jadi anotasi untuk resource ID bertipe string
**Pada line 7–13, diimpor beberapa komponen layout :**

- Arragement, untuk pengaturan posisi child dalam layout
- Column, Row untuk struktur tata letak vertikal dan horizontal
- fillMaxSize, fillMaxWidth, height, padding untuk mengatur ukuran dan margin komponen

Pada line 14, rememberScrollState untuk menyimpan dan mengingat posisi scroll
Pada line 15, diimport KeyboardOptions untuk mengatur jenis keyboard dan aksi input
Pada line 16, diimport verticalScroll untuk memungkinkan scroll secara vertikal pada layout
**Pada line 17–22, diimpor komponen Material 3:**

- MaterialTheme, untuk mengakses tema aplikasi
- RadioButton untuk pemilihan opsi
- Switch untuk tombol on off fitur
- Text menampilkan tulisan ke layar
- TextField, digunakan untuk mengambil input teks dari pengguna

**Pada line 23–28, fitur state Compose seperti:**

- @Composable, untuk menandai fungsi sebagai composable
- remember, mutableStateOf, mutableDoubleStateOf, untuk mendefinisikan dan mengingat state yang dapat berubah
- getValue, setValue, untuk properti delegasi state

**Pada line 29–35 :**

- Modifier untuk mengatur tampilan, ukuran, padding komponen
- stringResource untuk mengambil string dari resource menggunakan ID
- ImeAction, KeyboardType untuk pengaturan aksi keyboard dan tipe input
- @Preview untuk pratinjau UI
- dp untuk unit ukuran
- TipCalculatorTheme untuk menggunakan tema khusus aplikasi
- NumberFormat untuk format angka

Pada line 36, untuk menerapkan tema khusus aplikasi yang sudah didefinisikan di folder ui.theme.
Pada line 37, untuk format angka ke dalam format mata uang.
Pada line 38, ceil dari kotlin.math untuk membulatkan nilai ke atas.

Pada line 40, kelas MainActivity yang turunan dari ComponentActivity
Pada line 41, ada fungsi yang dipanggil saat activity pertama kali dibuat. Parameter savedInstanceState digunakan untuk menyimpan data jika activity perlu dibuat ulang.
Pada line 42, memanggil onCreate dari superclass (ComponentActivity) agar fungsi dasar activity tetap berjalan.
Pada line 43, setContent untuk menampilkan UI didalam kurung kurawalnya nanti jadi isi tampilan Activity.

Pada line 44, buat temanya
Pada line 45, suface buat tempat UI bisa dikasih warna dll, fillMaxSize buat ngisi layar penuh
Pada line 46, manggil fungsi TipCalculatorLayout

Pada line 53, buat nandain fungsi UI.
Pada line 54, buat fungsi utama tampilan kalkulator.
Pada line 55, nyimpan input tagihan dari pengguna.
Pada line 56, buat nyimpan presentase tipnya.
Pada line 57, buat nyimpan pilihan tip mau dibultkan atau engga.
Pada line 59, buat ngubah input ke angka, kalau kosong maka 0.0
Pada line 60, ngitung jumlah tip dari input, persen sama pembulatan tadi.

Pada line 62, buat nyusun semua elemen dari atas ke bawah.
Pada line 63 – 65, buat beri jarak 32dp di isi, terus buat kolom supaya bisa dicroll kalau isinya kepanjangan.
Pada line 66, buat rata tengah horizontal.
Pada line 67, buat ngasih jarak 16dp antar elemen.

Pada line 69, buat ngasih judul : Calculate Tip, uk medium.

Pada line 71-77, buat nampilin input angka di bill amount ada ambil nilai input, diperbarui setiap pengguna mengetik.

Pada line 81-82 buat nampilin pilihan persen tip

Pada line 85 isinya menampilkan switch on off buat tip dibulatkan atau engga, terus ngumbah nilai roundUp nya pas diganti.

Pada line 86-88, ada text lagi buat nampilkan perhitungan tip dalam bentuk teks – uk teks kecil.

Pada line 93 ada composable lagi buat tampila UI.
Pada line 94, buat nampilin input angka .
Pada line, 95-99 ada :

- value: String , Nilai input yang sedang diketik.
- label: Int (@StringRes) , ID dari teks label
- onValueChanged: (String) -> Unit fungsi saat pengguna mengetik untuk memperbarui nilai.
- keyboardOptions: KeyboardOptions ngatur jenis keyboard.
- modifier: Modifier untuk pengaturan tampilan tambahan.

Pada TextField (line 101-107) nampilin input teks satu baris, lebarmya fillMaxWidth, label teks dari stringResouce(label), keyboardnya sesuai keyboardOptions.

Pada 111 ada composable lagi yang merupakan fungsi UI.
Pada line 112, nampilkan pilihan persen
Pada 113, text buat label teks
Pada line 114, ada row buat ngasih pilihan secara horizontal, rata tengah vertical.
Pada line 115, buat isi tiga pilihan persen

Di line 116-123, ada setiap pilihan di dalam row, isinya ada tombol pilihan radiobutton, aktif nilainya sama selectedTip, pas di klik muncul onTipSelected(percent), text nya buat nampilin nilai persennya.

Pada line 130, sama seperti sebelumnya
Pada line 131-134, buat ngasih fungsi isinya teks sama switch untuk aktif atau nonaktif opsi dari pembulatan
Pada line 135-141, ada row buat elemen secara horizontal, arrangement.SpaceBetween buat jarak maz antara text dan switch, Alignment.CenterVertically buat mensejajarkan elemen secara vertical Tengah.
Pada line 142-145, buat nampilin text , Modifier.weight(1f) buat ngisi ruang yang tersedia.
Pada line 146-148, buat switch on off checked buat status switch ngikutin nilai roundUp, onCheckedChange = onRoundChanged buat manggil fungsi kalau switchnya diganti.

Pada line 153, merupakan fungsi ngitung jumlah tip berdasarkan input.

- amount: Double → Jumlah tagihan.
- tipPercent: Double (default 15.0) → Persentase tip.
- roundUp: Boolean → Jika true, hasil tip akan dibulatkan ke atas.

Isinya di line 155-158 :

- Hitung tip: tip = tipPercent / 100 * amount
- Jika roundUp == true, bulatkan ke atas dengan ceil(tip)
- Format hasil jadi bentuk uang: NumberFormat.getCurrencyInstance().format(tip)

Pada line 161-167, buat pratinjau tampilannya

- @Preview(showBackground = true) → Menampilkan preview dengan latar belakang.
- Di dalamnya, memanggil TipCalculatorLayout() dengan tema TipCalculatorTheme

2. **Strings.xml**

Pada baris 1, deklarasi resources yang berisi kumpulan string yang digunakan di seluruh aplikasi.
Pada baris 2, berisi nama aplikasi "Tip Calculator" yang akan muncul di layar utama aplikasi.
Pada baris 3, bernama calculate_tip untuk judul proses menghitung tip.
Pada baris 4, berisi teks "Bill Amount", digunakan untuk label input jumlah tagihan.
Pada baris 5, berfungsi sebagai label untuk memilih persentase tip, berisi teks "Tip (%)".
Pada baris 6, berisi teks "Round up tip?", digunakan sebagai label untuk opsi apakah ingin membulatkan tip atau tidak.
Pada baris 7, berisi teks "Tip Amount: %1$s", di mana %1$s adalah placeholder yang akan diisi dengan jumlah tip yang dihitung saat aplikasi dijalankan.
Pada baris 8, resource ditutup

# MODUL 3 : Build A Scrollable List

## SOAL 1

Buatlah sebuah aplikasi Android menggunakan XML atau Jetpack Compose yang dapat menampilkan list dengan ketentuan berikut:

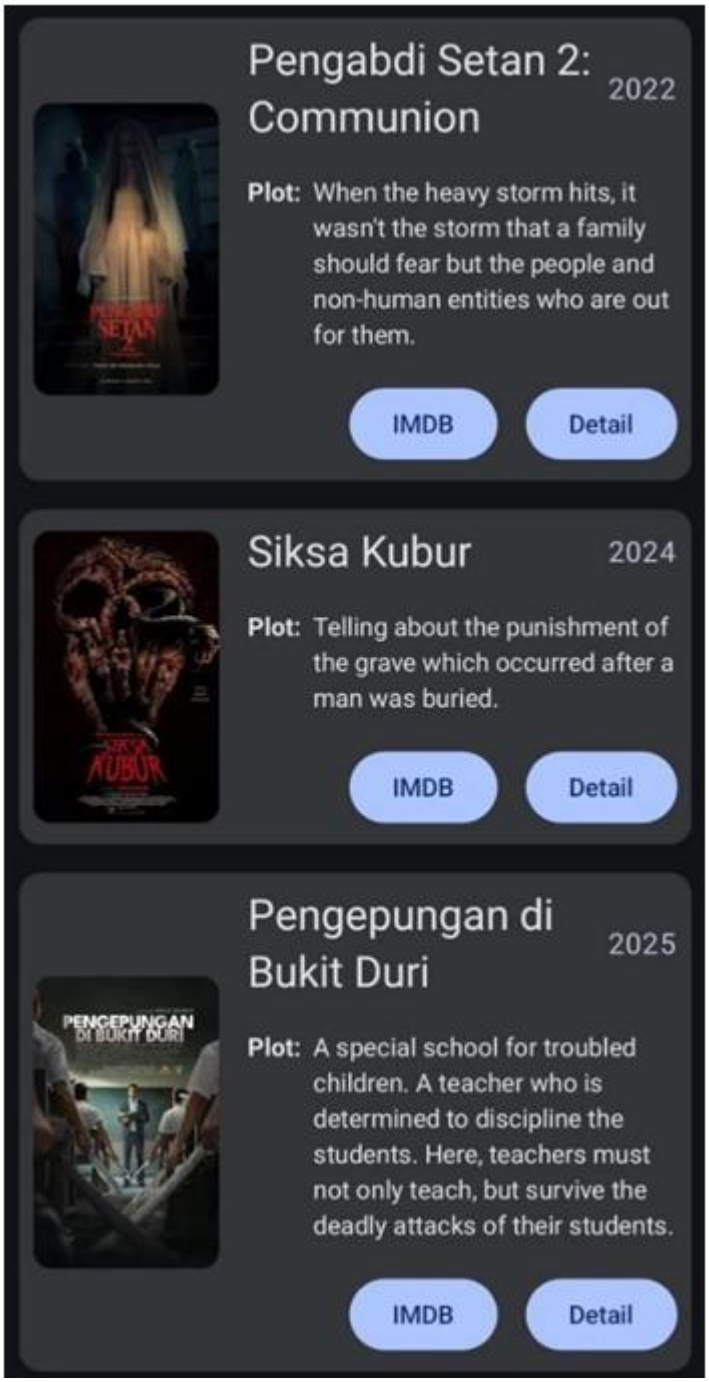
1. List menggunakan fungsi RecyclerView (XML) atau LazyColumn (Compose)
2. List paling sedikit menampilkan 5 item. Tema item yang ingin ditampilkan bebas
3. Item pada list menampilkan teks dan gambar sesuai dengan contoh di bawah
4. Terdapat 2 button dalam list, dengan fungsi berikut:
    a. Button pertama menggunakan intent eksplisit untuk membuka aplikasi atau browser lain
    b. b. Button kedua menggunakan Navigation component/intent untuk membuka laman detail item
5. Sudut item pada list dan gambar di dalam list melengkung atau rounded corner menggunakan Radius

6. Saat orientasi perangkat berubah/dirotasi, baik ke portrait maupun landscape, aplikasi responsif dan dapat menunjukkan list dengan baik. Data di dalam list tidak boleh hilang
7. Aplikasi menggunakan arsitektur single activity (satu activity memiliki beberapa fragment)
8. Aplikasi berbasis XML harus menggunakan ViewBinding

## SOAL 2

Mengapa RecyclerView masih digunakan, padahal RecyclerView memiliki kode yang panjang dan bersifat boiler-plate, dibandingkan LazyColumn dengan kode yang lebih singkat?

UI item list harus berisi 1 gambar, 2 button (intent eksplisit dan navigasi), dan 2 baris teks dan setiap baris memiliki 2 teks yang berbeda. Diusahakan agar desain UI item list menyerupai UI berikut:




*Gambar 8.Contoh UI Modul 3*

Desain UI laman detail bebas, tetapi diusahakan untuk mengikuti kaidah desain Material Design dan data item ditampilkan penuh di laman detail seperti contoh berikut:


*Gambar 9. Contoh UI Modul 3*

Jawaban 1.

## A. Source Code

### 1. MainActivity.kt

*Tabel 5. MainActivity/kt Modul3*

```
1  package com.example.gracemanhwa_picks
2
3  import android.os.Bundle
4  import androidx.activity.ComponentActivity
5  import androidx.activity.compose.setContent
6  import androidx.navigation.NavType
7  import androidx.navigation.compose.*
8  import androidx.navigation.navArgument
9  import com.example.gracemanhwa_picks.ui.theme.GracemanhwaPicksTheme
10
11 data class Manhwa(
12     val id: Int,
13     val title: String,
14     val author: String,
15     val description: String,
16     val imageRes: Int,
17     val url: String
18 )
19
20 val manhwaList = listOf(
21     Manhwa(
22         1,
23         "Solo Leveling",
24         "Chu-Gong",
25         "In a world where hunters — human warriors who possess supernatural abilities
       — must battle deadly monsters to protect all mankind from certain annihilation, a
       notoriously weak hunter named Sung Jin-woo finds himself in a seemingly endless
       struggle for survival. "
26             + "One day, after narrowly surviving an overwhelmingly powerful double
       dungeon that nearly wipes out his entire party, a mysterious program called the
       System chooses him as its sole player and in turn, gives him the unique ability to
       level up in strength. "
27             + "This is something no other hunter is able to do, as a hunter's abilities are
       set once they awaken. Jinwoo then sets out on a journey as he fights against all kinds
       of enemies, both man and monster, to discover the secrets of the dungeons and the
       true source of his powers. "
28             + "He soon discovers that he has been chosen to inherit the position of
```

```
Shadow Monarch, essentially turning him into an immortal necromancer who has absolute rule over the dead. "
            + "He is the only Monarch who fights to save humanity, as the other Monarchs are all trying to kill him and wipe out the humans.",
        R.drawable.solo_leveling,
        "https://www.tappytoon.com/en/book/solo-leveling-official"
    ),
    Manhwa(
        2,
        "Omniscient Reader",
        "SingNSong",
        "Kim Dokja is a young man leading a simple life, who has been the sole reader of a novel \"Three Ways to Survive in a Ruined World\" for 13 years of his life. "
            + "As he was reading the novel's final chapter, reality and the world of fiction started to merge, allowing him to appear at the beginning point of the story. "
            + "Being the only person who knew how the world could end, Kim Dokja is determined to create a different ending by solving and conquering various challenges, known as scenarios, which are operated by dokkaebi.",
        R.drawable.omniscient_reader,
        "https://www.webtoons.com/en/action/omniscient-reader/list?title_no=2154"
    ),
    Manhwa(
        3,
        "The Beginning After the End",
        "TurtleMe",
        "It follows the life of the late King Grey after his untimely and mysterious death. Reborn as Arthur Leywin, he seeks to correct his past mistakes in the vibrant new continent of Dicathen, a world of magic and fantastical creatures. "
            + "Equipped with the knowledge of a powerful king in his mid-thirties, Arthur navigates his new life as the magic-wielding child of two retired adventurers and gains purpose through each of his new experiences—something he lacked in his previous life. "
            + "When a kind dragon sacrifices her life to protect him, Arthur resolves to live a sincere, kind, and courageous life with those he loves. With the help of a lost elf princess and the Elven Kingdom of Elenoir, Arthur begins his long journey to find his true place in the world.\n\n"
            + "As the years pass, Arthur becomes more and more comfortable in this world, positioning himself as a young, but respected figure. However, deja-vu strikes as a war brews between Dicathen and the Vritra, a clan of banished deities now ruling over a faraway continent. "
            + "Arthur must rise as a leader, despite his fear of becoming the war-hardened monster he once was in his past life. "
            + "As the war rages on, Arthur discovers that he was not reborn to this world by chance...nor was he the only one.",
```

```
53          R.drawable.tbate,
54          "https://tapas.io/series/tbate-comic/info"
55      ),
56      Manhwa(
57          4,
58          "Eleceed",
59          "Jeho Son",
60          "Jiwoo is a kind-hearted young man who harnesses the lightning-quick reflexes
        of a cat to secretly make the world a better place – one saved little child or foster pet
        at a time. "
61                  + "Kayden is a secret agent on the run, who finds himself stuck in the body
        of a…um…decidedly fat old fluffy cat. "
62                  + "Together, armed with Jiwoo's superpowers and Kayden's uber-smarts,
        they're out to fight those forces who would let evil rule this world. "
63                  + "That is, if they can stand each other long enough to get the job done.",
64          R.drawable.eleceed,
65          "https://www.webtoons.com/en/action/eleceed/list?title_no=1571"
66      ),
67      Manhwa(
68          5,
69          "Killer Peter",
70          "Kim Junghyun",
71          "On the surface, Glory Hound is a simple human rights organization. In reality,
        the organization has some of the best assassins in the world, in charge of performing
        legendary.\n\n"
72                  + "One of their best members was simply known as Apostle Peter, and he
        retired in protest of the new leader, Raphael. However, resignations were not
        accepted, and Peter was soon ambushed. Despite his best efforts, he dies.\n\n"
73                  + "Instead of dying, though, Peter miraculously found himself back in his
        teenage body. He doesn't know why, but he knows one thing: he will destroy Glory
        Hound.",
74          R.drawable.killer_peter,
75          "https://www.webtoons.com/en/action/killer-peter/list?title_no=5816"
76      ),
77      Manhwa(
78          6,
79          "Player Who Can't Level Up",
80          "GaVinGe",
81          "When Kim Kigyu received his invitation to become a player (a unique-ability
        player, at that), he thought his struggles were over. But no matter how hard he tries,
        he just can't seem to get past level 1! "
82                  + "After five years of working as a guide on the lower floors of the tower,
        he's finally discovered his ability to link with "Egos" and raise his stats. "
83                  + "As his new skills unlock adventures in unexplored gates, Kigyu gets his
```

```
        chance to defy expectations and show the world that rank isn’t everything.",
84          R.drawable.player_cant_level_up,
85          "https://tapas.io/episode/2414063"
86      ),
87      Manhwa(
88          7,
89          "SSS-Class Revival Hunter",
90          "Shinnoa",
91          "After the Tower suddenly appeared, individuals who wished to pursue their
        personal values began to inhabit it, coming to be called \"hunters.\" "
92              + "Everyone had their own goals, but only a chosen few were
        acknowledged and given powerful skills by the mysterious structure. "
93              + "Kim Gong-Ja, a weak F-Class hunter without any skills, is envious of
        those who were blessed by the Tower. "
94              + "Letting his jealousy overcome him one day, Gong-Ja abruptly receives a
        S-Class skill that allows him to copy a skill from someone else—after they kill
        him.\n\n"
95              + "Sooner than he likes, Gong-Ja gets to test his newly acquired ability on
        the legendary hunter known as the Flame Emperor. "
96              + "As he is dying, Gong-Ja learns the evil truth about the man he once
        admired the most. "
97              + "Receiving another potent skill that allows him to revive and go back in
        time by 24 hours, Gong-Ja devises a plan to travel 11 years into the past to eliminate
        the Flame Emperor and cement himself as the world's best hunter.",
98          R.drawable.sss_class_hunter,
99          "https://www.mangaread.org/manga/sss-class-suicide-hunter/"
100     )
101 )
102
103 class MainActivity : ComponentActivity() {
104     override fun onCreate(savedInstanceState: Bundle?) {
105         super.onCreate(savedInstanceState)
106         setContent {
107             GracemanhwaPicksTheme {
108                 val navController = rememberNavController()
109                 NavHost(navController, startDestination = "list") {
110                     composable("list") {
111                         ListScreen(navController)
112                     }
113                     composable(
114                         "detail/{id}",
115                         arguments = listOf(navArgument("id") { type = NavType.IntType })
116                     ) { backStackEntry ->
117                         val id = backStackEntry.arguments?.getInt("id") ?: 0
```

```
                    val item = manhwaList.first { it.id == id }
                    DetailScreen(item)
                }
            }
        }
    }
}
```

