

**LAPORAN AKHIR PRAKTIKUM
PEMROGAMAN MOBILE**



Oleh:

Damarjati Suryo Laksono NIM. 2310817210014

**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 : ViewModel 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 : Alysa Armelia

NIM : 2310817120009

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Zulfa Auliya Akbar
NIM. 2210817210026

Muti`a Maulida S.Kom M.T.I
NIP. 19881027 201903 20 13

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI.....	3
DAFTAR GAMBAR	5
DAFTAR TABEL.....	7
MODUL 1 : ANDROID BASIC WITH KOTLIN	10
SOAL 1	10
A. Source Code	12
B. Output Program.....	18
C. Pembahasan.....	19
MODUL 2 : ANDROID LAYOUT	23
SOAL 1	23
A. Source Code	24
B. Output Program.....	35
C. Pembahasan.....	37
MODUL 3: BUILD A SCROLLABLE LIST.....	44
SOAL 1	44
A. Source Code	46
B. Output Program.....	72
C. Pembahasan.....	74
SOAL 2	82
A. Pembahasan.....	82
MODUL 4 : VIEWMODEL AND DEBUGGING	83
SOAL 1	83
A. Source Code	83

B. Output Program.....	113
C. Pembahasan.....	116
SOAL 2	130
A. Pembahasan.....	130
MODUL 5 : Connect to the Internet	132
SOAL 1	132
A. Source Code	133
B. Output Program.....	176
C. Pembahasan.....	178
TAUTAN GIT.....	211

DAFTAR GAMBAR

Modul 1 : Android Basic With Kotlin

Gambar 1. 1. Tampilan Awal Aplikasi	10
Gambar 1. 2. Tampilan Roll Dadu Double	11
Gambar 1. 3. Screenshot Hasil Jawaban Soal 1	18
Gambar 1. 4. Screenshoot Horizontal Aplikasi.....	18

Modul 2 : Android Layout

Gambar 2. 1. Tampilan Awal Aplikasi	23
Gambar 2. 2. Tampilan Aplikasi Setelah Dijalankan.....	24
Gambar 2. 3. Screenshot Hasil Jawaban Soal 1 Awalan.....	35
Gambar 2. 4. Screenshot Hasil Jawaban Soal 1	36
Gambar 2. 5. Screenshot Hasil Jawaban Soal 1	36

MODUL 3: Build A Scrollable List

Gambar 3. 1. Contoh UI List.....	45
Gambar 3. 2. Gambar UI Detail	45
Gambar 3. 3. Screenshot Hasil Jawaban Soal 1	72
Gambar 3. 4. Screenshot Hasil Jawaban Soal 1	72
Gambar 3. 5. Screenshot tombol Detail	73
Gambar 3. 6. Screenshot tombol Info	73

Modul 4 : ViewModel And Debugging

Gambar 4. 1. Screenshot Hasil Jawaban Soal 1	113
Gambar 4. 2. Screenshot Hasil Jawaban Soal 1	114
Gambar 4. 3. Screenshot tombol Detail	114
Gambar 4. 4. Screenshot tombol Info	115
Gambar 4. 5. Screenshot list data.....	115
Gambar 4. 6. Screenshot Tombol Detail.....	115
Gambar 4. 7. Screenshot Debugger.....	115
Gambar 4. 8. Screenshot Debugger Tombol Detail	116

Gambar 4. 9. Contoh Penggunaan Debugger.....	130
----------------------------------------------	-----

MODUL 5 : Connect To The Internet

Gambar 5. 1. Screenshot Hasil Jawaban Soal 1	176
Gambar 5. 2. Screenshot Hasil Jawaban Soal 1	177
Gambar 5. 3. Screenshot tombol Detail	177
Gambar 5. 4. creenshot tombol Info.....	178

DAFTAR TABEL

Modul 1 : Android Basic With Kotlin

Tabel 1. 1. Source Code Jawaban Soal 1	14
Tabel 1. 2. Source Code Dice_ViewModel.kt	15
Tabel 1. 3. Source Code activity_main.xml	17

Modul 2 : Android Layout

Tabel 2. 1. Source Code Jawaban Soal 1	26
Tabel 2. 2. Source Code Jawaban Soal 1	28
Tabel 2. 3. Source Code Jawaban Soal 1	28
Tabel 2. 4. Source Code Jawaban Soal 1	32
Tabel 2. 5. Source Code Jawaban Soal 1	34

MODUL 3: Build A Scrollable List

Tabel 3. 1. Source Code Jawaban Soal 1	46
Tabel 3. 2. Source Code Jawaban Soal 1	49
Tabel 3. 2. Source Code Jawaban Soal 1	61
Tabel 3. 4. Source Code Jawaban Soal 1	63
Tabel 3. 5. Source Code Jawaban Soal 1	63
Tabel 3. 6. Source Code Jawaban Soal 1	65
Tabel 3. 7. Source Code Jawaban Soal 1	66
Tabel 3. 8. Source Code Jawaban Soal 1	67
Tabel 3. 9. Source Code Jawaban Soal 1	70
Tabel 3. 10. Source Code Jawaban Soal 1	71

Modul 4 : ViewModel And Debugging

Tabel 4. 1. Source Code MainActivity	84
Tabel 4. 2. Source Code FragmentGuweh	86
Tabel 4. 3. Source Code HomeFragment	89
Tabel 4. 4. Source Code HomeViewModel	101

Tabel 4. 5. Source Code HomeViewModelFactory	102
Tabel 4. 6. Source Code MyAdapter.....	104
Tabel 4. 7. Source Code MyData	104
Tabel 4. 8. Source Code detail_fragment.....	106
Tabel 4. 9. Source Code activity_main	107
Tabel 4. 10. Source Code home_fragment.....	108
Tabel 4. 11. Source Code item_list	111
Tabel 4. 12. Source Code nav_graph	113

MODUL 5 : Connect To The Internet

Tabel 5. 1. Source Code MyApiResponse	133
Tabel 5. 2. Source Code MyApiResponse	134
Tabel 5. 3. Source Code MyService.....	134
Tabel 5. 4. Source Code BookDao.....	136
Tabel 5. 5. Source Code BookDbEntity.....	137
Tabel 5. 6. Source Code AppDatabase.....	138
Tabel 5. 6. Source Code BookMapper.kt.....	140
Tabel 5. 8. Source Code BookRepositoryImpl	143
Tabel 5. 9. Source Code Book	144
Tabel 5. 10. Source Code BookRepository	145
Tabel 5. 11. Source Code GetBooksUseCase	145
Tabel 5. 12. Source Code BookUi	146
Tabel 5. 13. Source Code DetailFragment	150
Tabel 5. 14. Source Code HomeFragment	154
Tabel 5. 15. Source Code MainActivity.....	157
Tabel 5. 16. Source Code MyAdapter.....	160
Tabel 5. 17. Source Code BookViewModel	163
Tabel 5. 18. Source Code BookViewModelFactory	165
Tabel 5. 19. Source Code detail_fragment.xml.....	168
Tabel 5. 20. Source Code activity_main	169
Tabel 5. 21. Source Code home_fragment.....	171
Tabel 5. 22. Source Code item_list	174

Tabel 5. 23. Source Code nav_graph	176
------------------------------------------	-----

MODUL 1 : ANDROID BASIC WITH KOTLIN

SOAL 1

Soal Praktikum:

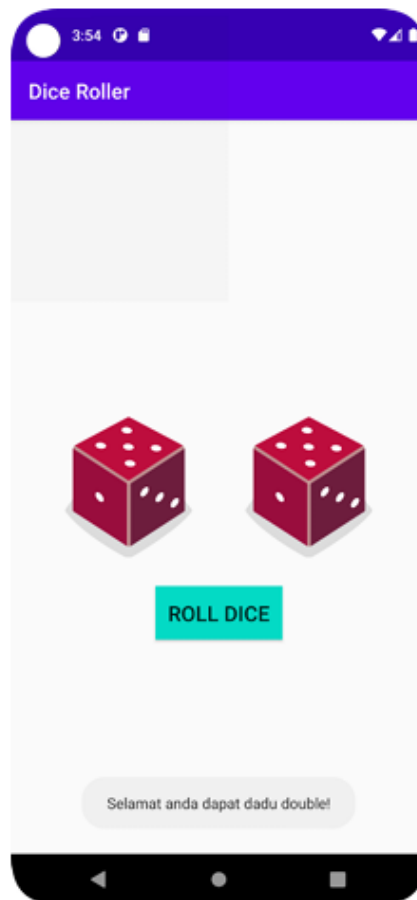
Buatlah sebuah aplikasi yang dapat menampilkan 2 (dua) buah dadu yang dapat berubah ubah tampilannya pada saat user menekan tombol “Roll Dice”. Aturan aplikasi yang akan dibangun adalah sebagaimana berikut:

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



Gambar 1. 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.
3. 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.
4. Untuk gambar dadu dapat didownload pada link berikut:
https://drive.google.com/u/0/uc?id=147HT2lIH5qin3z5ta7H9y2N_5OMW81Ll&export=download



Gambar 1. 2. Tampilan Roll Dadu Double

A. Source Code

1. MainActivity.kt

1	package com.example.diceroller
2	
3	import android.os.Bundle
4	import android.widget.Toast
5	import androidx.activity.viewModels
6	import androidx.appcompat.app.AppCompatActivity
7	import androidx.lifecycle.Observer
8	import
	com.example.diceroller.databinding.ActivityMainB
	inding
9	
10	class MainActivity : AppCompatActivity() {
11	
12	private lateinit var binding:
	ActivityMainBinding
13	private val viewModel: Dice_ViewModel by
	viewModels()
14	
15	override fun onCreate(savedInstanceState:
	Bundle?) {
16	super.onCreate(savedInstanceState)
17	
18	binding =
	ActivityMainBinding.inflate(layoutInflater)
19	setContentView(binding.root)
20	
21	binding.button.setOnClickListener {
22	viewModel.rollDice()
23	}
24	

25	viewModel.dice1.observe(this) { value ->
26	binding.imageView.setImageResource(getDiceImage(
27	value))
28	}
29	viewModel.dice2.observe(this) { value ->
30	binding.imageView2.setImageResource(getDiceImage
31	(value))
32	}
33	viewModel.isDouble.observe(this)
34	{ isDouble ->
35	val message = if (isDouble) {
36	"Selamat anda dapat dadu
37	double!"
38	} else {
39	"Anda belum beruntung!"
40	}
41	Toast.makeText(this, message,
42	Toast.LENGTH_SHORT).show()
43	}
44	}
45	private fun getDiceImage(number: Int): Int {
46	return when (number) {
47	1 -> R.drawable.dice_1
48	2 -> R.drawable.dice_2
49	3 -> R.drawable.dice_3
	4 -> R.drawable.dice_4
	5 -> R.drawable.dice_5

50	else -> R.drawable.dice_6
51	}
52	}
53	}

Tabel 1. 1. Source Code Jawaban Soal 1

2. Dice_ViewModel.kt

1	package com.example.diceroller
2	
3	import androidx.lifecycle.LiveData
4	import androidx.lifecycle.MutableLiveData
5	import androidx.lifecycle.ViewModel
6	
7	class Dice_ViewModel : ViewModel() {
8	
9	private val _dice1 = MutableLiveData(1)
10	val dice1: LiveData<Int> = _dice1
11	
12	private val _dice2 = MutableLiveData(1)
13	val dice2: LiveData<Int> = _dice2
14	
15	private val _isDouble =
	MutableLiveData(false)
16	val isDouble: LiveData<Boolean> = _isDouble
17	
18	fun rollDice() {
19	val d1 = (1..6).random()
20	val d2 = (1..6).random()
21	_dice1.value = d1
22	_dice2.value = d2
23	_isDouble.value = (d1 == d2)

24	}
25	}

Tabel 1. 2. Source Code Dice_ViewModel.kt

3. activity_main.xml

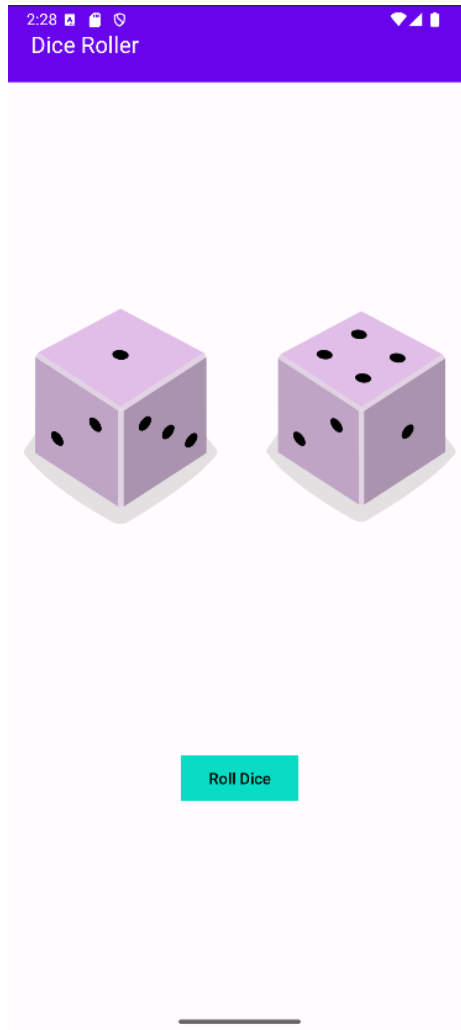
1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:id="@+id/main"
6	android:layout_width="match_parent"
7	android:layout_height="match_parent"
8	tools:context=".MainActivity">
9	
10	<TextView
11	android:id="@+id/headerTitle"
12	android:layout_width="0dp"
13	android:layout_height="wrap_content"
14	android:background="#6804ec"
15	android:padding="20dp"
16	android:text="@string/app_name"
17	android:textColor="#FFFFFF"
18	android:textSize="20sp"
19	app:layout_constraintEnd_toEndOf="parent"
20	app:layout_constraintHorizontal_bias="0.0"
21	app:layout_constraintStart_toStartOf="parent"
22	app:layout_constraintTop_toTopOf="parent" />
23	

24	<LinearLayout
25	android:id="@+id/diceContainer"
26	android:layout_width="wrap_content"
27	android:layout_height="wrap_content"
28	android:orientation="horizontal"
29	android:gravity="center"
30	app:layout_constraintTop_toBottomOf="@+id/header
	Title"
31	app:layout_constraintBottom_toTopOf="@+id/button
	"
32	app:layout_constraintStart_toStartOf="parent"
33	app:layout_constraintEnd_toEndOf="parent"
34	app:layout_constraintVertical_bias="0.5">
35	
36	<ImageView
37	android:id="@+id/imageView"
38	android:layout_width="wrap_content"
39	android:layout_height="wrap_content"
40	android:src="@drawable/dice_0" />
41	
42	<ImageView
43	android:id="@+id/imageView2"
44	android:layout_width="wrap_content"
45	android:layout_height="wrap_content"
46	android:src="@drawable/dice_0"
47	android:layout_marginStart="16dp" />
48	
49	</LinearLayout>
50	
51	<com.google.android.material.button.MaterialButt
	on
52	android:id="@+id/button"

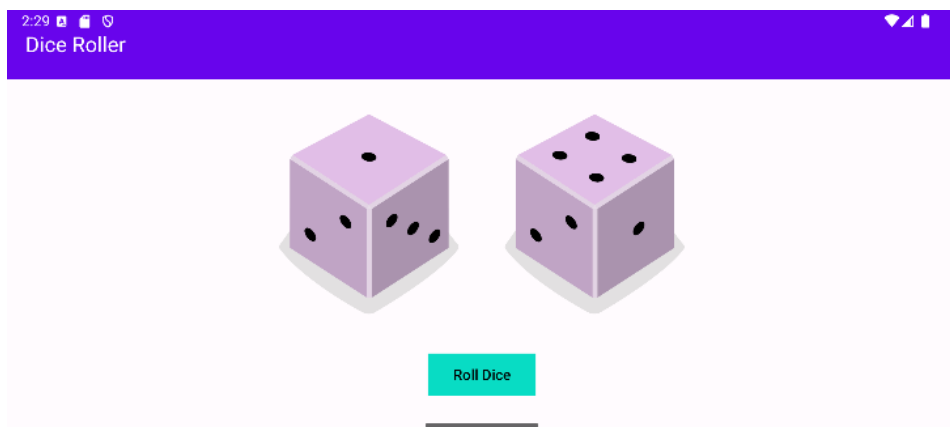
53	android:layout_width="wrap_content"
54	android:layout_height="wrap_content"
55	android:text="@string/button_1"
56	android:backgroundTint="#08dcc4"
57	android:textColor="@color/black"
58	app:cornerRadius="0dp"
59	app:layout_constraintTop_toBottomOf="@+id/diceCo ntainer"
60	app:layout_constraintBottom_toBottomOf="parent"
61	app:layout_constraintStart_toStartOf="parent"
62	app:layout_constraintEnd_toEndOf="parent"
63	app:layout_constraintVertical_bias="0.0"
64	android:layout_marginBottom="32dp" />
65	
66	</androidx.constraintlayout.widget.ConstraintLay out>