*Tabel 10. Source Code Jawaban Soal 1*

**2. ListScreen.kt**

```
package com.example.gracemanhwa_picks

import android.content.Intent
import android.net.Uri
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ListScreen(navController: NavController) {
    val context = LocalContext.current

    Scaffold(
        topBar = {
            CenterAlignedTopAppBar(
                title = {
                    Text(
                        text = "Grz's Manhwa Picks",
                        style = MaterialTheme.typography.titleSmall
                    )
```

```
33              }
34          )
35      }
36  ) { innerPadding ->
37      LazyColumn(
38          modifier = Modifier
39              .padding(innerPadding)
40              .fillMaxSize()
41              .padding(8.dp)
42      ) {
43          items(manhwaList) { item ->
44              Card(
45                  shape = RoundedCornerShape(20.dp),
46                  modifier = Modifier
47                      .padding(8.dp)
48                      .fillMaxWidth()
49              ) {
50                  Column(modifier = Modifier.padding(8.dp)) {
51                      Image(
52                          painter = painterResource(id = item.imageRes),
53                          contentDescription = item.title,
54                          contentScale = ContentScale.Crop,
55                          modifier = Modifier
56                              .fillMaxWidth()
57                              .height(180.dp)
58                              .clip(RoundedCornerShape(16.dp))
59                      )
60                      Spacer(modifier = Modifier.height(8.dp))
61                      Text(
62                          item.title,
63                          style = MaterialTheme.typography.titleLarge
64                      )
65                      Text(
66                          "By ${item.author}",
67                          style = MaterialTheme.typography.bodyMedium
68                      )
69                      Row(
70                          modifier = Modifier
71                              .fillMaxWidth()
72                              .padding(top = 8.dp),
73                          horizontalArrangement = Arrangement.SpaceEvenly
74                      ) {
75                          Button(onClick = {
76                              val intent = Intent(Intent.ACTION_VIEW, Uri.parse(item.url))
```

```
                        context.startActivity(intent)
                    }) {
                        Text("Baca")
                    }
                    Button(onClick = {
                        navController.navigate("detail/${item.id}")
                    }) {
                        Text("Detail")
                    }
                }
            }
        }
    }
}
```

*Tabel 2. Source Code Jawaban Soal 1*

**3. DetailScreen.kt**

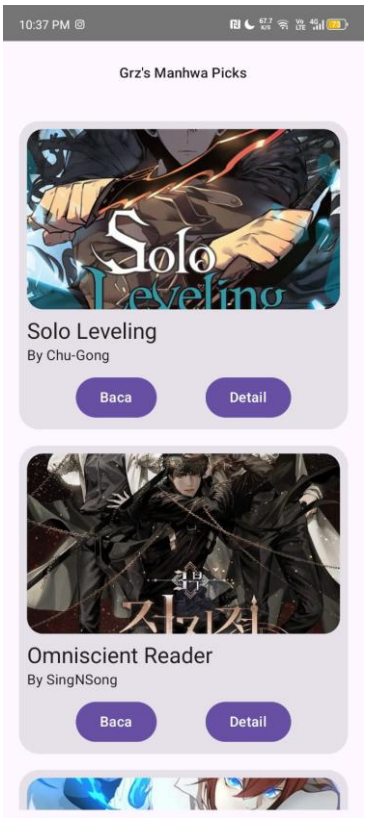
```
package com.example.gracemanhwa_picks

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp

@Composable
fun DetailScreen(item: Manhwa) {
    val scrollState = rememberScrollState()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(scrollState)
            .padding(16.dp)
```

```
    ) {
        Image(
            painter = painterResource(id = item.imageRes),
            contentDescription = item.title,
            modifier = Modifier
                .fillMaxWidth()
                .height(250.dp)
                .clip(RoundedCornerShape(16.dp)),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(item.title, style = MaterialTheme.typography.headlineMedium)
        Text("By ${item.author}", style =
MaterialTheme.typography.titleMedium)
        Spacer(modifier = Modifier.height(8.dp))
        Divider()
        Spacer(modifier = Modifier.height(8.dp))
        Text(item.description, style = MaterialTheme.typography.bodyLarge)
    }
}
```

**4.Theme.kt**

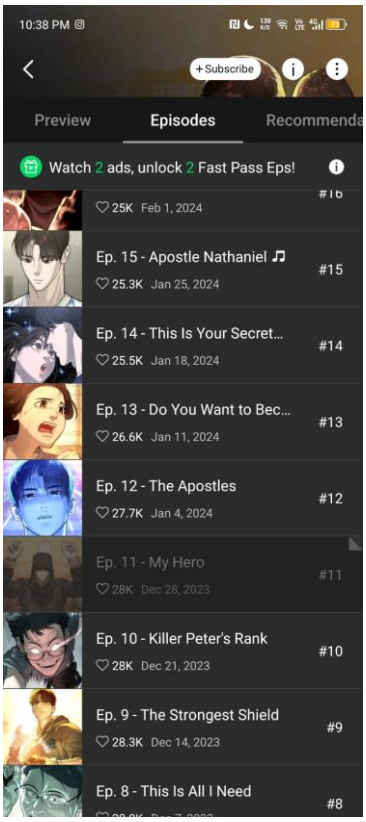
```
package com.example.gracemanhwa_picks.ui.theme

import androidx.compose.material3.*
import androidx.compose.runtime.Composable

@Composable
fun GracemanhwaPicksTheme(content: @Composable () -> Unit) {
    MaterialTheme(
        colorScheme = lightColorScheme(),
        typography = Typography(),
        content = content
    )
}
```

## B. Output Program




*Gambar 11. Output Modul 3*


*Gambar 12. Output Modul 3*




*Gambar 13. Output Modul 3*

## C. Pembahasan

### 1. MainActivity.kt:

Pada line 1, file ini berada di dalam package com.example.gracemanhwa_picks.
Pada line 3-9, mengimpor berbagai fungsionalitas untuk aplikasi:

- Line 3: Bundle untuk membawa data antar aktivitas.
- Line 4: ComponentActivity untuk aktivitas dengan Jetpack Compose.
- Line 5: setContent untuk menetapkan tampilan UI.
- Line 6: NavType untuk jenis data navigasi.
- Line 7: NavController untuk kontrol navigasi.
- Line 8: navArgument untuk mendefinisikan parameter navigasi.
- Line 9: GracemanhwaPicksTheme untuk tema aplikasi.

Pada line 11-18, didefinisikan sebuah data class bernama Manhwa. data class digunakan untuk membuat sebuah kelas yang hanya berfungsi untuk menyimpan data.

- val id: Int: Menyimpan ID manhwa, berupa bilangan bulat (Int).
- val title: String: Menyimpan judul manhwa, berupa teks (String).
- val author: String: Menyimpan nama pengarang manhwa, berupa teks (String).
- val description: String: Menyimpan deskripsi manhwa, berupa teks (String).

- val imageRes: Int: Menyimpan resource ID gambar yang digunakan untuk manhwa, berupa bilangan bulat (Int), biasanya menunjuk ke file gambar.
- val url: String: Menyimpan URL (tautan) untuk membaca manhwa, berupa teks (String).

Lalu selanjutnya ada val manhwa list yang ada manhwa() isinya list masing-masing property yang mencakup atau yang berisi : id, title, author, description, imageRes, url.

Begitu seterusnya lalu selanjutnya ada di line 103-119 :
Line 103: MainActivity adalah kelas utama aplikasi yang mewarisi dari ComponentActivity.
Line 104-105: onCreate dipanggil saat aktivitas dibuat, dan super.onCreate(savedInstanceState) memastikan aktivitas diinisialisasi dengan benar.
Line 106: setContent digunakan untuk menetapkan tampilan UI aplikasi dengan Jetpack Compose.
Line 107: GracemanhwaPicksTheme menetapkan tema aplikasi.
Line 108: navController dibuat untuk mengelola navigasi antar layar.
Line 109-110: NavHost mendefinisikan struktur navigasi, dengan layar awal "list".
Line 111-113: Menampilkan ListScreen saat navigasi ke "list".
Line 114-116: Menetapkan layar "detail/{id}" untuk menerima argumen id dan menavigasi ke detail.
Line 117-118: Mengambil id dari argumen navigasi dan memberi nilai default 0 jika tidak ada.
Line 119: Mencari item dari manhwaList yang memiliki id sesuai argumen.

2. **ListScreen.kt**

1 file berasa di package utama

3-18 buat import: ngatur halaman, ubah URL supaya bisa dibacam nampilin gambar, nyediakan layoutm komponen scrollable (kayak recyle view tpi ver compose), sudutnya agar bulat, import material 3, buat fungsi UI dengan @Composable, ngatur ukuran dan posisi dengan Modifier, potong tampilan jadi sudut bulat pakai clip, atur skala gambar biar pas di tempatnya, ambil context Android buat buka link, ambil gambar dari drawable pakai resource, atur ukuran elemen dengan satuan dp, dan terakhir, navigasi antar layar pakai NavController.

Line 20, kode ini akan menggunakan API eksperimental -- padahal belum stabil tapi supaya bisa dipakai makanya pakai ini
Line 21, ngasih tau list screen itu compoable
Line 22, definisi fungsi yang ada parameter dari navController buat navigasi layar di app
Line 23, buat dapetin contect yang dibutuhkan
Line 25, ngasih layout dasar struktur umum top bar, bottom bar, dan konten utama
Line 26, top bar ada center aligned

Line 27, judul posisi tengah.
Line 28-31, judul titlenya kecil

LazyColumn, yang memuat setiap item hanya saat dibutuhkan untuk efisiensi. Setiap item dibungkus dalam Card dengan sudut melengkung dan padding, yang berisi gambar, judul manhwa, nama pengarang, dan dua tombol. Gambar ditampilkan dengan efek crop agar sesuai dengan ukuran yang telah ditentukan dan memiliki sudut melengkung. Setelah gambar, terdapat teks yang menampilkan judul dan pengarang manhwa dengan gaya teks yang telah disesuaikan. Di bagian bawah, terdapat dua tombol: tombol pertama untuk membuka URL manhwa di browser menggunakan Intent, dan tombol kedua untuk menavigasi ke layar detail manhwa dengan NavController

3. **DetailScreen.kt**

Pada line 1 itu menunjukan kalau file ada di package utama pada aplikasi

Line 3-14 itu fungsinya untuk import :

- 3 nampilin gambar
- 4 layout dasar
- 5 ingat status scroll
- 6 sudut lengkung
- 7 column scroll vertikal
- 8 material design 3
- 9 nandain kalau composable
- 10 modifier -- tata letak, uk., padding, dll
- 11 motong bentuk tampilan
- 12 ngatur gambar ditampilkannya gimana
- 13 membuat gambar dari file lain
- 14 untuk ukuran supaya konsisten

16 buat nandain itu compose

Line 17 nampilin detail manhwa

18 supaya bisa scroll + diingat

20 isinya column buat nyusus komponen vertikal

21 fill max size berarti ukuran layarnya penuh

22 bisa di scroll ke bawah + diingat

23 jarak

26  image -- buat nampilin gambar

27-33 ngambil gambar dari file, ada deskripsi konten, di modifier ada lebar gambar yang ngikutin lebar layar, tingginya 250, gambarnya sudutnya dibulatkan, gambarnya dipastiin penuh

35 beri jarak vertikal 16dp per elemen

36 nampilim judul ukurannya medium

37 nampilin nama author tulisannya sedang juga

38 jarak setelah garis 8dp

39 divider buat garis horizontal tipis

40 jarak lagi setelah garis 8dp

41  eskripsi panjangnya

4. **Theme.kt**

Pada line 1 ada package untuk tema dari aplikasi
Pada line 3-4 digunakan untuk import elemen compose material 3 dan composable function
Pada line 6 ada composable -- untuk UI
Lalu line 7 untuk menerima composable lain buat parameternya, jadi UI nya dibungkus.
Pada line 8 ada material theme -- ini fungsi untuk tema
Line 9 ada skema warnanya di mode terang
Line 10 untuk mengatur gaya huruf
Line 11 terdapat content untuk menampilkan isi UI yang dibungkus

**Jawaban nomor 2 .**

Meskipun LazyColumn menawarkan kode yang lebih singkat dan deklaratif dibandingkan RecyclerView, ada beberapa alasan mengapa RecyclerView masih banyak digunakan dalam pengembangan Android. Pertama, banyak aplikasi yang sudah ada menggunakan RecyclerView, dan migrasi ke LazyColumn memerlukan upaya besar. Kedua, RecyclerView memberikan kontrol lebih besar dalam hal kustomisasi, seperti animasi item dan dekorasi, yang penting untuk aplikasi yang membutuhkan tampilan dan interaksi yang lebih spesifik. Selain itu, RecyclerView didukung oleh banyak pustaka pihak ketiga yang membantu mempercepat pengembangan dan menambahkan fungsionalitas tambahan. Pada kasus daftar yang sangat besar atau kompleks, RecyclerView sering kali memberikan kinerja yang lebih

baik. Terakhir, banyak pengembang Android yang sudah berpengalaman dengan RecyclerView dan lebih nyaman menggunakannya, terutama dalam proyek yang memerlukan kontrol detail. Meski LazyColumn lebih modern, RecyclerView tetap relevan, terutama dalam proyek besar atau yang membutuhkan kustomisasi tinggi.