Tabel 1. 3. Source Code activity_main.xml

B. Output Program



Gambar 1. 3. Screenshot Hasil Jawaban Soal 1



Gambar 1. 4. Screenshoot Horizontal Aplikasi

C. Pembahasan

1. MainActivity.kt

Pada line 1-8 kita perlu melakukan deklarasi nama package file Kotlin agar dapat menentukan lokasi file sesuai struktur project. Dimana disini ada `import` yang berguna saat kita ingin mengimpor kelas-kelas dibutuhkan seperti, `Toast` untuk menampilkan pesan pop-up, `AppCompatActivity` sebagai activity dasar dalam Android, `viewModels()` untuk mengambil `ViewModel`, dan `ActivityMainBinding` untuk mengakses elemen layout menggunakan `View Binding`. Seperti di line [10], [12], [13], dan [53] kita perlu mendeklarasikan class dan properti. Yang mana disini ada class `MainActivity` merupakan activity utama atau untuk bagian halaman utama aplikasi, `binding` berguna saat menghubungkan layout XML dengan kode Kotlin, juga `viewModel` membantu menghubungkan ke class `Dice_ViewModel` yang berisi logika bagaimana cara melempar dadu tersebut. Lalu, di line [15], [16], dan [41] ada `onCreate()` karena fungsi ini dipanggil saat activity dibuat pertama kali dengan `savedInstanceState` berguna untuk menyimpan data saat kita melakukan rotasi layar, dll. Kemudian, di line [18] – [19] perlu melakukan inisialisasi layout karena menggunakan `View Binding` untuk mempermudah akses elemen UI jangan lupa disini ada `setContentView(binding.root)` berguna saat menampilkan layout XML ke layar. Selanjutnya, di line [21] – [22] adalah tombol untuk melempar dadu saat tombol tersebut ditekan nantinya fungsi `rollDice()` bagian `ViewModel` dipanggil. Dengan fungsi ini juga akan mengacak dua angka dadu dan `update LiveData`. Dilihat dari line [25] – [31] adalah kode agar bisa mengamati `LiveData` hal ini berguna untuk mengamati perubahan nilai `dice1` karena saat nilainya berubah, maka gambar dadu pertama (`imageView`) akan diperbarui sesuai angka didapat juga berlaku sama seperti kode di atas bagian `dice2` karena untuk dadu kedua (`imageView2`). Ada line [33] – [40] kita perlu menampilkan pesan `Toast`

untuk `Double` karena penggunaan `isDouble` adalah `LiveData<Boolean>` dari `ViewModel`, jika nilainya `true` (kedua dadu sama) yang ditampilkan pesan keberuntungan dan juga berlaku saat tidak sama akan ditampilkan pesan anda belum beruntung, serta `Toast.makeText(...)` berguna menampilkan pesan pendek di bawah layar. Setelah itu, line [43] – [52] berguna sebagai fungsi untuk mengubah angka jadi gambar karena fungsi ini akan mengembalikan ID gambar `R.drawable.dice_Xs` sesuai angka dadu (1-6) dengan bagian `observe` berperan untuk mengganti gambar dadu.

2. Dice_ViewModel.kt

Pada line [1] – [5] kita perlu melakukan deklarasi nama package file Kotlin agar dapat menentukan lokasi file sesuai struktur project. Dimana disini ada `import` yang berguna saat kita ingin mengimpor kelas-kelas dibutuhkan seperti, `LiveData`, `MutableLiveData` berguna untuk mengamati data secara otomatis dari UI, juga ada `ViewModel` adalah class dari Android Architecture Component untuk menyimpan dan mengelola data UI. Kemudian, di line [7] dan [25] kita perlu melakukan deklarasi class `ViewModel` karena `Dice_ViewModel` bagian dari `ViewModel` khusus untuk aplikasi ini bertujuan dalam menyimpan data serta logika melempar dadu agar tetap bertahan walaupun kita merotasi layarnya. Lalu, di line [9] – [13] `LiveData` untuk dadu pertama yang mana `_dice1` agar menyimpan angka dadu pertama secara default nilainya 1 melibatkan `MutableLiveData` untuk dapat dirubah dari dalam `ViewModel` dengan `dice1` akan dibuka sebagai `LiveData` memudahkan saat mengamati dari segi UI (tapi tidak bisa diubah dari luar). Juga disini mungkin pasti bingung kenapa dibagi dua (`_dice1` dan `dice1`)? karena ada enkapsulasi supaya hanya `ViewModel` yang bisa mengubah nilai dan bagian UI hanya bisa membaca, hal ini berlaku `LiveData` untuk dadu kedua agar menyimpan angka dadu kedua. Selanjutnya, di line [15] – [16] adalah `LiveData` untuk deteksi double maksudnya akan menyimpan nilai `true`

apabila kedua dadu nilainya sama dan nilai ini digunakan di UI agar bisa menampilkan toast keberuntungan. Berikutnya, di line [18] – [24] ada fungsi `rollDice()` yang akan mengacak dua angka antara 1 sampai 6 dengan menyimpan hasilnya ke `_dice1` dan `_dice2` dan perlu di cek lagi apakah hasilnya sama, jika ya maka set `_isDouble` ke `true` nantinya `LiveData` secara otomatis memberi tahu UI saat nilainya berubah.

3. `activity_main.xml`

Pada line [2] – [8] terdapat struktur utama layout terdiri dari `ConstraintLayout` merupakan Layout fleksibel yang bisa memberikan kemungkinan ke kita bagaimana cara mengatur posisi elemen UI berdasarkan hubungan antar elemen melibatkan `match_parent` untuk lebar dan tinggi layout mengikuti ukuran layar penuh dengan `tools:context=".MainActivity"` agar menunjukkan layout ini digunakan oleh `MainActivity`. Kemudian, lihat line [10] – [22] berguna sebagai `TextView` di Judul Aplikasi yang mana agar bisa menampilkan judul aplikasi menggunakan `@string/app_name` dari `strings.xml` dengan ukuran lebar `0dp` berarti mengikuti batas constraint kiri dan kanan (dari parent), `padding="20dp"` yaitu jarak dalam elemen, `background` ungu dan teks putih, juga posisi itu tempel ke bagian atas layar atau `Top_toTopOf="parent"`, serta menempel ke kiri dan kanan layar (Start & End). Lalu, di line [24] – [49] ada `LinearLayout` sebagai wadah gambar dadu disini nantinya berisi dua buah gambar dadu secara horizontal dengan `wrap_content` untuk ukurannya mengikuti isi melibatkan `gravity="center"` menjadi isi elemen diposisikan di Tengah dan posisi terletak di bawah `headerTitle`, di atas tombol atau button, dan diletakkan di tengah vertikal (`vertical_bias="0.5"` = tengah antara atas dan bawah). Selanjutnya, line [36] – [40] ada `ImageView` untuk menampilkan gambar dadu pertama yang mana gambar awal adalah `dice_0`. Berikutnya, line [42] – [47] ada `ImageView 2` untuk menampilkan gambar dadu kedua dengan diberi jarak ke kanan

ukurannya 16dp dari gambar dadu pertama agar tidak menempel. Setelah itu, line [51] – [64] ada `MaterialButton` adalah tombol digunakan untuk lempar dadu, teks diambil dari `strings.xml` atau `@string/button_1` yang warna latar tombol itu cyan (#08dcc4) menggunakan `cornerRadius="0dp"` agar tombol berbentuk kotak dengan posisi di bawah `diceContainer` dan tempel ke bagian bawah layar (`Bottom_toBottomOf="parent"`) ukuran margin 32dp.

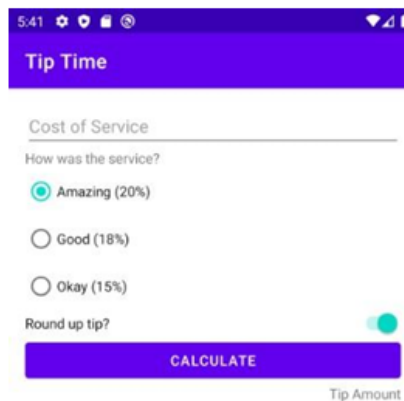
MODUL 2 : ANDROID LAYOUT

SOAL 1

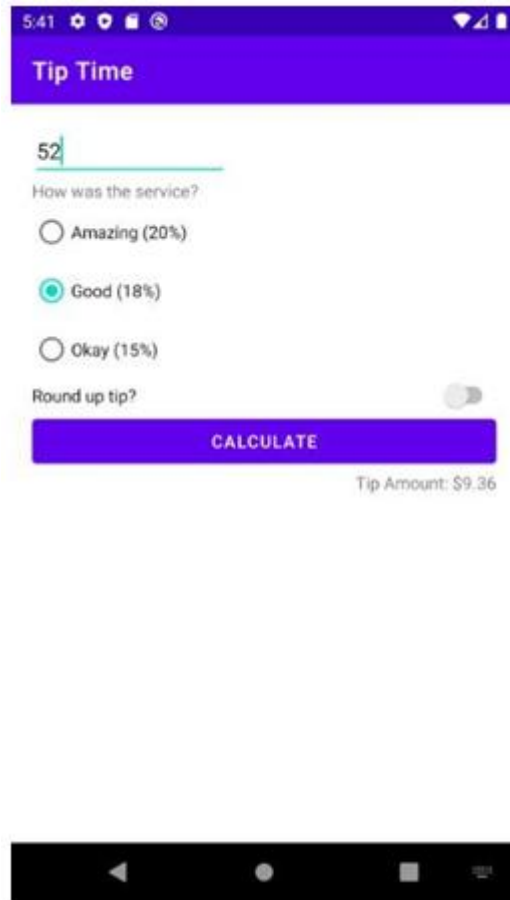
Soal Praktikum:

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 2. 1. Tampilan Awal Aplikasi



Gambar 2. 2. Tampilan Aplikasi Setelah Dijalankan

A. Source Code

1. MainActivity.kt

```

1 package com.example.kalkulatortip
2
3 import android.os.Bundle
4 import android.widget.Toast
5 import androidx.appcompat.app.AppCompatActivity
6 import androidx.lifecycle.ViewModelProvider
7 import
8 com.example.kalkulatortip.databinding.ActivityMa
  inBinding
9 import kotlin.math.ceil

```


10	class MainActivity : AppCompatActivity() {
11	
12	private lateinit var binding:
	ActivityMainBinding
13	private lateinit var viewModel: TipViewModel
14	
15	override fun onCreate(savedInstanceState:
	Bundle?) {
16	super.onCreate(savedInstanceState)
17	
18	binding =
	ActivityMainBinding.inflate(layoutInflater)
19	setContentView(binding.root)
20	
21	viewModel =
	ViewModelProvider(this) [TipViewModel::class.java
]
22	
23	viewModel.lastTipResult?.let {
24	binding.textLala.text = it
25	}
26	
27	binding.button.setOnClickListener {
28	calculateTip()
29	}
30	}
31	
32	private fun calculateTip() {
33	val costInput =
	binding.costInput.text.toString()
34	val cost = costInput.toDoubleOrNull()
35	

36	if (cost == null cost == 0.0) {
37	Toast.makeText(this, "Input tidak valid, silakan masukkan angka yang benar", Toast.LENGTH_SHORT).show()
38	val errorResult = getString(R.string.tip_result, 0.0)
39	binding.textLala.text = errorResult
40	viewModel.lastTipResult = errorResult
41	return
42	}
43	
44	val tipPercentage = when (binding.radioGroup.checkedRadioButtonId) {
45	R.id.radio_pirates -> 0.20
46	R.id.radio_ninjas -> 0.18
47	R.id.radio_Capucino -> 0.15
48	else -> 0.15
49	}
50	
51	var tip = cost * tipPercentage
52	if (binding.saveSwitch.isChecked) {
53	tip = ceil(tip)
54	}
55	
56	val result = getString(R.string.tip_result, tip)
57	binding.textLala.text = result
58	viewModel.lastTipResult = result
59	}
60	}

Tabel 2. 1. Source Code Jawaban Soal 1

2. SplashActivity.kt

1	package com.example.kalkulatortip
2	
3	import android.annotation.SuppressLint
4	import android.content.Intent
5	import android.os.Bundle
6	import android.os.Handler
7	import android.os.Looper
8	import androidx.appcompat.app.AppCompatActivity
9	import
	com.example.kalkulatortip.databinding.ActivitySpl
	ashBinding
10	
11	
12	@SuppressLint("CustomSplashScreen")
13	class SplashActivity : AppCompatActivity() {
14	
15	private lateinit var binding:
	ActivitySplashBinding
16	
17	override fun onCreate(savedInstanceState:
	Bundle?) {
18	super.onCreate(savedInstanceState)
19	
20	binding =
	ActivitySplashBinding.inflate(layoutInflater)
21	setContentView(binding.root)
22	
23	Handler(Looper.getMainLooper()).postDelayed({
24	goToMainActivity()
25	}, 3000L)
26	}

27	
28	private fun goToMainActivity() {
29	Intent(this, MainActivity::class.java).also{
30	startActivity(it)
31	finish()
32	}
33	}
34	}

Tabel 2. 2. Source Code Jawaban Soal 1

3. TipViewModel.kt

1	package com.example.kalkulatortip
2	
3	import androidx.lifecycle.ViewModel
4	
5	class TipViewModel : ViewModel() {
6	var lastTipResult: String? = null
7	}

Tabel 2. 3. Source Code Jawaban Soal 1

4. Activity_main.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<ScrollView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"

5	android:layout_width="match_parent"
6	android:layout_height="match_parent"
7	android:background="@color/white"
8	tools:context=".MainActivity">
9	
10	<androidx.constraintlayout.widget.ConstraintLayout
	ut
11	android:id="@+id/main"
12	android:layout_width="match_parent"
13	android:layout_height="wrap_content"
14	tools:context=".MainActivity">
15	
16	<TextView
17	android:id="@+id/textView"
18	android:layout_width="0dp"
19	android:layout_height="wrap_content"
20	android:background="#6804ec"
21	android:padding="20dp"
22	android:text="@string/app_name"
23	android:textColor="#FFFFFF"
24	android:textSize="20sp"
25	app:layout_constraintEnd_toEndOf="parent"
26	app:layout_constraintHorizontal_bias="0.0"
27	app:layout_constraintStart_toStartOf="parent"
28	app:layout_constraintTop_toTopOf="parent" />
29	
30	<com.google.android.material.textfield.TextInputLayout
31	android:id="@+id/username_input"
32	android:layout_width="375dp"
33	android:layout_height="wrap_content"
34	android:layout_marginTop="24dp"

35	android:hint="@string/app_Nono"
36	app:helperTextEnabled="true"
37	app:helperText="@string/app_Nene"
38	app:endIconMode="clear_text"
39	app:layout_constraintEnd_toEndOf="parent"
40	app:layout_constraintHorizontal_bias="0.497"
41	app:layout_constraintStart_toStartOf="parent"
42	app:layout_constraintTop_toBottomOf="@+id/textView">
43	
44	<com.google.android.material.textfield.TextInputEditText
45	android:id="@+id/cost_input"
46	android:layout_width="match_parent"
47	android:layout_height="wrap_content"
48	android:inputType="text" />
49	</com.google.android.material.textfield.TextInputLayout>
50	
51	
52	<RadioGroup
53	android:id="@+id/radioGroup"
54	android:layout_width="375dp"
55	android:layout_height="wrap_content"
56	android:orientation="vertical"
57	app:layout_constraintEnd_toEndOf="parent"
58	app:layout_constraintStart_toStartOf="parent"
59	app:layout_constraintTop_toBottomOf="@+id/username_input">
60	
61	<RadioButton
62	android:id="@+id/radio_pirates"

63	android:layout_width="wrap_content"
64	android:layout_height="wrap_content"
65	android:text="@string/app_legam" />
66	
67	<RadioButton
68	android:id="@+id/radio_ninjas"
69	android:layout_width="wrap_content"
70	android:layout_height="wrap_content"
71	android:text="@string/app_white" />
72	
73	<RadioButton
74	android:id="@+id/radio_Capucino"
75	android:layout_width="wrap_content"
76	android:layout_height="wrap_content"
77	android:text="@string/app_coklat" />
78	
79	</RadioGroup>
80	
81	<com.google.android.material.materialswitch.MaterialSwitch
82	android:id="@+id/save_switch"
83	android:layout_width="375dp"
84	android:layout_height="wrap_content"
85	android:checked="false"
86	android:text="@string/app_Nunu"
87	app:layout_constraintEnd_toEndOf="parent"
88	app:layout_constraintHorizontal_bias="0.497"
89	app:layout_constraintStart_toStartOf="parent"
90	app:layout_constraintTop_toBottomOf="@+id/radioGroup" />
91	
92	<com.google.android.material.button.MaterialButt

	on
93	android:id="@+id/button"
94	android:layout_width="375dp"
95	android:layout_height="wrap_content"
96	android:text="@string/app_Medium"
97	android:backgroundTint="#6804ec"
98	android:textColor="@color/white"
99	app:cornerRadius="5dp"
100	app:layout_constraintEnd_toEndOf="parent"
101	app:layout_constraintHorizontal_bias="0.497"
102	app:layout_constraintStart_toStartOf="parent"
103	app:layout_constraintTop_toBottomOf="@+id/save_s witch"
104	android:layout_marginBottom="32dp" />
105	
106	<TextView
107	android:id="@+id/textLala"
108	android:layout_width="0dp"
109	android:layout_height="wrap_content"
110	android:paddingEnd="20dp"
111	android:text="@string/app_Nana"
112	android:textColor="#4f4f4f"
113	android:textSize="14sp"
114	app:layout_constraintEnd_toEndOf="parent"
115	app:layout_constraintHorizontal_bias="0.0"
116	app:layout_constraintTop_toBottomOf="@+id/button "/>
117	
118	</androidx.constraintlayout.widget.ConstraintLay out>
119	</ScrollView>

Tabel 2. 4. Source Code Jawaban Soal 1

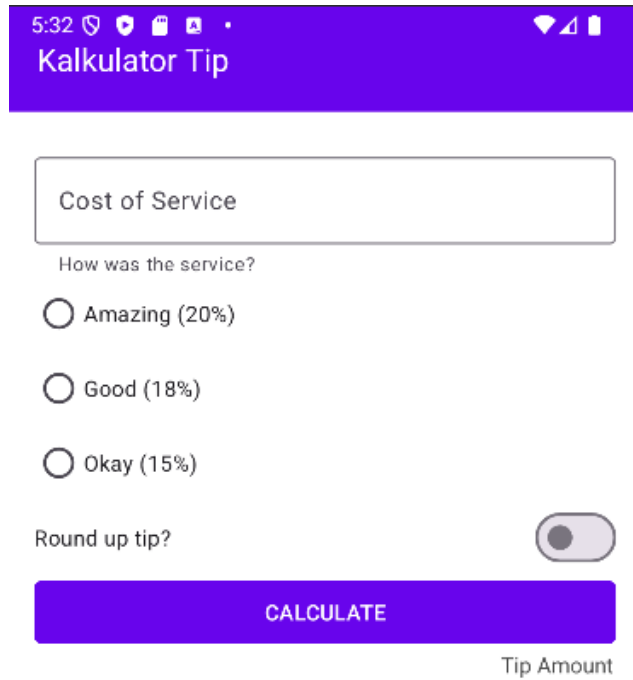
5. Activity_splash.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
3	android:layout_width="match_parent"
4	android:layout_height="match_parent"
5	xmlns:app="http://schemas.android.com/apk/res-auto"
6	xmlns:tools="http://schemas.android.com/tools"
7	tools:context=".SplashActivity"
8	android:id="@+id/splash"
9	android:background="@color/white">
10	
11	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
12	android:layout_width="wrap_content"
13	android:layout_height="wrap_content"
14	app:layout_constraintStart_toStartOf="parent"
15	app:layout_constraintEnd_toEndOf="parent"
16	app:layout_constraintTop_toTopOf="parent"
17	app:layout_constraintBottom_toBottomOf="parent">
18	<androidx.appcompat.widget.AppCompatImageView
19	android:id="@+id/iv_logo"
20	android:layout_width="200dp"
21	android:background="@color/white"
22	android:layout_height="200dp"
23	app:srcCompat="@drawable/amer_kalkulator"
24	app:layout_constraintStart_toStartOf="parent"
25	app:layout_constraintEnd_toEndOf="parent"
26	app:layout_constraintTop_toTopOf="parent"

27	app:layout_constraintBottom_toBottomOf="parent"/
	>
28	
29	</androidx.constraintlayout.widget.ConstraintLayout>
30	
31	<ProgressBar
32	style="@style/Widget.AppCompat.ProgressBar.Horizontal"
33	android:layout_width="200dp"
34	android:layout_height="32dp"
35	android:indeterminate="true"
36	app:layout_constraintStart_toStartOf="parent"
37	app:layout_constraintEnd_toEndOf="parent"
38	app:layout_constraintBottom_toBottomOf="parent"
39	android:indeterminateTint="#013B43"
40	tools:layout_editor_absoluteX="106dp"
41	tools:layout_editor_absoluteY="611dp" />
42	
43	</androidx.constraintlayout.widget.ConstraintLayout>

Tabel 2. 5. Source Code Jawaban Soal 1

B. Output Program



5:32

Kalkulator Tip

Cost of Service

How was the service?

☐ Amazing (20%)

☐ Good (18%)

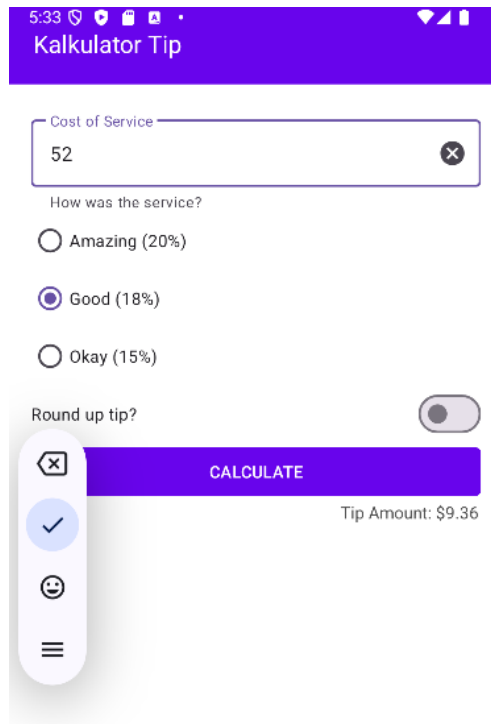
☐ Okay (15%)

Round up tip? ☐

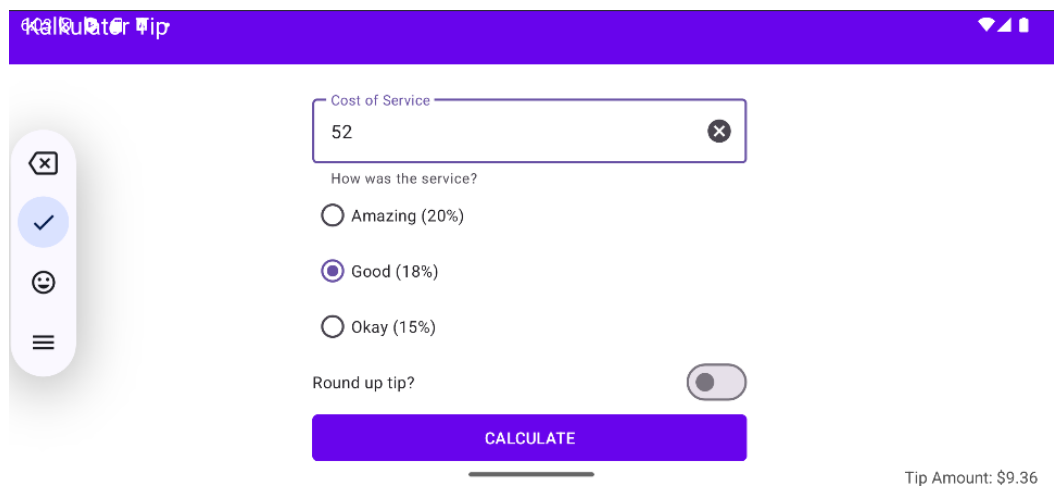
CALCULATE

Tip Amount

Gambar 2. 3. Screenshot Hasil Jawaban Soal 1 Awalan



Gambar 2. 4. Screenshot Hasil Jawaban Soal 1



Gambar 2. 5. Screenshot Hasil Jawaban Soal 1

C. Pembahasan

1. MainActivity.kt

Pada line [1] – [8] kita perlu melakukan deklarasi nama package file Kotlin agar dapat menentukan lokasi file sesuai struktur project. Dimana disini ada `import` yang berguna saat kita ingin mengimpor kelas-kelas dibutuhkan seperti, `Toast` untuk menampilkan pesan pop-up, `ViewModelProvider` berguna untuk menghubungkan `ViewModel` ke `activity`, `ActivityMainBinding` untuk mengakses view lewat `View Binding`, `ceil` bagian dari fungsi Kotlin untuk membulatkan angka ke atas.

Pada di line [10], [12], [13], dan [60] deklarasi juga inisialisasi ada di baris ini karena `MainActivity` turunan `AppCompatActivity` adalah komponen utama layar Android. Selain itu, ada `binding` berguna untuk mengakses elemen UI dari layout `activity_main.xml`. Serta, `viewModel` membantu dalam menyimpan data seperti hasil tip agar tetap tersedia saat rotasi layar.

Pada di line [15], [16], dan [41] ada `onCreate()` karena fungsi ini dipanggil saat `activity` dibuat pertama kali dengan `savedInstanceState` berguna untuk menyimpan data saat kita melakukan rotasi layar, dll. Setelah itu, di line [18] – [19] perlu melakukan inisialisasi layout karena menggunakan `View Binding` untuk mempermudah akses elemen UI jangan lupa disini ada `setContentView(binding.root)` berguna saat menampilkan layout XML ke layar. Selanjutnya, di line [21] ada `viewModel = ViewModelProvider(...)` berguna saat ingin menghubungkan `ViewModel` agar data tetap ada meskipun aktivitas sudah direstart. Berikutnya, di line [23] – [25] ada `viewModel.lastTipResult?.let` ketika ada hasil tip sebelumnya yang akan tampil di `textLala`. Serta, di line [27] – [29] `binding.button.setOnClickListener` digunakan saat tombol kita tekan nantinya bisa menjalankan fungsi `calculateTip()`.