# MODUL 4 : View Model and Debugging

## SOAL 1

Lanjutkan aplikasi Android berbasis XML dan Jetpack Compose yang sudah dibuat pada Modul 3 dengan menambahkan modifikasi sesuai ketentuan berikut:

a. Buatlah sebuah ViewModel untuk menyimpan dan mengelola data dari list item. Data tidak boleh disimpan langsung di dalam Fragment atau Activity.

b. Gunakan ViewModelFactory dalam pembuatan ViewModel

c. Gunakan StateFlow untuk mengelola event onClick dan data list item dari ViewModel ke Fragment

d. gunakan logging untuk event berikut:

a. Log saat data item masuk ke dalam list

b. Log saat tombol Detail dan tombol Explicit Intent ditekan

c. Log data dari list yang dipilih ketika berpindah ke halaman Detail

e. Gunakan tool Debugger di Android Studio untuk melakukan debugging pada aplikasi. Cari setidaknya satu breakpoint yang relevan dengan aplikasi. Lalu, gunakan fitur Step Into, Step Over, dan Step Out. Setelah itu, jelaskan fungsi Debugger, cara menggunakan Debugger, serta fitur Step Into, Step Over, dan Step Out

### A. Source Code

*Tabel 6. Manhwa.kt Modul 4*

```
1  package com.example.gracemanhwa_picks.data
2
3  data class Manhwa(
4      val id: Int,
5      val title: String,
6      val author: String,
7      val description: String,
8      val imageRes: Int,
```

```
    val url: String
)
```

*Tabel 7.ManhwaRepository.kt*

```
package com.example.gracemanhwa_picks.data

import com.example.gracemanhwa_picks.R

class ManhwaRepository {
    fun getManhwas(): List<Manhwa> {
        return listOf(
            Manhwa(
                id = 1,
                title = "Solo Leveling",
                author = "Chu-Gong",
                description = "In a world where hunters — human warriors who possess supernatural abilities — must battle deadly monsters to protect all mankind from certain annihilation, a notoriously weak hunter named Sung Jin-woo finds himself in a seemingly endless struggle for survival. "
                        + "One day, after narrowly surviving an overwhelmingly powerful double dungeon that nearly wipes out his entire party, a mysterious program called the System chooses him as its sole player and in turn, gives him the unique ability to level up in strength. "
                        + "This is something no other hunter is able to do, as a hunter's abilities are set once they awaken. Jinwoo then sets out on a journey as he fights against all kinds of enemies, both man and monster, to discover the secrets of the dungeons and the true source of his powers. "
                        + "He soon discovers that he has been chosen to inherit the position of Shadow Monarch, essentially turning him into an immortal necromancer who has absolute rule over the dead. "
                        + "He is the only Monarch who fights to save humanity, as the other Monarchs are all trying to kill him and wipe out the humans.",
```

```
17                  imageRes =
    R.drawable.solo_leveling,
18                  url =
    "https://www.tappytoon.com/en/book/solo-leveling-
    official"
19              ),
20              Manhwa(
21                  id = 2,
22                  title = "Omniscient Reader",
23                  author = "SingNSong",
24                  description = "Kim Dokja is a young
    man leading a simple life, who has been the sole
    reader of a novel \"Three Ways to Survive in a
    Ruined World\" for 13 years of his life. "
25                          + "As he was reading the
    novel's final chapter, reality and the world of
    fiction started to merge, allowing him to appear at
    the beginning point of the story. "
26                          + "Being the only person
    who knew how the world could end, Kim Dokja is
    determined to create a different ending by solving
    and conquering various challenges, known as
    scenarios, which are operated by dokkaebi.",
27                  imageRes =
    R.drawable.omniscient_reader,
28                  url =
    "https://www.webtoons.com/en/action/omniscient-
    reader/list?title_no=2154"
29              ),
30              Manhwa(
31                  id = 3,
32                  title = "The Beginning After the
    End",
33                  author = "TurtleMe",
34                  description = "It follows the life
    of the late King Grey after his untimely and
    mysterious death. Reborn as Arthur Leywin, he seeks
    to correct his past mistakes in the vibrant new
    continent of Dicathen, a world of magic and
    fantastical creatures. "
35                          + "Equipped with the
    knowledge of a powerful king in his mid-thirties,
    Arthur navigates his new life as the magic-wielding
    child of two retired adventurers and gains purpose
    through each of his new experiences—something he
```

```
    lacked in his previous life. "
36                          + "When a kind dragon
    sacrifices her life to protect him, Arthur resolves
    to live a sincere, kind, and courageous life with
    those he loves. With the help of a lost elf
    princess and the Elven Kingdom of Elenoir, Arthur
    begins his long journey to find his true place in
    the world.\n\n"
37                          + "As the years pass,
    Arthur becomes more and more comfortable in this
    world, positioning himself as a young, but
    respected figure. However, deja-vu strikes as a war
    brews between Dicathen and the Vritra, a clan of
    banished deities now ruling over a faraway
    continent. "
38                          + "Arthur must rise as a
    leader, despite his fear of becoming the war-
    hardened monster he once was in his past life. "
39                          + "As the war rages on,
    Arthur discovers that he was not reborn to this
    world by chance...nor was he the only one.",
40                  imageRes = R.drawable.tbate,
41                  url =
    "https://tapas.io/series/tbate-comic/info"
42              ),
43              Manhwa(
44                  id = 4,
45                  title = "Eleceed",
46                  author = "Jeho Son",
47                  description = "Jiwoo is a kind-
    hearted young man who harnesses the lightning-quick
    reflexes of a cat to secretly make the world a
    better place - one saved little child or foster pet
    at a time. "
48                          + "Kayden is a secret agent
    on the run, who finds himself stuck in the body of
    a…um…decidedly fat old fluffy cat. "
49                          + "Together, armed with
    Jiwoo's superpowers and Kayden's uber-smarts,
    they're out to fight those forces who would let
    evil rule this world. "
50                          + "That is, if they can
    stand each other long enough to get the job done.",
51                  imageRes = R.drawable.eleceed,
52                  url =
```

```
    "https://www.webtoons.com/en/action/eleceed/list?title_no=1571"
53              ),
54              Manhwa(
55                  id = 5,
56                  title = "Killer Peter",
57                  author = "Kim Junghyun",
58                  description = "On the surface, Glory Hound is a simple human rights organization. In reality, the organization has some of the best assassins in the world, in charge of performing legendary.\n\n"
59                          + "One of their best members was simply known as Apostle Peter, and he retired in protest of the new leader, Raphael. However, resignations were not accepted, and Peter was soon ambushed. Despite his best efforts, he dies.\n\n"
60                          + "Instead of dying, though, Peter miraculously found himself back in his teenage body. He doesn't know why, but he knows one thing: he will destroy Glory Hound.",
61                  imageRes = R.drawable.killer_peter,
62                  url = "https://www.webtoons.com/en/action/killer-peter/list?title_no=5816"
63              ),
64              Manhwa(
65                  id = 6,
66                  title = "Player Who Can't Level Up",
67                  author = "GaVinGe",
68                  description = "When Kim Kigyu received his invitation to become a player (a unique-ability player, at that), he thought his struggles were over. But no matter how hard he tries, he just can’t seem to get past level 1! "
69                          + "After five years of working as a guide on the lower floors of the tower, he’s finally discovered his ability to link with “Egos” and raise his stats. "
70                          + "As his new skills unlock adventures in unexplored gates, Kigyu gets his chance to defy expectations and show the world that rank isn’t everything.",
```

```
71                  imageRes =
    R.drawable.player_cant_level_up,
72                  url =
    "https://tapas.io/episode/2414063"
73              ),
74              Manhwa(
75                  id = 7,
76                  title = "SSS-Class Revival Hunter",
77                  author = "Shinnoa",
78                  description = "After the Tower
    suddenly appeared, individuals who wished to pursue
    their personal values began to inhabit it, coming
    to be called \"hunters.\" "
79                          + "Everyone had their own
    goals, but only a chosen few were acknowledged and
    given powerful skills by the mysterious structure.
    "
80                          + "Kim Gong-Ja, a weak F-
    Class hunter without any skills, is envious of
    those who were blessed by the Tower. "
81                          + "Letting his jealousy
    overcome him one day, Gong-Ja abruptly receives a
    S-Class skill that allows him to copy a skill from
    someone else—after they kill him.\n\n"
82                          + "Sooner than he likes,
    Gong-Ja gets to test his newly acquired ability on
    the legendary hunter known as the Flame Emperor. "
83                          + "As he is dying, Gong-Ja
    learns the evil truth about the man he once admired
    the most. "
84                          + "Receiving another potent
    skill that allows him to revive and go back in time
    by 24 hours, Gong-Ja devises a plan to travel 11
    years into the past to eliminate the Flame Emperor
    and cement himself as the world's best hunter.",
85                  imageRes =
    R.drawable.sss_class_hunter,
86                  url =
    "https://www.mangaread.org/manga/sss-class-suicide-
    hunter/"
87              )
88          )
        }
89  }
90
```

*Tabel 8. MainActivity.kt Modul 4*

```
01  package com.example.gracemanhwa_picks
2
3   import android.os.Bundle
4   import androidx.activity.ComponentActivity
5   import androidx.activity.compose.setContent
6   import androidx.activity.viewModels
7   import androidx.compose.runtime.Composable
8   import androidx.navigation.NavType
9   import androidx.navigation.compose.NavHost
10  import androidx.navigation.compose.composable
    import
    androidx.navigation.compose.rememberNavController
    import androidx.navigation.navArgument
    import
    com.example.gracemanhwa_picks.data.ManhwaRepository
    import
    com.example.gracemanhwa_picks.ui.DetailScreen
    import com.example.gracemanhwa_picks.ui.ListScreen
    import
    com.example.gracemanhwa_picks.ui.GracemanhwaPicksTh
    eme
    import
    com.example.gracemanhwa_picks.ui.ViewModel.ManhwaVi
    ewModel
    import
    com.example.gracemanhwa_picks.ui.ViewModel.ViewMode
    lFactory

    class MainActivity : ComponentActivity() {

        private val viewModel: ManhwaViewModel by
    viewModels {
            ViewModelFactory(ManhwaRepository())
        }

        override fun onCreate(savedInstanceState:
    Bundle?) {
            super.onCreate(savedInstanceState)
            setContent {
                GracemanhwaPicksTheme {
                    ManhwaApp(viewModel = viewModel)
                }
```

```
        }
    }
}

@Composable
fun ManhwaApp(viewModel: ManhwaViewModel) {
    val navController = rememberNavController()
    NavHost(navController = navController,
startDestination = "list") {
        composable("list") {
            ListScreen(navController =
navController, viewModel = viewModel)
        }

        composable(
            route = "detail/{id}",
            arguments = listOf(navArgument("id") {
type = NavType.IntType })
        ) { backStackEntry ->
            val id =
backStackEntry.arguments?.getInt("id") ?: 0
            viewModel.getManhwaById(id)?.let { item
->
                DetailScreen(item = item)
            }
        }
    }
}
```

*Tabel 9. ManhwaViewModel.kt*

```
1   package com.example.gracemanhwa_picks.ui.ViewModel
2
3   import android.util.Log
4   import androidx.lifecycle.ViewModel
5   import com.example.gracemanhwa_picks.data.Manhwa
6   import
7   com.example.gracemanhwa_picks.data.ManhwaRepository
8   import kotlinx.coroutines.flow.MutableStateFlow
9   import kotlinx.coroutines.flow.StateFlow
10
    class ManhwaViewModel(private val repository:
```

```
ManhwaRepository): ViewModel() {

    private val _manhwas =
MutableStateFlow<List<Manhwa>>(emptyList())
    val manhwas: StateFlow<List<Manhwa>> get() =
_manhwas

    init {
        loadManhwas()
    }

    private fun loadManhwas() {
        _manhwas.value = repository.getManhwas()
        Log.d("Manhwa ViewModel", "Manhwa data
loaded into the list.")
    }

    fun getManhwaByIdId(id: Int): Manhwa? {
        val manhwa = _manhwas.value.firstOrNull {
it.id == id}
        if (manhwa != null) {
            Log.d("Manhwa ViewModel", "Navigating
to Detail for: ${manhwa.title}")
        }
        return manhwa
    }
}
```

*Tabel 10. ViewModelFactory.kt*

```
01  package com.example.gracemanhwa_picks.ui.ViewModel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import
6   com.example.gracemanhwa_picks.data.ManhwaRepository
7
8   class ViewModelFactory(private val repository:
9   ManhwaRepository) : ViewModelProvider.Factory {
10      override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
        if
    (modelClass.isAssignableFrom(ManhwaViewModel::class
    .java)) {
            @Suppress("UNCHECKED_CAST")
```

```
            return ManhwaViewModel(repository) as T
        }
        throw IllegalArgumentException("Unknown
ViewModel class")
    }
}
```

*Tabel 11. DetailScreen.kt*

```
01  package com.example.gracemanhwa_picks.ui
2
3   import androidx.compose.foundation.Image
4   import androidx.compose.foundation.layout.*
5   import
6   androidx.compose.foundation.rememberScrollState
7   import
8   androidx.compose.foundation.shape.RoundedCornerShap
    e
9   import androidx.compose.foundation.verticalScroll
10  import androidx.compose.material3.*
11  import androidx.compose.runtime.Composable
12  import androidx.compose.ui.Modifier
13  import androidx.compose.ui.draw.clip
14  import androidx.compose.ui.layout.ContentScale
15  import androidx.compose.ui.res.painterResource
16  import androidx.compose.ui.unit.dp
17  import com.example.gracemanhwa_picks.data.Manhwa
18
19  @Composable
20  fun DetailScreen(item: Manhwa) {
21      val scrollState = rememberScrollState()
22
23      Column(
24          modifier = Modifier
25              .fillMaxSize()
26              .verticalScroll(scrollState)
27              .padding(16.dp)
28      ) {
29          Image(
30              painter = painterResource(id =
    item.imageRes),
31              contentDescription = item.title,
32              modifier = Modifier
33                  .fillMaxWidth()
```

```
34                  .height(250.dp)
35                  .clip(RoundedCornerShape(16.dp)),
36              contentScale = ContentScale.Crop
37          )
38          Spacer(modifier = Modifier.height(16.dp))
39          Text(item.title, style =
    MaterialTheme.typography.headlineMedium)
40          Text("By ${item.author}", style =
    MaterialTheme.typography.titleMedium)
41          Spacer(modifier = Modifier.height(8.dp))
42          Divider()
43          Spacer(modifier = Modifier.height(8.dp))
44          Text(item.description, style =
    MaterialTheme.typography.bodyLarge)
45      }
46  }
```

*Tabel 12. ListScreen.kt Modul 4*

```
01  package com.example.gracemanhwa_picks.ui
2
3   import androidx.compose.foundation.Image
4   import androidx.compose.foundation.layout.*
5   import
6   androidx.compose.foundation.rememberScrollState
7   import
8   androidx.compose.foundation.shape.RoundedCornerShap
    e
9   import androidx.compose.foundation.verticalScroll
10  import androidx.compose.material3.*
11  import androidx.compose.runtime.Composable
12  import androidx.compose.ui.Modifier
13  import androidx.compose.ui.draw.clip
14  import androidx.compose.ui.layout.ContentScale
15  import androidx.compose.ui.res.painterResource
16  import androidx.compose.ui.unit.dp
17  import com.example.gracemanhwa_picks.data.Manhwa
18
19  @Composable
20  fun DetailScreen(item: Manhwa) {
21      val scrollState = rememberScrollState()
```