Pada line [32] – [59] adalah fungsi `calculateTip()`. Kemudian, di line [32] – [34] berguna untuk mengambil input yang kita masukkan dan ubah ke tipe `Double`? jika tidak valid nantinya menjadi `null`. Lajutan ada di line [36] – [42] untuk bagian validasi input apabila kosong atau 0, tampilkan toast dan set hasil ke 0. Lalu, di line [49] – [49] untuk kita bisa pilih presentase tip sesuai `radio button` yang sudah dipilih. Selanjutnya, [51] – [54] adalah bagian bagaimana menghitung tip `biaya x persen` dengan kondisi jika `switch "save"` aktif nantinya bulatkan ke atas. Berikutnya, di line [56] – [58] kit berhasil membuat format hasil tip dengan string dari `strings.xml` yang nantinya ditampilkan ke UI dan simpan ke bagian `ViewModel`.

2. `Splash_Activity.kt`

Pada line [1] – [9] kita perlu melakukan deklarasi nama `package` file Kotlin agar dapat menentukan lokasi file sesuai struktur project. Dimana disini ada `import` yang berguna saat kita ingin mengimpor kelas-kelas dibutuhkan seperti, `SuppressLint` berguna att ingin menonaktifkan warning tertentu dari Android Studio, `Intent` untuk memudahkan ketika kita berpindah dari satu activity ke activity lain, `Handler` dan `Looper` untuk menjalankan kode dengan penundaan (`delay`), serta `ActivtySplashBinding` biasanya binding secara otomatis ke layout XML splash screen.

Pada line [12], [13], [15], dan [34] adalah deklarasi class juga bagian view binding menggunakan `SplashActivity` karena class yang ditampilkan pertama saat aplikasi dibuka dengan `@SuppressLint("CustomSplashScreen")` agar bisa menonaktifkan peringatan dari Android 12 ke atas punya `Splash API` bawaan, tapi di sini kita bikin manual melibatkann binding untuk bisa mengakses elemen UI di layout `activity_splash.xml`.

Pada line [17] – [26] adalah fungsi `onCreate()` yang dipanggil saat kita menggunakan activity pertama kali dibuat melibatkan binding

= ...inflate(layoutInflater) agar bisa menghubungkan layout splash screen menuju bagian activity dengan setContentView(binding.root) memudahkan saat menampilkan tampilan splash ke layar. Kemudian, Handler(...) berguna saat ingin menjalankan kode setelah delay (3 detik / 3000ms). Lalu, ada postDelayed({ ... }, 3000L) berguna untuk menunda fungsi goToMainActivity() selama 3 detik.

Pada line [28] – [33] adalah fungsi goToMainActivity() dengan membuat Intent berguna untuk berpindah dari SplashActivity menuju MainActivity dengan startActivity(it) agar bisa memulai activity utama dan ada finish() untuk menutup SplashActivity supaya tidak kembali ke splash saat kita menekan tombol back.

3. TipViewModel

Pada line [1] dan [3] kita perlu melakukan deklarasi nama package file Kotlin agar dapat menentukan lokasi file sesuai struktur project. Dimana disini ada import androidx.lifecycle.ViewModel berguna untuk mengimpor ViewModel dari Android Jetpack, bagian dari Android Architecture Components.

Pada line [5] – [7] adalah class TipViewModel karena disini ada class TipViewModel : ViewModel() berguna saat membuat class TipViewModel turunan dari class ViewModel melibatkan var lastTipResult: String? = null sebagai properti dalam menyimpan hasil terakhir kalkulasi tip atau perhitungan tip dengan tipe datanya String?, artinya bisa bernilai teks atau null hasilnya.

4. activity_main.xml

Pada line [1] – [8] juga [119] adalah bagian dari deklarasi awal dan ScrollView dimana ada <?xml ... ?> untuk deklarasi XML standar

dengan `<ScrollView>` sebagai layout vertikal yang bisa di scroll, digunakan agar semua konten di dalamnya tetap bisa diakses karena ada bagian yang tidak terlihat di layar saat kita rotate meskipun lebih tinggi dari layar tersebut. Lalu, di line [5] – [6] `android:layout_width="match_parent"` berguna sebagai lebar mengikuti layar juga ada `android:layout_height="match_parent"` untuk tingginya bisa mengikuti layar. Kemudian, `android:background="@color/white"` menjadi latar belakang putih. Serta, `tools:context=".MainActivity"` untuk menandakan file ini digunakan di `MainActivity` saat preview di Android Studio.

Pada line [10] – [14] juga [118] terdapat struktur utama layout terdiri dari `ConstraintLayout` merupakan Layout fleksibel yang bisa memberikan kemungkinan ke kita bagaimana cara mengatur posisi elemen UI berdasarkan hubungan antar elemen melibatkan `match_parent` untuk lebar dan tinggi layout mengikuti ukuran layar penuh yang menyesuaikan isi di dalamnya. Juga bagian ini cocok untuk desain kompleks dan responsif.

Pada line [16] – [28] adalah bagian judul aplikasi (`TextView`) untuk menampilkan nama aplikasi. Dimana ada `android:background="#6804ec"` sebagai background warna ungu tua dengan `android:textColor="#FFFFFF"` agar menjadi teks putih yang ukuran teksnya 20sp melibatkan `app:layout_constraintTop_toTopOf="parent"` biasanya ditempatkan di bagian atas layar dengan semua ukuran sisi padding 20dp.

Pada line [30] – [49] adalah bagian input Harga (`TextInputLayout + TextInputEditText`) karena ada komponen material design untuk form input lebih rapi dengan `TextInputLayout` dapat memberi fitur tambahan seperti, hint, helper text, dan ikon (`endIconMode="clear_text"` ikon silang untuk menghapus teks

dengan cepat). Kemudian, disini ada `TextInputEditText` sebagai field tempat kita memasukkan nilai biaya nantinya dihitung tip-nya. Lalu, ada Hint & helper text diambil dari `strings.xml` (`@string/app_Nono` dan `@string/app_Nene`) untuk teks bantu di bawah input.

Pada line [52] – [79] adalah bagian `RadioGroup`: Pilihan Tip. Ada `RadioGroup` berguna untuk kita atau pengguna memilih salah satu persentase tip dari 3 pilihan seperti, `radio_pirates = 20%`, `radio_ninjas = 18%`, dan `radio_Capucino = 15%` dengan hanya bisa memilih satu dari tiga radio button ini. Lalu, label teks diatur nantinya dari `strings.xml`, misal `@string/app_legam`, `@string/app_white`, dan `@string/app_coklat`.

Pada line [81] – [90] adalah bagian `materialSwitch` atau biasanya disebut Switch Bulat Modern (on/off) karena Switch ini digunakan untuk menentukan apakah tip perlu dibulatkan ke atas atau tidak oleh `ceil()` disini dengan berisi properti seperti, `android:checked="false"` untuk default-nya saat posisi off, `android:text="@string/app_Nunu"` sebagai label switch-nya, dan Layout-nya berada di bawah `RadioGroup`.

Pada line [92] – [104] adalah bagian `MaterialButton` atau tombol hitung tip untuk memicu jalannya kalkulasi tip (`button.setOnClickListener`) di kode kotlin kita buat dengan di dalamnya ada tampilan `android:text="@string/app_Medium"` sebagai label tombol, `android:backgroundTint="#6804ec"` agar menjadi warna ungu dengan `android:textColor="@color/white"` sebagai teks putih, dan `app:cornerRadius="5dp"` untuk sudutnya bisa agak membulat atau tombol sedikit melengkung.

Pada line [106] – [116] adalah bagian `TextView` hasil tip untuk menampilkan hasil tip setelah tombol kita klik karena digunakan sebagai

catatan atau disclaimer letaknya di bawah tombol melibatkan `android:textSize="14sp"` dan `android:textColor="#4f4f4f"` menjadi tampilan teks dengan warna abu-abu tua (`#4f4f4f`), ukuran kecil (`14sp`), teks ini dari `strings.xml`.

5. `activity_splash.xml`

Pada line [1] – [9] juga [43] adalah bagian `Root Layout`: `ConstraintLayout` merupakan Layout fleksibel yang bisa memberikan kemungkinan ke kita bagaimana cara mengatur posisi elemen UI berdasarkan hubungan antar elemen berdasarkan constraint melibatkan `match_parent` untuk lebar dan tinggi layout mengikuti ukuran layar penuh dengan `android:background="@color/white"` sebagai latar belakang layar splash putih serta ada `tools:context=".SplashActivity"` agar menunjukkan layout ini digunakan oleh `SplashActivity`.

Pada line [11] – [17] juga [29] adalah bagian `Inner ConstraintLayout` yang menjadi pusat konten karena layout ini digunakan agar bisa menempatkan logo yang sudah kita buat di tengah layar secara horizontal dan vertical melibatkan `wrap_content` berarti ukuran layout akan menyesuaikan kontennya dengan *Di-constraint* ke semua sisi (`Top`, `Bottom`, `Start`, `End`) ke parent, sehingga berada di tengah layar.

Pada line [18] – [27] adalah bagian logo aplikasi (`AppCompatActivity`) untuk menampilkan gambar atau logo aplikasi dengan ukuran `200x200dp` melibatkan `app:srcCompat="@drawable/amer_kalkulator"` agar bisa menampilkan gambar bernama `amer_kalkulator.png` atau `.xml` dari folder `res/drawable/` dimana background digunakan putih. Serta perlu *di-constraint* ke semua sisi layout atasnya, sehingga logo tetap berada di tengah layar.

Pada line [31] – [41] adalah bagian `ProgressBar` atau loading. Dimana `ProgressBar` horizontal berguna untuk menunjukkan bahwa aplikasi sedang ada di proses memuat. Kemudian, `indeterminate="true"` untuk tidak menunjukkan progress tertentu dengan hanya ada animasi loading terus menerus. Lalu, `indeterminateTint="#013B43"` sebagai warna loading bar-nya itu hijau toska dengan ditempatkan di bawah layar atau bottom constraint menuju parent.

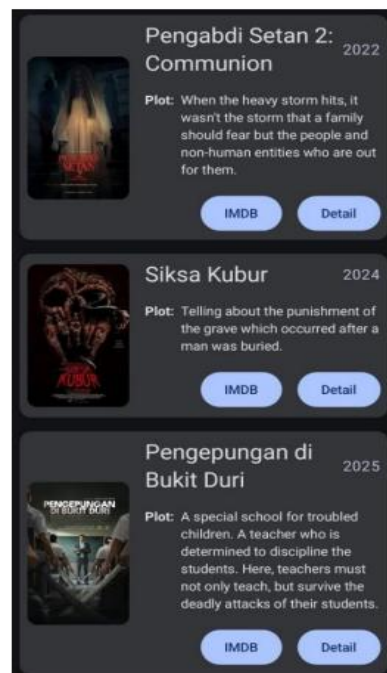
MODUL 3: BUILD A SCROLLABLE LIST

SOAL 1

Soal Praktikum:

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. 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

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 3. 1. Contoh UI List

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 3. 2. Gambar UI Detail

A. Source Code

1. MainActivity.kt

1	<code>package com.example.londondestination</code>
2	
3	<code>import android.os.Bundle</code>
4	<code>import androidx.appcompat.app.AppCompatActivity</code>
5	<code>import</code> <code>com.example.londondestination.databinding.Activi</code> <code>tyMainBinding</code>
6	
7	<code>class MainActivity : AppCompatActivity() {</code>
8	<code> private lateinit var binding:</code> <code>ActivityMainBinding</code>
9	<code> override fun onCreate(savedInstanceState:</code> <code>Bundle?) {</code>
10	
11	<code> super.onCreate(savedInstanceState)</code>
12	<code> binding =</code> <code>ActivityMainBinding.inflate(layoutInflater)</code>
13	<code> setContentView(binding.root)</code>
14	<code> }</code>
15	<code>}</code>

Tabel 3. 1. Source Code Jawaban Soal 1

2. FragmentGuweh.kt

1	<code>package com.example.londondestination</code>
2	
3	<code>import android.os.Bundle</code>
4	<code>import android.view.LayoutInflater</code>
5	<code>import android.view.View</code>
6	<code>import android.view.ViewGroup</code>
7	<code>import androidx.fragment.app.Fragment</code>

8	import
	com.example.londondestination.databinding.Detail
	FragmentBinding
9	
10	
11	class FragmentGuweh : Fragment() {
12	
13	private var _binding: DetailFragmentBinding?
	= null
14	private val binding get() = _binding!!
15	
16	override fun onCreateView(
17	inflater: LayoutInflater, container:
	ViewGroup?,
18	savedInstanceState: Bundle?
): View {
19	_binding =
	DetailFragmentBinding.inflate(inflater,
	container, false)
20	return binding.root
21	}
22	
23	override fun onViewCreated(view: View,
	savedInstanceState: Bundle?) {
24	super.onViewCreated(view,
	savedInstanceState)
25	
26	val imageResId =
	arguments?.getInt("imageResId") ?:
	R.drawable.ic_launcher_background
27	val nama =

	<code>arguments?.getString("nama") ?: "Nama tidak tersedia"</code>
28	<code>val deskripsi =</code>
	<code>arguments?.getString("deskripsi") ?: "Deskripsi tidak tersedia"</code>
29	
30	
31	<code>binding.detailImage.setImageResource(imageResId)</code>
32	<code>binding.detailTitle.text = nama</code>
33	<code>binding.detailDescription.text =</code>
	<code>deskripsi</code>
34	<code>}</code>
35	
36	<code>override fun onDestroyView() {</code>
37	<code>super.onDestroyView()</code>
38	<code>_binding = null</code>
39	<code>}</code>
40	
41	<code>companion object {</code>
42	<code>fun newInstance(imageResId: Int, nama: String, deskripsi: String): FragmentGuweh {</code>
43	<code>val fragment = FragmentGuweh()</code>
44	<code>val args = Bundle()</code>
45	<code>args.putInt("imageResId",</code>
	<code>imageResId)</code>
46	<code>args.putString("nama", nama)</code>
47	<code>args.putString("deskripsi",</code>
	<code>deskripsi)</code>
48	<code>fragment.arguments = args</code>
49	<code>return fragment</code>
50	<code>}</code>

51	}
52	}

Tabel 3. 2. Source Code Jawaban Soal 1

3. HomeFragment.kt

1	package com.example.londondestination
2	
3	import android.os.Bundle
4	import android.view.LayoutInflater
5	import android.view.View
6	import android.view.ViewGroup
7	import androidx.fragment.app.Fragment
8	import
	androidx.recyclerview.widget.LinearLayoutManager
9	import
	androidx.navigation.fragment.findNavController
10	import androidx.core.os.bundleOf
11	import
	com.example.londondestination.databinding.HomeFr
	agmentBinding
12	
13	class HomeFragment : Fragment() {
14	
15	private var _binding: HomeFragmentBinding? =
	null
16	private val binding get() = _binding!!
17	
18	private lateinit var adapter: MyAdapter
19	private val myDataList = listOf(
20	MyData(
21	nama = "Natural History Museum",
22	description = "Natural History

23	<p>Museum di London adalah destinasi wajib bagi pecinta sains, sejarah alam, dan keluarga dengan anak-anak. Museum ini memiliki lebih dari 80 juta spesimen, mencakup zoologi, paleontologi, botani, dan geologi. Daya tarik utama termasuk rangka dinosaurus, seperti Diplodocus dan model animatronik Tyrannosaurus Rex. Selain itu, pengunjung bisa melihat koleksi batuan langka, meteorit, serta kristal dan permata.\n" +</p> <p>"\n" + "Di Earth Galleries, ada pameran interaktif tentang geologi bumi, gunung berapi, dan gempa bumi. Museum ini juga memiliki zona edukasi yang menarik untuk anak-anak. Dibangun dengan arsitektur Romanesque bergaya Victoria, bangunannya memukau dengan interior yang megah. Yang menarik, museum ini gratis untuk dikunjungi, memberikan pengalaman yang tak terlupakan bagi semua pengunjung.",</p>
24	<p> descriptionsingkat = "Museum sejarah tentang alam",</p>
25	<p> year = 1881,</p>
26	<p> image =</p> <p>R.drawable.naturalhistorymuseum,</p>
27	<p> link =</p> <p>"https://en.wikipedia.org/wiki/Natural_History_Museum,_London"</p>
28	<p>),</p>
29	<p> MyData(</p>
30	<p> nama = "London Eye",</p>
31	<p> description = "London Eye adalah kincir raksasa setinggi 135 meter di tepi Sungai Thames, dan menjadi salah satu ikon paling</p>

	terkenal di London. Dari dalam kapsul kacanya, pengunjung bisa menikmati panorama kota yang menakjubkan, termasuk pemandangan Big Ben, Gedung Parlemen, dan Sungai Thames. Pada hari cerah, jarak pandang bisa mencapai hampir 40 kilometer.\n" +
32	<pre> "\n" + "Waktu terbaik untuk naik adalah saat senja, ketika cahaya kota mulai menyala dan langit berwarna keemasan karena menciptakan suasana romantis dan tenang. Setiap putaran berlangsung sekitar 30 menit, memberikan cukup waktu untuk mengagumi pemandangan, mengambil foto, atau sekadar menikmati momen dari ketinggian. London Eye bukan hanya atraksi wisata, tetapi juga simbol kebanggaan kota yang terus memikat baik turis maupun warga lokal.", </pre>
33	<pre> descriptionsingkat = "Komedi putar raksasa London", </pre>
34	<pre> year = 2000, </pre>
35	<pre> image = R.drawable.londoneye, </pre>
36	<pre> link = "https://en.wikipedia.org/wiki/London_Eye" </pre>
37	<pre>), </pre>
38	<pre> MyData(</pre>
39	<pre> nama = "British Museum", </pre>
40	<pre> description = "British Museum adalah destinasi luar biasa bagi pecinta sejarah dan arkeologi, dengan koleksi global yang mencakup ribuan tahun peradaban manusia. Begitu masuk, suasana elegan langsung membawa pengunjung seakan menjelajahi masa lalu dari mumi Mesir, patung Yunani, artefak Mesopotamia, hingga </pre>