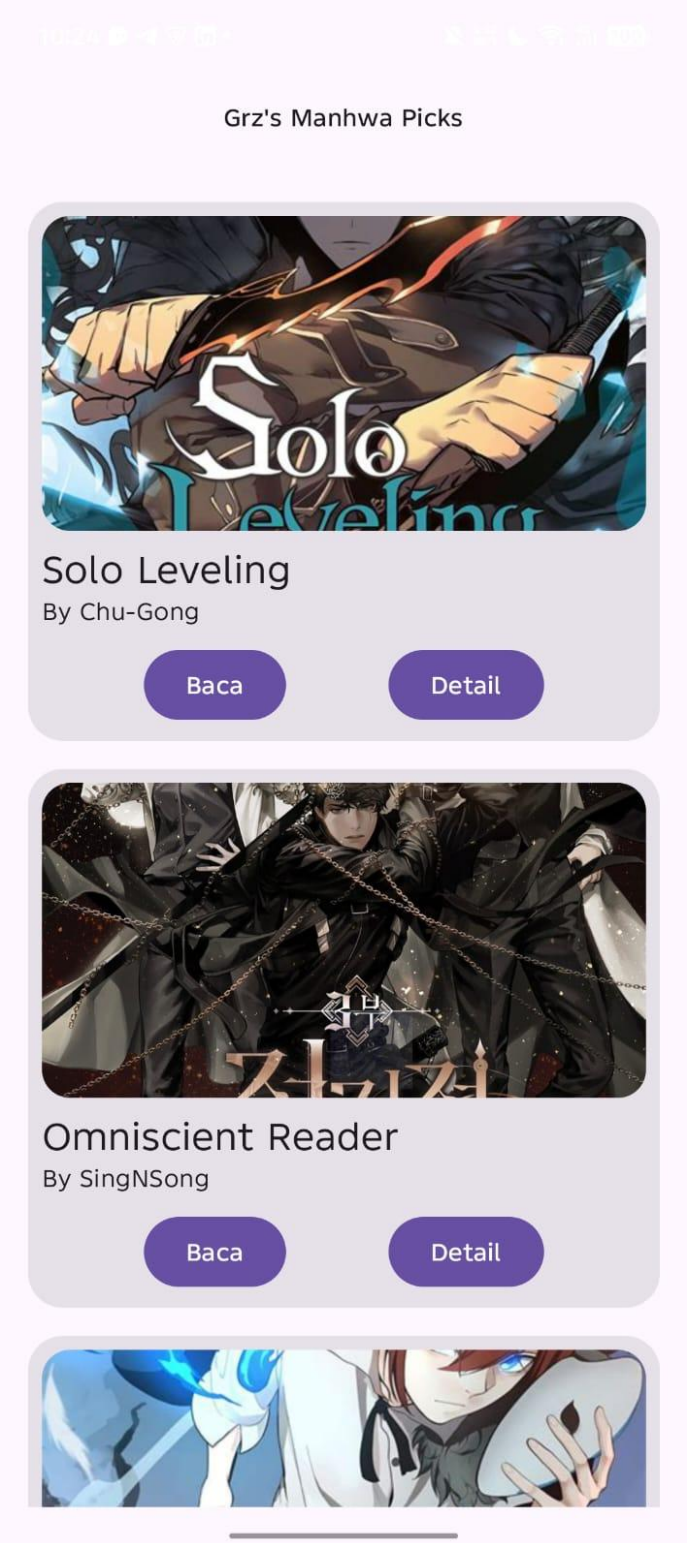
```
22      Column(
23          modifier = Modifier
24              .fillMaxSize()
25              .verticalScroll(scrollState)
26              .padding(16.dp)
27      ) {
28          Image(
29              painter = painterResource(id =
    item.imageRes),
30              contentDescription = item.title,
31              modifier = Modifier
32                  .fillMaxWidth()
33                  .height(250.dp)
34                  .clip(RoundedCornerShape(16.dp)),
35              contentScale = ContentScale.Crop
36          )
37          Spacer(modifier = Modifier.height(16.dp))
38          Text(item.title, style =
    MaterialTheme.typography.headlineMedium)
39          Text("By ${item.author}", style =
    MaterialTheme.typography.titleMedium)
40          Spacer(modifier = Modifier.height(8.dp))
41          Divider()
42          Spacer(modifier = Modifier.height(8.dp))
44          Text(item.description, style =
    MaterialTheme.typography.bodyLarge)
45      }
46  }
```

*Tabel 13. Theme.kt Modul 4*

```
01  package com.example.gracemanhwa_picks.ui
2
3   import androidx.compose.material3.*
4   import androidx.compose.runtime.Composable
5
6   @Composable
7   fun GracemanhwaPicksTheme(content: @Composable () -
    > Unit) {
8       MaterialTheme(
9           colorScheme = lightColorScheme(),
10          typography = Typography(),
```

```
11          content = content
12      )
13  }
```

## B. Output Program



*Gambar 14. Output Program Modul 4*

## C. Pembahasan

Manhwa.kt untuk definisi struktur data untuk objek manhwa. Pada baris pertama hingga ketiga, package dan import tidak dituliskan karena file ini tidak memerlukan import eksternal. Deklarasi data class Manhwa dimulai pada baris keempat dengan parameter seperti id, title, author, description, imageRes, dan url. Tipe data yang digunakan seluruhnya bersifat eksplisit, seperti Int untuk id dan imageRes, serta String untuk properti lainnya. Struktur ini memungkinkan penyimpanan berbagai informasi penting yang akan digunakan dalam tampilan daftar dan detail manhwa.

ManhwaRepository.kt digunakan sebagai penyedia data manhwa. Package dideklarasikan pada awal, dan import untuk resource R digunakan agar dapat mengakses gambar manhwa dari drawable. Fungsi utama di dalam file ini adalah getManhwas() yang berada pada baris keenam. Fungsi ini mengembalikan list yang terdiri dari beberapa objek Manhwa. Masing-masing objek Manhwa diinisialisasi dengan data seperti judul, penulis, deskripsi panjang, resource gambar, dan URL. Setiap manhwa memiliki deskripsi unik yang menjelaskan latar belakang cerita dan karakter utama, serta tujuan atau konflik dalam kisah tersebut. Data yang tersedia sangat lengkap, mendukung kebutuhan aplikasi untuk menampilkan informasi manhwa secara mendetail.

MainActivity.kt memulai aktivitas utama aplikasi Android. Package serta import berbagai komponen Compose dan ViewModel dideklarasikan di awal. Komponen ComponentActivity diturunkan ke class MainActivity, dan pada baris kesebelas dilakukan inisialisasi viewModel menggunakan delegate by viewModels, disertai factory ViewModelFactory yang menerima instance ManhwaRepository. Dalam fungsi onCreate, method setContent dipanggil untuk mengatur konten tampilan dengan tema GracemanhwaPicksTheme. Fungsi ManhwaApp digunakan sebagai root composable. Di dalam fungsi ManhwaApp, controller navigasi diciptakan menggunakan rememberNavController. Route "list" diatur untuk menampilkan ListScreen, sementara route "detail/{id}" akan mengambil argumen id dan menampilkan DetailScreen berdasarkan item yang sesuai dari ViewModel. Pendekatan ini mendukung arsitektur single-activity dengan navigasi antar composable yang efisien.

ManhwaViewModel.kt berisi logika ViewModel untuk mengelola data manhwa. Package dan import mendeklarasikan penggunaan ViewModel, logging, dan StateFlow. Kelas ManhwaViewModel menerima parameter repository untuk mengakses data. Di dalamnya, variabel _manhwas bertipe MutableStateFlow digunakan untuk menyimpan data secara mutable, sedangkan variabel publik manhwas bertipe StateFlow hanya menyediakan akses baca. Proses pengambilan data dimulai

dari fungsi init yang langsung memanggil loadManhwas(). Fungsi ini mengambil data dari repository dan mengisi _manhwas dengan daftar manhwa. Logging dilakukan untuk mencatat bahwa data telah dimuat. Fungsi getManhwaById menerima parameter id dan mencari objek manhwa dengan ID tersebut. Jika ditemukan, logging mencatat navigasi ke halaman detail berdasarkan judul manhwa yang dipilih.

ViewModelFactory.kt bertugas menyediakan mekanisme pembuatan ViewModel dengan parameter. Package dan import mendefinisikan penggunaan ViewModel dan ViewModelProvider. Kelas ViewModelFactory menerima parameter repository. Implementasi fungsi create digunakan untuk menghasilkan objek ManhwaViewModel jika modelClass yang diminta sesuai. Jika tidak cocok, maka dilempar pengecualian, memastikan ViewModel dapat menerima parameter eksternal dengan cara yang sesuai standar Android.

DetailScreen.kt merupakan composable untuk menampilkan halaman detail manhwa. Pada bagian awal, dilakukan import terhadap komponen layout seperti Column, Image, Text, dan Spacer, serta fungsi pendukung seperti painterResource dan ContentScale. Fungsi DetailScreen menerima parameter item bertipe Manhwa. Scroll state disiapkan agar tampilan dapat digulir secara vertikal. Layout utama menggunakan Column dengan modifier untuk ukuran penuh, scrollable, dan padding. Di dalamnya, gambar manhwa ditampilkan menggunakan Image dengan bentuk rounded dan skala cropping agar memenuhi tampilan horizontal. Selanjutnya, ditampilkan judul manhwa dengan gaya teks headline, disusul nama penulis dengan gaya teks title. Setelah itu, deskripsi panjang manhwa ditampilkan dengan pemisah berupa Divider dan jarak antar elemen dengan Spacer.

ListScreen.kt seharusnya berisi tampilan daftar manhwa, namun isi dari file yang ada adalah salinan DetailScreen.kt, menunjukkan kemungkinan kesalahan duplikasi konten saat penulisan laporan. Karena strukturnya identik, penjelasannya pun sama seperti pada DetailScreen, dan seharusnya diganti dengan kode yang menampilkan daftar manhwa dalam bentuk list item yang dapat ditekan untuk berpindah ke detail.

Theme.kt berfungsi untuk mengatur tema aplikasi. Package dan import mengarah ke penggunaan Material3 dan composable function. Fungsi GracemanhwaPicksTheme menerima lambda composable sebagai parameter. Di dalamnya, MaterialTheme digunakan dengan skema warna terang dan tipografi default. Fungsi ini membungkus semua konten aplikasi agar tetap konsisten secara tampilan dan nuansa visual.

## SOAL 2

Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

**Jawaban Soal 2**

Dalam arsitektur aplikasi Android, Application class merupakan komponen inti yang pertama kali diinisialisasi oleh sistem saat aplikasi dijalankan. Kelas ini hanya dibuat satu kali dan tetap aktif selama siklus hidup aplikasi berlangsung, sehingga sangat cocok digunakan untuk mengelola inisialisasi global. Salah satu fungsi utamanya adalah melakukan konfigurasi awal terhadap komponen atau pustaka yang digunakan di seluruh aplikasi, seperti Firebase, Retrofit, Room, atau library logging. Selain itu, Application class juga dapat dimanfaatkan untuk menyimpan state atau objek yang bersifat global, seperti repository, container dependency injection, atau konfigurasi tertentu yang diperlukan oleh berbagai aktivitas dan komponen UI.

Dengan membuat subclass dari Application dan meng-override fungsi onCreate(), proses inisialisasi yang penting dapat dilakukan sebelum aktivitas atau fragment pertama ditampilkan. Contohnya, repository atau service bisa diinisialisasi satu kali di sini, lalu disuntikkan ke dalam ViewModel melalui ViewModelFactory. Application juga bisa digunakan untuk memantau perilaku aplikasi secara keseluruhan, seperti mencatat log aktivitas, mendeteksi crash, atau mengelola analitik pengguna. Agar class ini dikenali oleh sistem Android, nama subclass Application perlu ditulis di atribut android:name pada file AndroidManifest.xml.

# MODUL 5 : Connect to the Internet

## SOAL 1

Lanjutkan aplikasi Android yang sudah dibuat pada Modul 4 dengan menambahkan modifikasi sesuai ketentuan berikut:

a. Gunakan networking library seperti Retrofit atau Ktor agar aplikasi dapat mengambil data dari remote API. Dalam penggunaan networking library, sertakan

generic response untuk status dan error handling pada API dan Flow untuk data stream.

b. Gunakan KotlinX Serialization sebagai library JSON.

c. Gunakan library seperti Coil atau Glide untuk image loading.

d. API yang digunakan pada modul ini bebas, contoh API gratis The Movie Database (TMDB) API yang menampilkan data film. Berikut link dokumentasi API: https://developer.themoviedb.org/docs/getting-started

e. Implementasikan konsep data persistence (misalnya offline-first app, pengaturan dark/light mode, fitur favorite, dll)

f. Gunakan caching strategy pada Room..

g. Untuk Modul 5, bebas memilih UI yang ingin digunakan, antara berbasis XML atau Jetpack Compose. Aplikasi harus mempertahankan fitur-fitur yang dibuat pada modul sebelumnya.

## A. Source Code

*Tabel 14. MovieDao.kt Modul 5*

```
01  package com.example.movielist.data.local.dao
2
3   import androidx.room.Dao
4   import androidx.room.Insert
5   import androidx.room.OnConflictStrategy
6   import androidx.room.Query
7   import
8   com.example.movielist.data.local.entities.MovieEnti
9   ty
10
11  @Dao
12  interface MovieDao {
13      @Insert(onConflict =
    OnConflictStrategy.REPLACE)
14      suspend fun insertAllMovies(movies:
15  List<MovieEntity>)

16      @Query("SELECT * FROM movies ORDER BY
    popularity DESC")
17      suspend fun getAllMovies(): List<MovieEntity>
18
19      @Query("DELETE FROM movies")
```

```
    suspend fun clearAllMovies()
}
```

*Tabel 15. AppDatabase.kt Modul 5*

```
package com.example.movielist.data.local.database

import androidx.room.Database
import androidx.room.RoomDatabase
import com.example.movielist.data.local.dao.MovieDao
import
com.example.movielist.data.local.entities.MovieEntit
y

@Database(entities = [MovieEntity::class], version =
1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun movieDao(): MovieDao

    companion object {
        const val DATABASE_NAME = "tmdb_app_db"
    }
}
```

*Tabel 16. MovieAppPreferences.kt Modul 5*

```
package com.example.movielist.data.local

import android.content.Context
import android.content.SharedPreferences

class MovieAppPreferences(context: Context) {

    private val sharedPreferences:
SharedPreferences =

context.getSharedPreferences("tmdb_app_prefs",
```

```
    Context.MODE_PRIVATE)
11
12      companion object {
13          private const val KEY_API_KEY = "api_key"
14          private const val KEY_DARK_MODE =
    "dark_mode"
15      }

16      fun saveApiKey(apiKey: String) {
17
18  sharedPreferences.edit().putString(KEY_API_KEY,
    apiKey).apply()
19      }
20
21      fun getApiKey(): String? {
22          return
    sharedPreferences.getString(KEY_API_KEY, null)
23      }

24      fun saveDarkModeState(isDarkMode: Boolean) {
25
    sharedPreferences.edit().putBoolean(KEY_DARK_MODE,
    isDarkMode).apply()
26      }
27
        fun getDarkModeState(): Boolean {
28          return
29  sharedPreferences.getBoolean(KEY_DARK_MODE, false)
30      }
31  }
```

*Tabel 17. RetrofitClient.kt Modul 5*

```
01  package com.example.movielist.data.remote.api
02
3
4   import
    com.jakewharton.retrofit2.converter.kotlinx.seriali
    zation.asConverterFactory
5   import kotlinx.serialization.json.Json
6   import okhttp3.MediaType.Companion.toMediaType
7   import okhttp3.OkHttpClient
8   import okhttp3.logging.HttpLoggingInterceptor
9   import retrofit2.Retrofit
```

```
import java.util.concurrent.TimeUnit

object RetrofitClient {

    private const val BASE_URL =
"https://api.themoviedb.org/3/"

    private val json = Json {
        ignoreUnknownKeys = true
        prettyPrint = true
    }

    private val okHttpClient: OkHttpClient by lazy
{
        val logging = HttpLoggingInterceptor()

logging.setLevel(HttpLoggingInterceptor.Level.BODY)

        OkHttpClient.Builder()
            .addInterceptor(logging)
            .connectTimeout(30, TimeUnit.SECONDS)
            .readTimeout(30, TimeUnit.SECONDS)
            .writeTimeout(30, TimeUnit.SECONDS)
            .build()
    }

    val tmdbApiService: TmdbApiService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .client(okHttpClient)

.addConverterFactory(json.asConverterFactory("appli
cation/json".toMediaType()))
            .build()
            .create(TmdbApiService::class.java)
    }
}
```

*Tabel 18. TmdbApiService.kt Modul 5*

```
package com.example.movielist.data.remote.api
-
2
3
```

```
    import
    com.example.movielist.data.remote.models.MovieListR
4   esponse
5   import retrofit2.Response
6   import retrofit2.http.GET
7   import retrofit2.http.Query
8
9   interface TmdbApiService {
10
11      @GET("movie/popular")
12      suspend fun getPopularMovies(
13          @Query("api_key") apiKey: String,
14          @Query("language") language: String = "en-
15  US",
16          @Query("page") page: Int = 1
        ): Response<MovieListResponse>
    }
```

*Tabel 19.MovieDto.kt Modul 5*

```
01  package com.example.movielist.data.remote.models
02
3   import kotlinx.serialization.SerialName
4   import kotlinx.serialization.Serializable
5
6   @Serializable
7   data class MovieDto(
8       val adult: Boolean,
9       @SerialName("backdrop_path")
10      val backdropPath: String?,
11      @SerialName("genre_ids")
12      val genreIds: List<Int>,
13      val id: Int,
14      @SerialName("original_language")
15      val originalLanguage: String,
16      @SerialName("original_title")
17      val originalTitle: String,
18      val overview: String,
19      val popularity: Double,
20      @SerialName("poster_path")
21      val posterPath: String?,
22      @SerialName("release_date")
23      val releaseDate: String,
24      val title: String,
25      val video: Boolean,
26      @SerialName("vote_average")
```

```
    val voteAverage: Double,
    @SerialName("vote_count")
    val voteCount: Int
)
```

*Tabel 20.MovieDtoExtension.kt*

```
package com.example.movielist.data.remote.models

import com.example.movielist.domain.model.Movie
import com.example.movielist.data.local.entities.MovieEntity

fun MovieDto.toDomainMovie(): Movie {
    return Movie(
        id = id,
        title = title,
        overview = overview,
        posterPath = posterPath,
        releaseDate = releaseDate,
        voteAverage = voteAverage
    )
}

fun MovieDto.toMovieEntity(): MovieEntity {
    return MovieEntity(
        id = id,
        title = title,
        overview = overview,
        posterPath = posterPath,
        releaseDate = releaseDate,
        voteAverage = voteAverage,
        popularity = popularity
    )
}
```

*Tabel 21. MovieListResponse.kt Modul 5*

```
package com.example.movielist.data.remote.models

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable
```

```
5
6   @Serializable
7   data class MovieListResponse(
8       val page: Int,
9       val results: List<MovieDto>,
10      @SerialName("total_pages")
11      val totalPages: Int,
12      @SerialName("total_results")
13      val totalResults: Int
14  )
```

*Tabel 22. MovieRepository.kt Modul 5*

```
01  package com.example.movielist.data.repository
-
2   import
    com.example.movielist.data.local.dao.MovieDao
3   import
    com.example.movielist.data.remote.api.TmdbApiServic
    e
4   import
    com.example.movielist.data.remote.models.toDomainMo
    vie
5   import
    com.example.movielist.data.remote.models.toMovieEnt
    ity
6   import com.example.movielist.domain.model.Movie
7   import com.example.movielist.utils.Result
8   import kotlinx.coroutines.flow.Flow
9   import kotlinx.coroutines.flow.flow
10  import retrofit2.HttpException
11  import java.io.IOException
12
13  interface MovieRepository {
14      fun getPopularMovies():
15  Flow<Result<List<Movie>>>
16  }
17
18  class MovieRepositoryImpl(
19      private val apiService: TmdbApiService,
20      private val movieDao: MovieDao,
21      private val apiKey: String
22  ) : MovieRepository {
23
```

```
24      override fun getPopularMovies():
        Flow<Result<List<Movie>>> = flow {
25          emit(Result.Loading)
26
27          val cachedMovies =
        movieDao.getAllMovies().map { it.toDomainMovie() }
28          if (cachedMovies.isNotEmpty()) {
29              emit(Result.Success(cachedMovies))
30          }
31
32          try {
33              val response =
        apiService.getPopularMovies(apiKey = apiKey)
34              if (response.isSuccessful) {
35                  val movieDtos =
        response.body()?.results ?: emptyList()
36                  val domainMovies = movieDtos.map {
        it.toDomainMovie() }
37
38                  movieDao.clearAllMovies()
39
        movieDao.insertAllMovies(movieDtos.map {
40      it.toMovieEntity() })
41
                    emit(Result.Success(domainMovies))
42              } else {
                    emit(Result.Error(Exception("API
43      Error: ${response.code()} ${response.message()}")))
44              }
45          } catch (e: HttpException) {
                emit(Result.Error(Exception("Network
46      Error (HTTP ${e.code()}): ${e.message()}")))
47          } catch (e: IOException) {
48              emit(Result.Error(Exception("No
        Internet Connection or API Timeout:
        ${e.message}")))
49          } catch (e: Exception) {
50              emit(Result.Error(Exception("An
        unexpected error occurred:
        ${e.localizedMessage}")))
51          }
52      }
53  }
```

*Tabel 23. Movie.kt Modul 5*

```
1   package com.example.movielist.domain.model
2
3   import android.os.Parcelable
4   import kotlinx.parcelize.Parcelize
5
6   @Parcelize
7   data class Movie(
8       val id: Int,
9       val title: String,
10      val overview: String,
11      val posterPath: String?,
12      val releaseDate: String,
13      val voteAverage: Double
14  ) : Parcelable
```

*Tabel 24. GetPopularMovieUseCase.kt Modul 5*

```
001 package com.example.movielist.domain.usecase
2
3
4   import com.example.movielist.domain.model.Movie
    import
    com.example.movielist.data.repository.MovieReposit
5   oryImpl
6   import com.example.movielist.utils.Result
7   import kotlinx.coroutines.flow.Flow
8
9   class GetPopularMoviesUseCase (
10      private val movieRepository:
11  MovieRepositoryImpl
12  ) {
13      operator fun invoke():
14  Flow<Result<List<Movie>>> {
            return movieRepository.getPopularMovies()
        }
    }
```

*Tabel 25. DetailActivity.kt Modul 5*

```
01  package
2   com.example.movielist.presentation.ui.activity
3
4   import android.os.Build
```

```
5  import android.os.Bundle
6  import android.view.MenuItem
7  import android.widget.Toast
8  import androidx.appcompat.app.AppCompatActivity
9  import com.bumptech.glide.Glide
   import
10 com.example.movielist.databinding.ActivityDetailBin
11 ding
12 import com.example.movielist.domain.model.Movie
13
14 class DetailActivity : AppCompatActivity() {
15
16     private lateinit var binding:
17 ActivityDetailBinding
18
19     companion object {
20         const val EXTRA_MOVIE = "extra_movie"
21     }
22
23     override fun onCreate(savedInstanceState:
   Bundle?) {
24         super.onCreate(savedInstanceState)
25         binding =
26 ActivityDetailBinding.inflate(layoutInflater)
27         setContentView(binding.root)
28
29
30
31 supportActionBar?.setDisplayHomeAsUpEnabled(true)

32
           val movie = if (Build.VERSION.SDK_INT >=
33 Build.VERSION_CODES.TIRAMISU) {
34             intent.getParcelableExtra(EXTRA_MOVIE,
35 Movie::class.java)
36         } else {
37             @Suppress("DEPRECATION")
38             intent.getParcelableExtra(EXTRA_MOVIE)
39         }
40
41         movie?.let {
42
43             supportActionBar?.title = it.title
44
               binding.apply {
```

```
                    tvDetailTitle.text = it.title
                    tvDetailReleaseDate.text = "Release
Date: ${it.releaseDate}"
                    tvDetailVoteAverage.text = "Rating:
${String.format("%.1f", it.voteAverage)}"
                    tvDetailOverview.text = it.overview

                    val imageUrl =
"https://image.tmdb.org/t/p/w500${it.posterPath}"
                    Glide.with(this@DetailActivity)
                        .load(imageUrl)
                        .centerCrop()
                        .into(ivDetailPoster)
                }
            } ?: run {
                Toast.makeText(this, "Film tidak
ditemukan.", Toast.LENGTH_SHORT).show()
                finish()
            }
        }


        override fun onOptionsItemSelected(item:
MenuItem): Boolean {
            if (item.itemId == android.R.id.home) {
                onBackPressedDispatcher.onBackPressed()
                return true
            }
            return super.onOptionsItemSelected(item)
        }
    }
```

*Tabel 26. MainActivity.kt Modul 5*

```
001 package
02  com.example.movielist.presentation.ui.activity
3
4
5   import android.content.Intent
6   import android.os.Bundle
7   import android.view.View
8   import android.widget.Toast
9   import androidx.activity.viewModels
10  import androidx.appcompat.app.AppCompatActivity
```

```
11  import androidx.appcompat.app.AppCompatDelegate
12  import androidx.lifecycle.Observer
13  import
    androidx.recyclerview.widget.LinearLayoutManager
14  import androidx.room.Room
    import
15  com.example.movielist.data.local.MovieAppPreferenc
    es
    import
16  com.example.movielist.data.local.database.AppDatab
    ase
17  import
    com.example.movielist.data.repository.MovieReposit
18  oryImpl
    import
    com.example.movielist.data.remote.api.RetrofitClie
19  nt
    import
    com.example.movielist.databinding.ActivityMainBind
20  ing
    import
    com.example.movielist.domain.usecase.GetPopularMov
21  iesUseCase
    import
    com.example.movielist.presentation.ui.adapter.Movi
22  eAdapter
23  import
24  com.example.movielist.presentation.viewmodel.Movie
25  ViewModel
26  import
27  com.example.movielist.presentation.viewmodel.ViewM
28  odelFactory
    import com.example.movielist.utils.Result
29  import com.example.movielist.R
30  class MainActivity : AppCompatActivity() {

31      private lateinit var binding:
32  ActivityMainBinding
33      private lateinit var movieAdapter:
34  MovieAdapter
35      private lateinit var movieAppPreferences:
36  MovieAppPreferences
37
38      private val movieViewModel: MovieViewModel by
    viewModels {
```

```
39          val apiService =
40  RetrofitClient.tmdbApiService
41          val database = Room.databaseBuilder(
                applicationContext,
42              AppDatabase::class.java,
                AppDatabase.DATABASE_NAME
43          ).build()
44          val movieDao = database.movieDao()
45          val tmdbApiKey =
46  "71819bfeac768c2a5b9a32b26e50cae1"
47          movieAppPreferences.saveApiKey(tmdbApiKey)
48
49          val movieRepositoryImpl =
        MovieRepositoryImpl(apiService, movieDao,
50  tmdbApiKey)
51          val getPopularMoviesUseCase =
52  GetPopularMoviesUseCase(movieRepositoryImpl)
53          ViewModelFactory(getPopularMoviesUseCase)
54      }
55
56      override fun onCreate(savedInstanceState:
57  Bundle?) {
58          super.onCreate(savedInstanceState)
59          binding =
60  ActivityMainBinding.inflate(layoutInflater)
61          setContentView(binding.root)
62
63
64  supportActionBar?.setDisplayHomeAsUpEnabled(false)
65          supportActionBar?.title = "Popular Movies"
66
67          movieAppPreferences =
68  MovieAppPreferences(this)
69
70          setupRecyclerView()
            observeViewModel()
71          setupDarkModeToggle()
72
73          binding.btnRetry.setOnClickListener {
74              movieViewModel.fetchPopularMovies()
75          }
        }
77
        private fun setupRecyclerView() {
78          movieAdapter = MovieAdapter()
```

```
79          binding.rvMovies.apply {
80              layoutManager =
81  LinearLayoutManager(this@MainActivity)
82              adapter = movieAdapter
83          }
84
            movieAdapter.onItemClick = { movie ->
85              val intent = Intent(this,
86  DetailActivity::class.java).apply {
87
    putExtra(DetailActivity.EXTRA_MOVIE, movie)
88              }
                startActivity(intent)
89          }
        }
90
        private fun observeViewModel() {
91          movieViewModel.popularMovies.observe(this,
92  Observer { result ->
93              when (result) {
                    is Result.Loading -> {
94                      binding.progressBar.visibility
    = View.VISIBLE
95                      binding.tvError.visibility =
    View.GONE
96                      binding.btnRetry.visibility =
    View.GONE
97                      binding.rvMovies.visibility =
98  View.GONE
99                  }
100                 is Result.Success -> {
101                     binding.progressBar.visibility
    = View.GONE
102                     binding.tvError.visibility =
    View.GONE
103                     binding.btnRetry.visibility =
    View.GONE
104                     binding.rvMovies.visibility =
    View.VISIBLE
105
    movieAdapter.submitList(result.data)
106                 }
                    is Result.Error -> {
107                     binding.progressBar.visibility
108 = View.GONE
```

```
109                         binding.rvMovies.visibility =
110 View.GONE
111                         binding.tvError.visibility =
112 View.VISIBLE
113                         binding.btnRetry.visibility =
    View.VISIBLE
114                         binding.tvError.text = "Error:
115 ${result.exception.message}"
116                         Toast.makeText(this, "Error:
117 ${result.exception.message}",
118 Toast.LENGTH_LONG).show()
                        }
119                 }
120             })
        }
121
122     private fun setupDarkModeToggle() {
123         val isDarkMode =
124 movieAppPreferences.getDarkModeState()
125         applyTheme(isDarkMode)
126         updateToggleIcon(isDarkMode)
127


128 binding.btnDarkModeToggle.setOnClickListener
129             val currentMode =
130 movieAppPreferences.getDarkModeState()
131             val newMode = !currentMode
132
133 movieAppPreferences.saveDarkModeState(newMode)
134             applyTheme(newMode)
135         }
136     }
137
        private fun applyTheme(isDarkMode: Boolean) {
            if (isDarkMode) {
138
139 AppCompatDelegate.setDefaultNightMode(AppCompatDel
    egate.MODE_NIGHT_YES)
140         } else {
141
142 AppCompatDelegate.setDefaultNightMode(AppCompatDel
    egate.MODE_NIGHT_NO)
        }
    }
```

```
    private fun updateToggleIcon(isDarkMode:
Boolean) {
        if (isDarkMode) {
binding.btnDarkModeToggle.setImageResource(R.drawa
ble.ic_light_bulb_off)
        } else {
binding.btnDarkModeToggle.setImageResource(R.drawa
ble.ic_light_bulb_on)
        }
    }
}
```

*Tabel 27. MovieAdapter.kt Modul 5*

```
01  package
-   com.example.movielist.presentation.ui.adapter
2
3
4   import android.view.LayoutInflater
5   import android.view.ViewGroup
6   import androidx.recyclerview.widget.DiffUtil
7   import androidx.recyclerview.widget.ListAdapter
8   import androidx.recyclerview.widget.RecyclerView
9   import com.bumptech.glide.Glide
    import
10  com.example.movielist.databinding.ItemMovieBinding
11  import com.example.movielist.domain.model.Movie
12
13  class MovieAdapter : ListAdapter<Movie,
14  MovieAdapter.MovieViewHolder>(MovieDiffCallback())
15  {
16
17      var onItemClick: ((Movie) -> Unit)? = null

18      override fun onCreateViewHolder(parent:
    ViewGroup, viewType: Int): MovieViewHolder {
            val binding =
19  ItemMovieBinding.inflate(LayoutInflater.from(parent
20  .context), parent, false)
21          return MovieViewHolder(binding)
22      }

23      override fun onBindViewHolder(holder:
24  MovieViewHolder, position: Int) {
```

```
25          val movie = getItem(position)
26          holder.bind(movie)
27      }

28      inner class MovieViewHolder(private val
29  binding: ItemMovieBinding) :
30          RecyclerView.ViewHolder(binding.root) {
31
32          init {
                binding.btnDetail.setOnClickListener {
33
34  onItemClick?.invoke(getItem(adapterPosition))
35              }
36          }
37
38          fun bind(movie: Movie) {
39              binding.apply {
                    tvMovieTitle.text = movie.title
40                  tvReleaseDate.text = "Release Date:
    ${movie.releaseDate}"
41                  tvVoteAverage.text = "Rating:
42  ${String.format("%.1f", movie.voteAverage)}"
43                  tvOverview.text = movie.overview

44                  val imageUrl =
45  "https://image.tmdb.org/t/p/w500${movie.posterPath}
46  "
47
48                  Glide.with(itemView.context)
49                      .load(imageUrl)
50                      .centerCrop()
51                      .into(ivPoster)
52              }
53          }
        }
54
55      class MovieDiffCallback :
    DiffUtil.ItemCallback<Movie>() {
56          override fun areItemsTheSame(oldItem:
57  Movie, newItem: Movie): Boolean {
58              return oldItem.id == newItem.id
            }
59
            override fun areContentsTheSame(oldItem:
60  Movie, newItem: Movie): Boolean {
```

```
61              return oldItem == newItem
            }
        }
    }
```

*Tabel 28. MovieViewModel.kt*

```
01  package
-   com.example.movielist.presentation.viewmodel
2
3
4   import androidx.lifecycle.LiveData
5   import androidx.lifecycle.MutableLiveData
6   import androidx.lifecycle.ViewModel
7   import androidx.lifecycle.viewModelScope
8   import com.example.movielist.domain.model.Movie
    import
    com.example.movielist.domain.usecase.GetPopularMovi
9   esUseCase
10  import com.example.movielist.utils.Result
    import kotlinx.coroutines.launch
11
12  class MovieViewModel (
        private val getPopularMoviesUseCase:
13  GetPopularMoviesUseCase
14  ) : ViewModel() {
15
        private val _popularMovies =
16  MutableLiveData<Result<List<Movie>>>()
        val popularMovies:
17  LiveData<Result<List<Movie>>> = _popularMovies
18
19      init {
20          fetchPopularMovies()
21      }
22
23      fun fetchPopularMovies() {
24          viewModelScope.launch {
25              getPopularMoviesUseCase().collect {
26  result ->
27                  _popularMovies.value = result
28              }
29          }
30
```

```
    }
}
```

*Tabel 29. ViewModelFactory.kt Modul 5*

```
01-  package com.example.movielist.presentation.viewmodel
2
3
4    import androidx.lifecycle.ViewModel
5    import androidx.lifecycle.ViewModelProvider
     import
     com.example.movielist.domain.usecase.GetPopularMoviesUs
6    eCase
7
8    class ViewModelFactory(
         private val getPopularMoviesUseCase:
9    GetPopularMoviesUseCase
10   ) : ViewModelProvider.Factory {
11
         override fun <T : ViewModel> create(modelClass:
12   Class<T>): T {
             if
13   (modelClass.isAssignableFrom(MovieViewModel::class.java
14   )) {
15               @Suppress("UNCHECKED_CAST")
                 return
16   MovieViewModel(getPopularMoviesUseCase) as T
17           }
             throw IllegalArgumentException("Unknown
18   ViewModel class")
19       }
     }
```

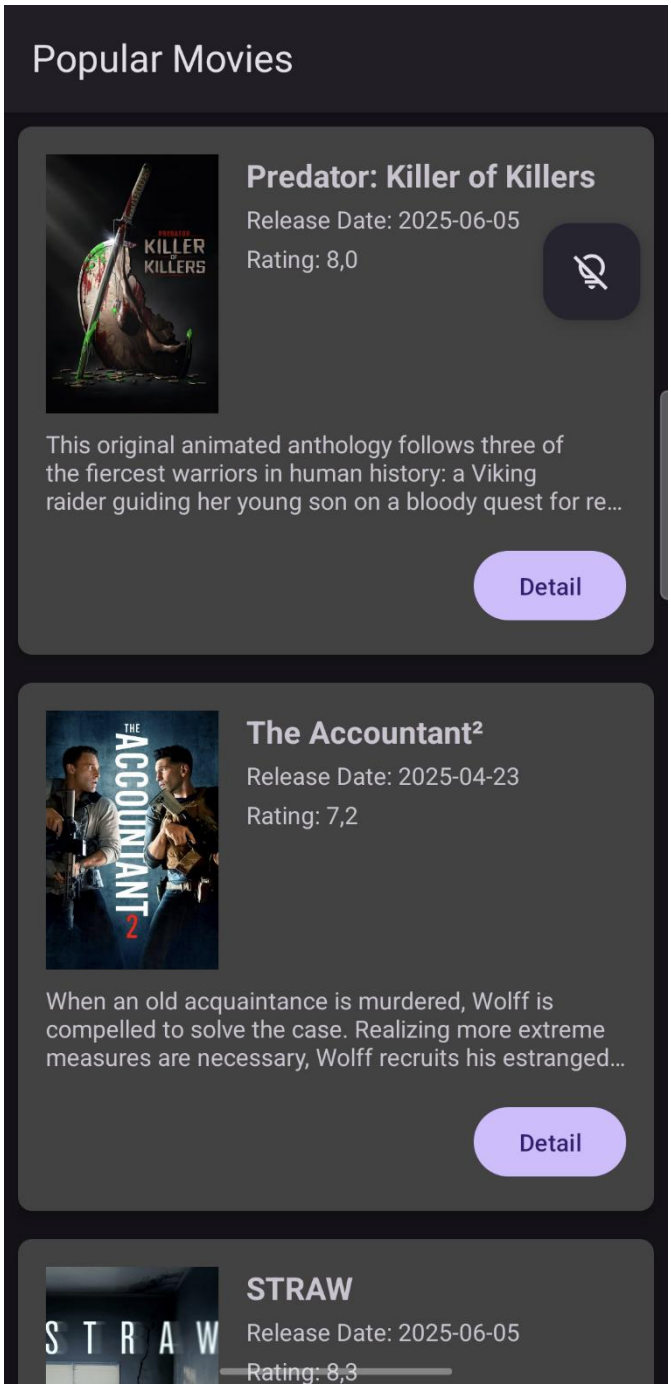
*Tabel 30. Result.kt Modul 5*

```
1    package com.example.movielist.utils
2
3    sealed class Result<out T> {
4        object Loading : Result<Nothing>()
5        data class Success<out T>(val data: T) :
     Result<T>()
6        data class Error(val exception: Exception) :
     Result<Nothing>()
7    }
```

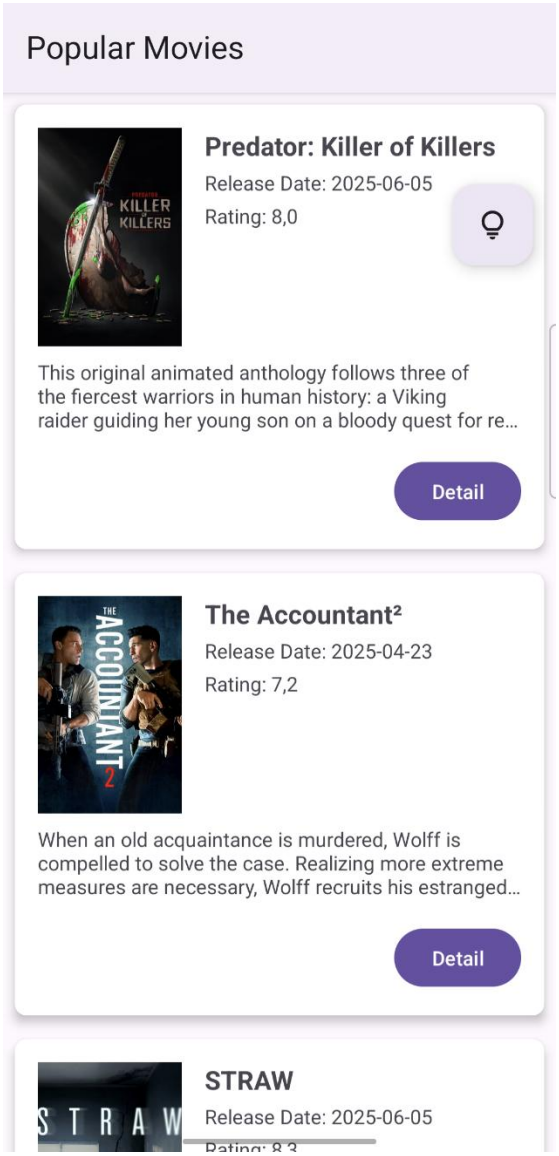
| | |
|---|---|

## B. Output Produk


*Gambar 15. Ouput Modul 5*




*Gambar 16. Ouput Modul 5*




*Gambar 17. Ouput Modul 5*

## C. Pembahasan

**MovieAppPreferences.kt:**

Pada baris 1, package com.example.movielist.data.local mendefinisikan *package* file ini di lapisan data lokal. Pada baris 3-4, import class Context dan SharedPreferences yang diperlukan untuk mengakses layanan sistem dan API penyimpanan data ringan.

Pada baris 6, class MovieAppPreferences(context: Context) mendefinisikan class MovieAppPreferences yang bertanggung jawab untuk mengelola akses ke SharedPreferences aplikasi. Konstruktornya menerima objek Context untuk inisialisasi. Pada baris 8-9, private val sharedPreferences: SharedPreferences = context.getSharedPreferences("tmdb_app_prefs", Context.MODE_PRIVATE) menginisialisasi instance SharedPreferences dengan nama file "tmdb_app_prefs" dan mode MODE_PRIVATE, berarti data hanya dapat diakses oleh aplikasi ini.

Pada baris 11, companion object { ... } mendefinisikan objek pendamping yang berisi konstanta kunci untuk data yang disimpan. Pada baris 12, private const val KEY_API_KEY = "api_key" mendefinisikan konstanta kunci untuk menyimpan API key. Pada baris 13, private const val KEY_DARK_MODE = "dark_mode" mendefinisikan konstanta kunci untuk menyimpan preferensi mode gelap.

Pada baris 15, fun saveApiKey(apiKey: String) adalah fungsi untuk menyimpan API key. Pada baris 16, sharedPreferences.edit().putString(KEY_API_KEY, apiKey).apply() mengambil editor SharedPreferences, menyimpan string API key, dan menerapkan perubahan secara asinkron. Pada baris 19, fun getApiKey(): String? adalah fungsi untuk mengambil API key yang tersimpan, mengembalikan null jika tidak ditemukan.

Pada baris 23, fun saveDarkModeState(isDarkMode: Boolean) adalah fungsi untuk menyimpan status mode gelap. Pada baris 24, sharedPreferences.edit().putBoolean(KEY_DARK_MODE, isDarkMode).apply() menyimpan status boolean mode gelap. Pada baris 27, fun getDarkModeState(): Boolean adalah fungsi untuk mengambil status mode gelap yang tersimpan, mengembalikan false secara default.

**MovieDao.kt**:

Pada baris 1, package com.example.movielist.data.local.dao mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk Data Access Object (DAO) film. Pada baris 3-7, import anotasi Room (@Dao, @Insert, @OnConflictStrategy, @Query) dan class MovieEntity yang akan berinteraksi dengan DAO ini.

Pada baris 9, @Dao menandai *interface* MovieDao sebagai Data Access Object untuk Room, yang akan secara otomatis diimplementasikan oleh Room. Pada baris 10, interface MovieDao mendeklarasikan antarmuka ini.

Pada baris 11, @Insert(onConflict = OnConflictStrategy.REPLACE) menandai fungsi ini sebagai operasi penyisipan. OnConflictStrategy.REPLACE akan mengganti data yang sudah ada jika ada konflik primary key. Pada baris 12, suspend fun insertAllMovies(movies: List<MovieEntity>) mendefinisikan fungsi suspend untuk menyisipkan daftar MovieEntity ke database. suspend menunjukkan bahwa ini adalah fungsi *coroutine* yang dapat dihentikan sementara.

Pada baris 14, @Query("SELECT * FROM movies ORDER BY popularity DESC") menandai fungsi ini dengan kueri SQL kustom untuk mengambil semua film dari tabel "movies" yang diurutkan berdasarkan popularitas secara descending. Pada baris 15, suspend fun getAllMovies(): List<MovieEntity> mendefinisikan fungsi suspend untuk mengambil semua MovieEntity dari database.

Pada baris 17, @Query("DELETE FROM movies") menandai fungsi ini dengan kueri SQL untuk menghapus semua entri dari tabel "movies". Pada baris 18, suspend fun clearAllMovies() mendefinisikan fungsi suspend untuk menghapus semua data film dari cache.

**AppDatabase.kt:**

Pada baris 1, package com.example.movielist.data.local.database mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk class database Room. Pada baris 3-7, import anotasi Room (@Database, @RoomDatabase, @TypeConverters), GenreConverter, MovieDao, dan MovieEntity.

Pada baris 9, @Database(entities = [MovieEntity::class], version = 2, exportSchema = false) menandai class AppDatabase sebagai database Room. entities = [MovieEntity::class] mendaftarkan MovieEntity sebagai tabel dalam database. version = 2 menetapkan versi database, yang perlu dinaikkan setiap kali skema database berubah (misalnya, penambahan kolom baru). exportSchema = false menonaktifkan ekspor skema database ke file, yang cocok untuk pengembangan.

Pada baris 10, @TypeConverters(GenreConverter::class) mendaftarkan GenreConverter sebagai *TypeConverter* untuk database ini, memungkinkan Room untuk menyimpan dan mengambil tipe data kompleks seperti List<Int> yang tidak didukung secara *native* oleh SQLite.

Pada baris 11, abstract class AppDatabase : RoomDatabase() mendefinisikan class abstrak AppDatabase yang mewarisi dari RoomDatabase. Room akan mengimplementasikan class ini secara otomatis. Pada baris 12, abstract fun movieDao(): MovieDao mendefinisikan fungsi abstrak untuk mendapatkan instance MovieDao, yang merupakan cara aplikasi berinteraksi dengan database.

Pada baris 14, companion object { ... } mendefinisikan objek pendamping. Pada baris 15, const val DATABASE_NAME = "tmdb_app_db" mendefinisikan konstanta untuk nama file database.

**MovieEntity.kt:**

Pada baris 1, package com.example.movielist.data.local.entities mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk entitas Room.

Pada baris 3-5, import anotasi Room (@Entity, @PrimaryKey) dan class Movie dari lapisan domain.

Pada baris 7, @Entity(tableName = "movies") menandai data class MovieEntity sebagai tabel dalam database Room dengan nama "movies". Pada baris 8, data class MovieEntity(...) mendefinisikan data class MovieEntity, yang merupakan representasi satu baris dalam tabel movies. Properti-propertinya (id, title, overview, dll.) menjadi kolom-kolom tabel.

Pada baris 9, @PrimaryKey val id: Int menandai id sebagai primary key, memastikan setiap film memiliki identifikasi unik di database. Pada baris 16, val genreIds: List<Int> adalah kolom baru yang menyimpan daftar ID genre; ini memerlukan TypeConverter karena List<Int> bukan tipe data yang didukung secara *native* oleh SQLite.

Pada baris 19, fun toDomainMovie(): Movie mendefinisikan fungsi di dalam MovieEntity yang mengonversi objek MovieEntity dari database menjadi Movie model domain. Ini adalah bagian penting dari pemetaan antar lapisan.

Pada baris 29, companion object { ... } mendefinisikan objek pendamping. Pada baris 30, fun fromDomainMovie(movie: Movie, popularity: Double, genreIds: List<Int>): MovieEntity adalah fungsi *factory* di companion object yang digunakan untuk membuat MovieEntity dari Movie domain model dan properti tambahan seperti popularity dan genreIds yang hanya ada di MovieEntity untuk tujuan penyimpanan.

**RetrofitClient.kt:**

Pada baris 1, package com.example.movielist.data.remote.api mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk klien API. Pada baris 3-9, import berbagai class dan fungsi yang diperlukan untuk konfigurasi Retrofit, KotlinX Serialization, OkHttp, dan penanganan waktu.

Pada baris 11, object RetrofitClient mendeklarasikan objek *singleton* RetrofitClient, yang berarti hanya ada satu instance dari objek ini di seluruh aplikasi. Ini adalah praktik umum untuk klien HTTP.

Pada baris 13, private const val BASE_URL = "https://api.themoviedb.org/3/" mendefinisikan konstanta URL dasar untuk semua permintaan ke TMDB API.

Pada baris 15-18, private val json = Json { ... } menginisialisasi instance Json dari KotlinX Serialization dengan konfigurasi kustom. ignoreUnknownKeys = true mengonfigurasi parser untuk mengabaikan kunci JSON yang tidak ada di model data Kotlin Anda, mencegah *crash* jika ada perubahan di API. prettyPrint = true digunakan untuk memformat output JSON agar mudah dibaca (berguna untuk *debugging*).

Pada baris 20-29, private val okHttpClient: OkHttpClient by lazy { ... } mendeklarasikan instance OkHttpClient secara *lazy* (akan dibuat saat pertama kali diakses). Di dalamnya, HttpLoggingInterceptor ditambahkan dengan level BODY untuk menampilkan detail permintaan dan respons HTTP di Logcat (berguna untuk *debugging*). Berbagai *timeout* (koneksi, baca, tulis) juga dikonfigurasi untuk mencegah permintaan menggantung terlalu lama.

Pada baris 32-38, val tmdbApiService: TmdbApiService by lazy { ... } mendeklarasikan instance TmdbApiService secara *lazy*. Retrofit.Builder() digunakan untuk membangun instance Retrofit dengan URL dasar, OkHttpClient yang sudah dikonfigurasi, dan KotlinX Serialization Converter Factory (json.asConverterFactory(...)) untuk mengonversi JSON menjadi objek Kotlin. Terakhir, .create(TmdbApiService::class.java) membuat implementasi TmdbApiService dari antarmuka yang didefinisikan.

**TmdbApiService.kt:**

Pada baris 1, package com.example.movielist.data.remote.api mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk antarmuka API. Pada baris 3-6, import class model respons MovieListResponse dan anotasi Retrofit (@GET, @Query, Response).

Pada baris 8, interface TmdbApiService mendeklarasikan antarmuka TmdbApiService. Ini mendefinisikan kontrak untuk berinteraksi dengan API TMDB.

Pada baris 10, @GET("movie/popular") menandai fungsi getPopularMovies() untuk melakukan permintaan HTTP GET ke endpoint "movie/popular" relatif terhadap URL dasar yang dikonfigurasi di RetrofitClient. Pada baris 11, suspend fun getPopularMovies(...) mendefinisikan fungsi suspend untuk mengambil daftar film populer. Kata kunci suspend berarti fungsi ini dapat dipanggil dari *coroutine*.

Pada baris 12, @Query("api_key") apiKey: String mendeklarasikan parameter kueri URL "api_key" yang wajib diisi. Pada baris 13, @Query("language") language: String = "en-US" menambahkan parameter kueri "language" dengan nilai default "en-US". Pada baris 14, @Query("page") page: Int = 1 menambahkan parameter kueri "page" untuk pagination dengan nilai default 1.

Pada baris 15, : Response<MovieListResponse> menentukan bahwa fungsi ini akan mengembalikan objek Response dari Retrofit yang membungkus MovieListResponse, yang berisi daftar film.

**MovieDto.kt:**

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk model data. Pada baris 3-4, import anotasi SerialName dan Serializable dari KotlinX Serialization.

Pada baris 6, @Serializable menandai data class MovieDto agar dapat diubah menjadi/dari format JSON oleh KotlinX Serialization. Pada baris 7, data class MovieDto(...) mendefinisikan data class MovieDto, yang berfungsi sebagai Data Transfer Object (DTO) untuk film. Struktur propertinya secara langsung memetakan struktur JSON yang diterima dari TMDB API.

Pada baris 8-28, setiap properti seperti adult, backdropPath, genreIds, id, originalLanguage, originalTitle, overview, popularity, posterPath, releaseDate, title, video, voteAverage, dan voteCount didefinisikan. Anotasi @SerialName("nama_json") digunakan untuk properti di mana nama Kotlin berbeda dari nama kunci di JSON (misalnya, backdropPath untuk backdrop_path). Tanda ? setelah tipe data (misalnya String?) menunjukkan bahwa properti tersebut bisa bernilai null.

**MovieListResponse.kt:**

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk model data. Pada baris 3-4, import anotasi SerialName dan Serializable dari KotlinX Serialization.

Pada baris 6, @Serializable menandai data class MovieListResponse agar dapat diubah menjadi/dari format JSON. Pada baris 7, data class MovieListResponse(...) mendefinisikan data class MovieListResponse, yang merepresentasikan struktur respons keseluruhan ketika meminta daftar film populer dari TMDB API.

Pada baris 8, val page: Int mendefinisikan properti untuk nomor halaman saat ini. Pada baris 9, val results: List<MovieDto> mendefinisikan properti results, yang merupakan daftar aktual dari objek MovieDto (daftar film).

Pada baris 10, @SerialName("total_pages") val totalPages: Int memetakan kunci JSON "total_pages" ke properti totalPages (jumlah total halaman hasil). Pada baris 12, @SerialName("total_results") val totalResults: Int memetakan kunci JSON "total_results" ke properti totalResults (jumlah total film yang ditemukan).

**MovieDtoExtension.kt:**

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini untuk file ekstensi model data. Pada baris 3-4, import class Movie dari domain dan MovieEntity dari lapisan data lokal.

Pada baris 6, fun MovieDto.toDomainMovie(): Movie mendefinisikan fungsi ekstensi untuk MovieDto. Fungsi ini mengonversi sebuah objek MovieDto (yang berasal dari API) menjadi Movie model domain yang bersih. Ini adalah bagian penting dari pemetaan antar lapisan data dan domain.

Pada baris 15, fun MovieDto.toMovieEntity(): MovieEntity mendefinisikan fungsi ekstensi lain untuk MovieDto. Fungsi ini mengonversi objek MovieDto dari API menjadi MovieEntity yang dapat disimpan di Room Database. Fungsi ini menggunakan MovieEntity.fromDomainMovie() untuk melakukan konversi, meneruskan properti yang relevan termasuk popularity dan genreIds yang spesifik untuk MovieEntity.

**MovieRepository.kt:**

Pada baris 1, package com.example.movielist.data.repository mendefinisikan *package* untuk repository, bagian dari lapisan data. Pada baris 3-10, import berbagai class dan interface yang dibutuhkan untuk fungsionalitas repository (DAO, API service, model, Flow, Result).

Pada baris 12, interface MovieRepository mendeklarasikan antarmuka MovieRepository. Ini mendefinisikan kontrak tentang bagaimana data film akan disediakan, tanpa mengungkapkan detail implementasinya. Pada baris 13, fun getPopularMovies(): Flow<Result<List<Movie>>> adalah satu-satunya fungsi dalam antarmuka, yang akan mengembalikan Flow yang membungkus Result dari daftar Movie.

Pada baris 16, class MovieRepositoryImpl(...) : MovieRepository mendefinisikan class MovieRepositoryImpl, yang merupakan implementasi konkret dari antarmuka MovieRepository. Konstruktornya menerima dependensi TmdbApiService (untuk jaringan) dan MovieDao (untuk database lokal).

Pada baris 20, override fun getPopularMovies(): Flow<Result<List<Movie>>> = flow { ... } mengimplementasikan fungsi dari antarmuka. Ini adalah inti dari strategi *caching* dan pengambilan data. Pada baris 21, emit(Result.Loading) segera memancarkan status Loading ke Flow, memberi tahu UI bahwa proses pengambilan data telah dimulai.

Pada baris 23, val cachedMovies = movieDao.getAllMovies().map { it.toDomainMovie() } mencoba mengambil data film yang sudah ada di cache Room Database. Data ini kemudian dipetakan ke domain model Movie. Pada baris 24-26, jika ada data di cache, data tersebut segera dipancarkan sebagai Result.Success, memastikan aplikasi dapat menampilkan data dengan cepat.

Pada baris 28-47, blok try { ... } catch (...) { ... } menangani pengambilan data dari jaringan dan berbagai jenis error. Pada baris 29, val response = apiService.getPopularMovies(apiKey = apiKey) melakukan panggilan API ke TMDB. Pada baris 30-32, jika panggilan API berhasil, DTO film diambil dan dipetakan ke domain model.

Pada baris 34, movieDao.clearAllMovies() menghapus data lama dari cache Room. Pada baris 35, movieDao.insertAllMovies(movieDtos.map { it.toMovieEntity() }) menyisipkan data film terbaru dari API ke dalam cache Room. Pada baris 37, emit(Result.Success(domainMovies)) memancarkan data terbaru ke Flow untuk diperbarui di UI. Baris 39-47 adalah blok catch yang menangani HttpException (kesalahan API), IOException (masalah koneksi/timeout), dan Exception umum, memancarkan Result.Error yang sesuai.

**Movie.kt:**

Pada baris 1, package com.example.movielist.domain.model mendefinisikan *package* ini sebagai bagian dari lapisan domain untuk model data. Pada baris 3-4, import antarmuka Parcelable dari Android dan anotasi @Parcelize dari plugin Kotlin Parcelize.

Pada baris 6, @Parcelize adalah anotasi yang secara otomatis menghasilkan implementasi kode *boilerplate* Parcelable untuk data class Movie, sehingga memungkinkan objek ini untuk dikirim antar komponen Android (seperti antar Activity) secara efisien tanpa harus menulis kode manual.

Pada baris 7, data class Movie(...) : Parcelable mendefinisikan data class Movie. Ini adalah model domain yang bersih, yang berarti ia tidak bergantung pada detail implementasi API (DTO) atau database (Entity). Ia hanya berisi data yang relevan untuk logika bisnis dan presentasi. : Parcelable menunjukkan bahwa class ini mengimplementasikan antarmuka Parcelable.

Pada baris 8-13, properti-properti seperti id, title, overview, posterPath, releaseDate, dan voteAverage adalah atribut-atribut film yang relevan di lapisan domain aplikasi.

**GetPopularMoviesUseCase.kt:**

Pada baris 1, package com.example.movielist.domain.usecase mendefinisikan *package* ini sebagai bagian dari lapisan domain untuk *use case*. Pada baris 3-6, import class yang dibutuhkan (Movie model domain, MovieRepositoryImpl implementasi repository, Result, Flow).

Pada baris 8, class GetPopularMoviesUseCase(...) mendefinisikan *use case* GetPopularMoviesUseCase. *Use case* ini mengkapsulasi logika bisnis spesifik untuk "mendapatkan daftar film populer".

Pada baris 9, private val movieRepository: MovieRepositoryImpl mendeklarasikan dependensi pada implementasi repository (MovieRepositoryImpl). Dalam arsitektur Clean Architecture yang lebih ketat, *use case* seharusnya bergantung pada antarmuka repository (yang berada di lapisan domain), tetapi di sini disesuaikan dengan keputusan untuk menggabungkan antarmuka dan implementasi repository.

Pada baris 11, operator fun invoke(): Flow<Result<List<Movie>>> adalah fungsi operator invoke. Ini memungkinkan instance dari GetPopularMoviesUseCase

dipanggil sebagai fungsi (misalnya getPopularMoviesUseCase()) alih-alih getPopularMoviesUseCase.invoke(). Fungsi ini mengembalikan Flow yang membungkus Result dari daftar Movie. Pada baris 12, return movieRepository.getPopularMovies() memanggil fungsi getPopularMovies() dari repository untuk mendapatkan data, dan mengembalikan Flow hasilnya. *Use case* ini sendiri tidak memiliki logika kompleks lain selain mendelegasikan tugas ke repository.

**MainActivity.kt:**

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk Activity utama. Pada baris 3-21, import berbagai class dan interface yang dibutuhkan untuk fungsionalitas Activity.

Pada baris 23, class MainActivity : AppCompatActivity() mendefinisikan class MainActivity sebagai titik masuk utama aplikasi. Pada baris 26-28, private lateinit var binding: ActivityMainBinding, private lateinit var movieAdapter: MovieAdapter, dan private lateinit var movieAppPreferences: MovieAppPreferences mendeklarasikan variabel untuk View Binding, adapter RecyclerView, dan preferensi aplikasi.

Pada baris 30-42, private val movieViewModel: MovieViewModel by viewModels { ... } mendeklarasikan dan menginisialisasi MovieViewModel. Di blok ini, semua dependensi ViewModel (API service, database Room, DAO, API key, repository, use case) diinjeksi secara manual ke ViewModelFactory.

Pada baris 44, override fun onCreate(savedInstanceState: Bundle?) adalah metode *lifecycle* yang dipanggil saat Activity pertama kali dibuat. Pada baris 46-47, binding = ActivityMainBinding.inflate(layoutInflater) dan setContentView(binding.root) menginisialisasi View Binding dan mengatur layout Activity. Pada baris 49, supportActionBar?.setDisplayHomeAsUpEnabled(false) menonaktifkan tombol kembali di ActionBar (karena ini adalah layar utama). Pada baris 50, supportActionBar?.title = "Popular Movies" mengatur judul ActionBar. Pada

baris 52, movieAppPreferences = MovieAppPreferences(this) menginisialisasi MovieAppPreferences.

Pada baris 54-56, setupRecyclerView(), observeViewModel(), dan setupDarkModeToggle() dipanggil untuk menyiapkan UI, mengamati data, dan mengelola mode gelap. Pada baris 58-60, binding.btnRetry.setOnClickListener { ... } mengatur *listener* klik untuk tombol "Retry" yang akan memanggil movieViewModel.fetchPopularMovies() untuk memuat ulang data.

Pada baris 63, private fun setupRecyclerView() menyiapkan RecyclerView dengan LinearLayoutManager dan MovieAdapter. Pada baris 70-74, movieAdapter.onItemClick = { movie -> ... } mengatur *callback* untuk item adapter, yang akan meluncurkan DetailActivity dan meneruskan objek Movie yang dipilih.

Pada baris 77, private fun observeViewModel() mengamati popularMovies LiveData dari movieViewModel. Pada baris 78-95, blok when (result) { ... } memperbarui UI berdasarkan Result state (Loading, Success, Error): menampilkan ProgressBar saat loading, menampilkan RecyclerView dengan data saat sukses, dan menampilkan pesan error serta tombol retry saat terjadi error.

Pada baris 97, private fun setupDarkModeToggle() mengelola fungsionalitas mode gelap. Pada baris 99, val isDarkMode = movieAppPreferences.getDarkModeState() membaca status mode gelap dari preferensi. Pada baris 100, applyTheme(isDarkMode) menerapkan tema yang sesuai, dan updateToggleIcon(isDarkMode) memperbarui ikon tombol. Pada baris 103-107, binding.btnDarkModeToggle.setOnClickListener { ... } menangani klik pada tombol mode gelap, membalik status mode, menyimpan ke preferensi, dan menerapkan tema baru.

Pada baris 110, private fun applyTheme(isDarkMode: Boolean) adalah fungsi untuk menerapkan tema. Pada baris 111-115, AppCompatDelegate.setDefaultNightMode() digunakan untuk mengalihkan tema

aplikasi antara mode siang dan malam. Pemanggilan ini akan menyebabkan Activity dibuat ulang.

Pada baris 117, private fun updateToggleIcon(isDarkMode: Boolean) adalah fungsi untuk memperbarui ikon pada tombol mode gelap (ImageButton) berdasarkan status isDarkMode (lampu mati untuk gelap, lampu nyala untuk terang).

**DetailActivity.kt:**

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk Activity detail. Pada baris 3-9, import berbagai class yang dibutuhkan untuk fungsionalitas Activity.

Pada baris 11, class DetailActivity : AppCompatActivity() mendefinisikan class DetailActivity, yang bertanggung jawab untuk menampilkan detail film. Pada baris 14, private lateinit var binding: ActivityDetailBinding mendeklarasikan variabel binding untuk View Binding.

Pada baris 16, companion object { ... } mendefinisikan objek pendamping. Pada baris 17, const val EXTRA_MOVIE = "extra_movie" mendefinisikan konstanta kunci yang digunakan untuk meneruskan objek Movie melalui Intent.

Pada baris 20, override fun onCreate(savedInstanceState: Bundle?) adalah metode *lifecycle* yang dipanggil saat Activity pertama kali dibuat. Pada baris 22-23, binding = ActivityDetailBinding.inflate(layoutInflater) dan setContentView(binding.root) menginisialisasi View Binding dan mengatur layout Activity. Pada baris 25, supportActionBar?.setDisplayHomeAsUpEnabled(true) mengaktifkan tombol kembali (panah ke kiri) di ActionBar Activity ini.

Pada baris 27-31, val movie = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) { ... } else { ... } mengambil objek Movie yang diteruskan dari MainActivity melalui Intent.getParcelableExtra(), dengan penanganan untuk API level yang berbeda.

Pada baris 33-49, blok movie?.let { ... } akan dieksekusi hanya jika objek movie berhasil diterima dan tidak null. Di dalamnya, pada baris 34, supportActionBar?.title = it.title mengatur judul ActionBar dengan judul film. Pada baris 36-40, data film (title, releaseDate, voteAverage, overview) ditampilkan ke TextView yang sesuai menggunakan binding. Pada baris 42, val imageUrl = "https://image.tmdb.org/t/p/w500${it.posterPath}" membangun URL lengkap untuk gambar poster film. Pada baris 43-46, Glide.with(this@DetailActivity).load(imageUrl).centerCrop().into(ivDetailPoster) menggunakan Glide untuk memuat gambar poster. Jika movie null, pada baris 49-51, Toast akan ditampilkan dan Activity akan ditutup.

Pada baris 54, override fun onOptionsItemSelected(item: MenuItem): Boolean adalah metode *callback* yang dipanggil saat item di ActionBar diklik. Pada baris 55-58, if (item.itemId == android.R.id.home) memeriksa apakah item yang diklik adalah tombol kembali (android.R.id.home), dan jika ya, onBackPressedDispatcher.onBackPressed() dipanggil untuk mensimulasikan penekanan tombol kembali dan mengakhiri Activity.

**MovieAdapter.kt:**

Pada baris 1, package com.example.movielist.presentation.ui.adapter mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk adapter RecyclerView. Pada baris 3-8, import berbagai class yang dibutuhkan untuk fungsionalitas adapter (LayoutInflater, ViewGroup, DiffUtil, ListAdapter, RecyclerView, Glide, View Binding, Movie model domain).

Pada baris 10, class MovieAdapter : ListAdapter<Movie, MovieAdapter.MovieViewHolder>(MovieDiffCallback()) mendefinisikan MovieAdapter. Ini adalah turunan dari ListAdapter, sebuah jenis adapter RecyclerView yang sangat efisien dalam memperbarui daftar item karena menggunakan DiffUtil untuk menghitung perbedaan antar daftar. Ia dikonfigurasi untuk menampilkan objek

Movie dan menggunakan MovieAdapter.MovieViewHolder. MovieDiffCallback() adalah *callback* yang digunakan oleh DiffUtil.

Pada baris 12, var onItemClick: ((Movie) -> Unit)? = null mendeklarasikan properti *lambda* nullable bernama onItemClick. Properti ini berfungsi sebagai *callback* yang dapat diatur dari MainActivity untuk merespons klik pada tombol "Detail" di setiap item daftar, meneruskan objek Movie yang diklik.

Pada baris 14, override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder adalah metode *callback* yang dipanggil ketika RecyclerView membutuhkan ViewHolder baru. Di dalamnya, ItemMovieBinding.inflate() digunakan untuk meng-*inflate* layout item_movie.xml dan membuat MovieViewHolder baru.

Pada baris 19, override fun onBindViewHolder(holder: MovieViewHolder, position: Int) adalah metode yang dipanggil untuk menampilkan data pada posisi tertentu. Ia mengambil objek Movie dari daftar menggunakan getItem(position) dan memanggil holder.bind(movie) untuk mengisi tampilan.

Pada baris 24, inner class MovieViewHolder(private val binding: ItemMovieBinding) : RecyclerView.ViewHolder(binding.root) mendefinisikan *inner class* MovieViewHolder yang berfungsi sebagai *container* untuk tampilan setiap item daftar. binding menyediakan akses mudah ke elemen UI dari item_movie.xml.

Pada baris 27, init { binding.btnDetail.setOnClickListener { ... } } adalah blok inisialisasi untuk MovieViewHolder. Di sinilah OnClickListener untuk btnDetail diatur. Ketika tombol diklik, onItemClick?.invoke(getItem(adapterPosition)) dipanggil, yang memicu *callback* di MainActivity dengan objek Movie yang sesuai.

Pada baris 32, fun bind(movie: Movie) adalah fungsi di dalam MovieViewHolder yang bertanggung jawab untuk mengisi elemen-elemen UI dengan data dari objek Movie. Pada baris 33-37, binding.apply { ... } digunakan untuk mengatur teks tvMovieTitle, tvReleaseDate, tvVoteAverage, dan tvOverview dari properti objek movie. Pada baris 39, val imageUrl =

"https://image.tmdb.org/t/p/w500${movie.posterPath}" membangun URL lengkap untuk gambar poster. Pada baris 40-43, Glide.with(itemView.context).load(imageUrl).centerCrop().into(ivPoster) menggunakan Glide untuk memuat gambar poster ke ImageView.

Pada baris 46, class MovieDiffCallback : DiffUtil.ItemCallback<Movie>() mendefinisikan *custom* DiffUtil.ItemCallback. Pada baris 47, override fun areItemsTheSame(oldItem: Movie, newItem: Movie): Boolean membandingkan dua item untuk melihat apakah mereka adalah objek yang sama (berdasarkan id). Pada baris 50, override fun areContentsTheSame(oldItem: Movie, newItem: Movie): Boolean membandingkan konten dari dua item yang sama persis untuk mendeteksi perubahan data.

**MovieViewModel.kt:**

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk ViewModel. Pada baris 3-9, import berbagai class dan interface yang dibutuhkan (LiveData, ViewModel, viewModelScope, Movie model domain, GetPopularMoviesUseCase, Result, *coroutine* launch).

Pada baris 11, class MovieViewModel(...) : ViewModel() mendefinisikan MovieViewModel, yang merupakan turunan dari androidx.lifecycle.ViewModel. ViewModel bertanggung jawab untuk menyiapkan dan mengelola data yang terkait dengan UI, memastikan data tetap ada saat konfigurasi perangkat berubah (misalnya, rotasi layar) dan melepaskan sumber daya saat tidak lagi dibutuhkan.

Pada baris 12, private val getPopularMoviesUseCase: GetPopularMoviesUseCase mendeklarasikan dependensi pada GetPopularMoviesUseCase. MovieViewModel tidak berinteraksi langsung dengan repository atau API, tetapi mendelegasikan semua logika bisnis ke *use case*.

Pada baris 15, private val _popularMovies = MutableLiveData<Result<List<Movie>>>() mendeklarasikan MutableLiveData

private. Ini adalah LiveData yang dapat diubah nilainya dan akan menampung hasil pengambilan data film (dibungkus dalam Result yang berisi daftar Movie). Pada baris 16, val popularMovies: LiveData<Result<List<Movie>>> = _popularMovies mengekspos versi LiveData yang tidak dapat diubah (immutable) ke UI, yang akan mengamatinya untuk mendapatkan pembaruan data.

Pada baris 18, init { fetchPopularMovies() } adalah blok inisialisasi yang akan dipanggil saat instance MovieViewModel pertama kali dibuat. Ini secara otomatis memicu proses pengambilan data film.

Pada baris 21, fun fetchPopularMovies() mendefinisikan fungsi untuk memicu pengambilan data film. Pada baris 22, viewModelScope.launch { ... } meluncurkan *coroutine* dalam cakupan viewModelScope. viewModelScope memastikan bahwa *coroutine* ini akan secara otomatis dibatalkan ketika ViewModel dihancurkan, mencegah kebocoran memori. Pada baris 23, getPopularMoviesUseCase().collect { result -> ... } memanggil invoke() operator dari *use case* dan mengumpulkan nilai-nilai yang dipancarkan oleh Flow yang dikembalikan oleh *use case*. Setiap kali *use case* memancarkan Result baru (Loading, Success, atau Error), blok collect akan menerimanya. Pada baris 24, _popularMovies.value = result memperbarui nilai MutableLiveData, yang secara otomatis akan memberitahu Observer di UI (MainActivity) untuk memperbarui tampilannya.

**ViewModelFactory.kt:**

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk ViewModel Factory. Pada baris 3-5, import class ViewModel, ViewModelProvider, dan GetPopularMoviesUseCase.

Pada baris 7, class ViewModelFactory(...) : ViewModelProvider.Factory mendefinisikan ViewModelFactory kustom yang mengimplementasikan ViewModelProvider.Factory. Factory ini bertanggung jawab untuk membuat instance ViewModel dengan dependensi yang diperlukan. Ini adalah cara manual untuk

melakukan *Dependency Injection* untuk ViewModel, karena ViewModel tidak dapat memiliki konstruktor dengan parameter secara langsung oleh sistem Android.

Pada baris 8, private val getPopularMoviesUseCase: GetPopularMoviesUseCase adalah dependensi yang dibutuhkan oleh MovieViewModel. Factory ini menerimanya melalui konstruktor.

Pada baris 11, override fun <T : ViewModel> create(modelClass: Class<T>): T adalah metode yang harus diimplementasikan dari ViewModelProvider.Factory. Metode ini bertanggung jawab untuk membuat instance ViewModel yang diminta.

Pada baris 12, if (modelClass.isAssignableFrom(MovieViewModel::class.java)) memeriksa apakah modelClass yang diminta adalah MovieViewModel. Jika ya, pada baris 14, return MovieViewModel(getPopularMoviesUseCase) as T membuat instance baru MovieViewModel dengan dependensi getPopularMoviesUseCase yang disuntikkan. as T adalah *unsafe cast* yang di-*suppress*. Pada baris 17, throw IllegalArgumentException("Unknown ViewModel class") melempar pengecualian jika modelClass yang diminta tidak dikenali oleh factory ini.

**Result.kt:**

Pada baris 1, package com.example.movielist.utils mendefinisikan *package* ini untuk utilitas umum. Pada baris 3, sealed class Result<out T> mendefinisikan sealed class bernama Result. Sealed class adalah class abstrak yang nilai-nilainya terbatas pada satu set subclass yang didefinisikan dalam class itu sendiri. Ini sangat berguna untuk merepresentasikan *state* yang berbeda dari sebuah operasi (seperti Loading, Success, Error) dengan cara yang aman dan *type-safe* (out T menunjukkan kovarian tipe).

Pada baris 4, object Loading : Result<Nothing>() adalah objek *singleton* yang merepresentasikan status data sedang dimuat. Nothing menunjukkan bahwa tidak ada data yang terkait dengan *state* ini.

Pada baris 5, data class Success<out T>(val data: T) : Result<T>() adalah *data class* yang merepresentasikan status data berhasil dimuat. Ia membungkus data aktual (val data: T).

Pada baris 6, data class Error(val exception: Exception) : Result<Nothing>() adalah *data class* yang merepresentasikan status terjadi kesalahan. Ia membungkus objek Exception yang menjelaskan kesalahan tersebut.

**activity_main.xml:**

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah tag root layout, menggunakan ConstraintLayout yang fleksibel untuk memposisikan dan mengukur tampilan. Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan *namespace* untuk atribut layout. Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat layout mengisi seluruh lebar dan tinggi layar. Pada baris 8, tools:context=".presentation.ui.activity.MainActivity" adalah atribut *tooling* untuk Android Studio.

Pada baris 11-18, <TextView android:id="@+id/tv_title_placeholder" ... /> adalah TextView *placeholder* yang disembunyikan (visibility="gone") karena judul "Popular Movies" diatur oleh ActionBar di MainActivity.kt.

Pada baris 21-26, <ProgressBar android:id="@+id/progress_bar" ... /> adalah ProgressBar yang awalnya disembunyikan (visibility="gone") dan diposisikan di tengah layar, berfungsi sebagai indikator loading data.

Pada baris 29-38, <TextView android:id="@+id/tv_error" ... /> adalah TextView untuk menampilkan pesan error. Awalnya disembunyikan, akan muncul saat terjadi kesalahan. app:layout_constraintVertical_chainStyle="packed" dan *constraint* terkait digunakan untuk memusatkan TextView ini bersama dengan tombol "Retry" secara vertikal.

Pada baris 41-48, <Button android:id="@+id/btn_retry" ... /> adalah tombol "Retry". Awalnya disembunyikan, akan muncul di bawah pesan error saat terjadi kesalahan, memungkinkan pengguna untuk mencoba memuat ulang data.

Pada baris 51-58, <androidx.recyclerview.widget.RecyclerView android:id="@+id/rv_movies" ... /> adalah RecyclerView yang digunakan untuk menampilkan daftar film. Ia dikonfigurasi untuk mengisi seluruh ruang match_parent dan dimulai dari bagian atas (constraintTopToTopOf="parent"), yang berarti ia akan berada di bawah ActionBar dan FloatingActionButton dark mode akan menimpanya. tools:listitem digunakan untuk pratinjau layout di Android Studio.

Pada baris 61-73, <com.google.android.material.floatingactionbutton.FloatingActionButton android:id="@+id/btn_dark_mode_toggle" ... /> adalah FloatingActionButton untuk mengalihkan mode gelap. Ia diposisikan di pojok kanan atas (layout_marginEnd, layout_marginTop="?attr/actionBarSize") agar terlihat di bawah ActionBar dan sedikit menimpa RecyclerView. Atribut clickable, focusable, contentDescription diatur. app:srcCompat menentukan ikon awal. app:fabSize="mini" membuatnya berukuran kecil. app:tint dan app:backgroundTint digunakan untuk mewarnai ikon dan *background* tombol agar beradaptasi secara otomatis dengan tema (?attr/colorOnSurface dan ?attr/colorSurface)

**item_movie.xml:**

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <androidx.cardview.widget.CardView ...> adalah tag root layout untuk item daftar film. CardView digunakan untuk memberikan tampilan item dengan sudut membulat (cardCornerRadius) dan elevasi (cardElevation), yang umum di Material Design.

Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="wrap_content" membuat CardView mengisi lebar penuh dan

tingginya sesuai konten. Pada baris 8, android:layout_margin="8dp" menambahkan margin di semua sisi CardView untuk jarak antar item.

Pada baris 12, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam CardView, digunakan untuk mengatur posisi elemen-elemen detail film di dalam kartu. Pada baris 13, android:padding="16dp" menambahkan padding di dalam ConstraintLayout.

Pada baris 15-20, <ImageView android:id="@+id/iv_poster" ... /> mendeklarasikan ImageView untuk menampilkan poster film. Ini dikonfigurasi dengan lebar dan tinggi tetap, skala centerCrop, dan diposisikan di pojok kiri atas.

Pada baris 22-29, <TextView android:id="@+id/tv_movie_title" ... /> adalah TextView untuk menampilkan judul film, diposisikan di sebelah kanan poster dengan gaya teks tebal dan ukuran yang lebih besar.

Pada baris 31-38, <TextView android:id="@+id/tv_release_date" ... /> adalah TextView untuk menampilkan tanggal rilis, diposisikan di bawah judul film.

Pada baris 40-47, <TextView android:id="@+id/tv_vote_average" ... /> adalah TextView untuk menampilkan rata-rata voting/rating film, diposisikan di bawah tanggal rilis.

Pada baris 49-56, <TextView android:id="@+id/tv_overview" ... /> adalah TextView untuk menampilkan ringkasan (overview) film. Ia dikonfigurasi dengan maxLines dan ellipsize="end" untuk memotong teks jika terlalu panjang dan menampilkan elipsis (...). Ini diposisikan di bawah poster utama.

Pada baris 58-63, <Button android:id="@+id/btn_detail" ... /> adalah tombol "Detail". Ini diposisikan di pojok kanan bawah kartu item dan akan meluncurkan DetailActivity saat diklik.

**activity_detail.xml:**

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <ScrollView ...> adalah tag root layout. ScrollView memungkinkan konten di dalamnya untuk digulir jika ukurannya melebihi tinggi layar, yang penting untuk halaman detail film yang mungkin memiliki sinopsis panjang. Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan *namespace*. Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat ScrollView mengisi seluruh layar. Pada baris 8, tools:context=".presentation.ui.activity.DetailActivity" adalah atribut *tooling* untuk Android Studio.

Pada baris 10, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam ScrollView, digunakan untuk mengatur posisi elemen-elemen detail. Pada baris 11, android:padding="16dp" menambahkan *padding* di dalam ConstraintLayout.

Pada baris 13-18, <ImageView android:id="@+id/iv_detail_poster" ... /> mendeklarasikan ImageView untuk menampilkan poster film detail. Ini dikonfigurasi dengan lebar mengisi parent (0dp), tinggi tetap (300dp), skala centerCrop, dan diposisikan di bagian atas layout.

Pada baris 20-27, <TextView android:id="@+id/tv_detail_title" ... /> adalah TextView untuk menampilkan judul film. Ini dikonfigurasi dengan ukuran teks besar, gaya tebal, dan diposisikan di bawah poster.

Pada baris 29-36, <TextView android:id="@+id/tv_detail_release_date" ... /> adalah TextView untuk menampilkan tanggal rilis film, diposisikan di bawah judul.

Pada baris 38-45, <TextView android:id="@+id/tv_detail_vote_average" ... /> adalah TextView untuk menampilkan rata-rata *voting* film, diposisikan di bawah tanggal rilis.

Pada baris 47-54, <TextView android:id="@+id/tv_detail_overview_label" ... /> adalah TextView sebagai label "Overview:". Ini dikonfigurasi dengan teks tebal dan diposisikan di bawah rata-rata *voting*.

Pada baris 56-62, <TextView android:id="@+id/tv_detail_overview" ... /> adalah TextView untuk menampilkan teks *overview* sebenarnya. Ini dikonfigurasi dengan ukuran teks normal dan diposisikan di bawah label *overview*.

**themes.xml dan themes.xml(night):**

Pada baris 1, <resources xmlns:tools="http://schemas.android.com/tools"> adalah tag root untuk file sumber daya, mendeklarasikan *namespace* Tools.

Pada baris 3, <style name="Base.Theme.MovieList" parent="Theme.Material3.DayNight"> mendefinisikan tema dasar aplikasi. name="Base.Theme.MovieList" adalah nama tema. parent="Theme.Material3.DayNight" adalah tema induk yang diwarisi. Theme.Material3.DayNight adalah tema Material Design 3 standar yang secara otomatis mendukung mode siang dan malam, dan yang penting, ia menyertakan ActionBar default yang diperlukan untuk judul dan tombol kembali di Activity.

Pada baris 6, <!-- <item name="colorPrimary">@color/my_light_primary</item> --> adalah komentar dan contoh bagaimana Anda bisa menyesuaikan atribut tema tertentu, seperti warna primer aplikasi.

Pada baris 9, <style name="Theme.MovieList" parent="Base.Theme.MovieList" /> adalah tema akhir yang sebenarnya digunakan oleh aplikasi. Ia mewarisi semua properti dari Base.Theme.MovieList. Ini adalah tema yang diterapkan secara *default* ke semua Activity kecuali jika Activity tersebut secara eksplisit menentukan tema lain di AndroidManifest.xml.

## Tautan Git

Berikut adalah tautan untuk semua source code yang telah dibuat.

natnutnot/PrakMobile at master + https://github.com/natnutnot/Mobile