	keramik Tiongkok dan seni Islam. Batu Rosetta adalah salah satu sorotan utama, bersama Patung Ramses II dan reruntuhan Kuil Parthenon yang ikonik.\n" +
41	<pre> "\n" + "Setiap koleksi disertai penjelasan informatif yang memudahkan pengunjung memahami konteks sejarahnya. Great Court, dengan atap kaca yang terang dan desain modern, menjadi pusat bangunan yang menawan dan nyaman untuk bersantai. British Museum bukan sekadar tempat melihat artefak, tapi juga ruang kontemplatif yang menghubungkan kita dengan jejak panjang umat manusia dan semuanya bisa dinikmati tanpa biaya masuk.", </pre>
42	<pre> descriptionsingkat = "Museum koleksi dunia", </pre>
43	<pre> year = 1753, </pre>
44	<pre> image = R.drawable.british_museum, </pre>
45	<pre> link = "https://en.wikipedia.org/wiki/British_Museum" </pre>
46	<pre>), </pre>
47	<pre> MyData(</pre>
48	<pre> nama = "Big Ben", </pre>
49	<pre> description = "Big Ben adalah ikon tak tergantikan London, berdiri megah di samping Gedung Parlemen di tepi Sungai Thames. Meski banyak mengira namanya merujuk pada menaranya, Big Ben sebenarnya adalah lonceng besar seberat lebih dari 13 ton di dalam Elizabeth Tower adalah nama resmi menara tersebut, yang diberikan untuk menghormati Ratu Elizabeth II.\n" + </pre>

50	"\n" + "Dentang Big Ben punya makna emosional yang dalam, sering terdengar dalam momen penting seperti pergantian tahun atau peringatan nasional, dan bahkan disiarkan BBC sejak 1920-an. Arsitekturnya yang anggun menjadi latar favorit para wisatawan, baik saat disinari mentari pagi maupun diterangi lampu malam hari. Walau akses ke dalam menara terbatas, cukup berdiri di dekatnya sudah membuat pengunjung merasa terhubung dengan sejarah dan semangat kota London yang tak lekang waktu",
51	descriptionsingkat = "Jam besar London",
52	year = 1859,
53	image = R.drawable.bigben,
54	link = "https://en.wikipedia.org/wiki/Big_Ben"
55),
56	MyData(nama = "Buckingham Palace",
57	description = "Buckingham Palace adalah simbol monarki Inggris yang berdiri megah di pusat London, berfungsi sebagai kediaman resmi Raja dan pusat berbagai acara kenegaraan. Dengan lebih dari 700 ruangan, istana ini mencerminkan kemewahan dan sejarah yang hidup, bahkan dari luar pagar hitamnya yang ikonik.\n"
58	+ "\n" + "Salah satu atraksi utama adalah Upacara Pergantian Penjaga, prosesi tradisional dengan seragam merah dan musik
59	

69	<pre> halaman eksekusi.\n" + "\n" + "Namun, sisi gelap itu berpadu dengan kemewahan karena di sinilah Permata Mahkota Inggris disimpan, termasuk mahkota dan tongkat kerajaan yang berkilau menakjubkan. Kontras antara sejarah kelam dan simbol kejayaan membuat kunjungan ke Tower of London terasa seperti menyusuri lorong waktu, menghadirkan pengalaman mendalam yang tak terlupakan.", </pre>
70	<pre> descriptionsingkat = "Benteng mempunyai banyak sejarah", </pre>
71	<pre> year = 1066, </pre>
72	<pre> image = R.drawable.toweroflondon, </pre>
73	<pre> link = "https://en.wikipedia.org/wiki/Tower_of_London" </pre>
74	<pre>), </pre>
75	<pre> MyData(</pre>
76	<pre> nama = "Warner Bros. Studio Tour London", </pre>
77	<pre> description = "Warner Bros. Studio Tour London adalah destinasi impian bagi penggemar Harry Potter, menawarkan pengalaman imersif ke dunia sihir yang sebelumnya hanya bisa dilihat di layar. Begitu masuk, kamu akan dibawa ke set asli seperti Great Hall, Diagon Alley, dan Privet Drive dimana semuanya penuh detail yang membuatmu merasa benar-benar berada di dunia Hogwarts.\n" + </pre>
78	<pre> "\n" + "Selain menjelajahi lokasi ikonik, pengunjung juga bisa melihat properti film seperti Horcrux, kostum rumah- </pre>

	rumah Hogwarts, hingga proses pembuatan efek visual yang menghidupkan sihir di layar. Setiap sudut studio menyuguhkan keajaiban yang membuat kamu makin menghargai imajinasi dan kerja keras di balik film. Dan tentu saja, mencicipi Butterbeer jadi penutup manis dari kunjungan yang terasa seperti pulang ke dunia masa kecil yang penuh keajaiban.",
79	descriptionsingkat = "Studio Harry Potter London",
80	year = 2012,
81	image =
	R.drawable.harrypotterstudio,
82	link =
	"https://www.wbstudiotour.co.uk/"
83),
84	MyData(
85	nama = "Hyde Park",
86	description = "Hyde Park adalah oase hijau di tengah London yang menawarkan ketenangan dan ruang bebas bagi siapa saja, dari warga lokal hingga turis. Dengan luas lebih dari 140 hektar, taman ini menjadi tempat ideal untuk jogging, bersepeda, membaca buku, atau sekadar duduk santai di tepi danau Serpentine.\n" +
87	"\n" + "Suasananya santai dan cocok buat piknik, bermain, atau menikmati kopi di bawah rindangnya pepohonan. Salah satu sudut paling unik adalah Speaker's Corner, simbol kebebasan berpendapat di mana siapa pun bisa berbicara di depan umum. Selain sebagai tempat pelarian dari hiruk pikuk kota, Hyde Park

	juga kerap menjadi lokasi konser besar dan festival, menjadikannya ruang publik yang dinamis dan menyatu dengan jiwa kota London.",
88	descriptionsingkat = "Taman pusat kota",
89	year = 1637,
90	image = R.drawable.hydepark,
91	link =
	"https://en.wikipedia.org/wiki/Hyde_Park,_London"
92),
93	MyData(
94	nama = "The Sherlock Holmes Museum",
95	description = "The Sherlock Holmes Museum di 221B Baker Street adalah surga bagi penggemar detektif legendaris ini, membawa pengunjung langsung ke dunia fiksi era Victoria yang terasa hidup. Interiornya merekonstruksi ruang kerja dan rumah Holmes secara detail-lengkap dengan perapian, kaca pembesar, dan barang-barang pribadi khas karakter dalam cerita.\n" +
96	"\n" + "Setiap sudut museum dirancang agar kamu seolah benar-benar berada di tengah kisah misteri bersama Holmes dan Watson. Lebih dari sekadar pameran, museum ini menawarkan pengalaman yang memuaskan rasa ingin tahu para penggemar dan pencinta cerita klasik.",
97	descriptionsingkat = "Museum seorang detektif yang ikonik",
98	year = 1990,

99	image =
	R.drawable.thesherlockholmesmuseum,
100	link = "https://www.sherlock-
	holmes.co.uk/"
101),
102	MyData(
103	nama = "St Paul's Cathedral",
104	description = "St Paul's Cathedral
	adalah salah satu ikon arsitektur paling
	menakjubkan di London, dikenal dengan kubah
	raksasanya yang mendominasi cakrawala kota. Dari
	luar terlihat megah, tapi keindahan sejatinya
	baru benar-benar terasa saat kamu melangkah
	masuk—ruang dalamnya hening, agung, dan sarat
	nuansa spiritual. Langit-langit tinggi, kaca
	patri indah, dan cahaya alami menciptakan
	atmosfer yang membuat siapa pun terdiam dalam
	kekaguman.\n" +
105	"\n" + "Salah satu
	pengalaman paling tak terlupakan di sini adalah
	menaiki ratusan anak tangga menuju puncak kubah.
	Dari atas, kamu bisa menikmati panorama kota
	London yang luas dan penuh sejarah, dari Sungai
	Thames hingga gedung-gedung modern yang berdiri
	berdampingan dengan bangunan bersejarah. Tak
	heran, St Paul's sering menjadi lokasi berbagai
	momen penting nasional, karena tempat ini bukan
	hanya gereja, tapi simbol kekuatan dan keindahan
	yang hidup modern di kota.",
106	descriptionsingkat = "Katedral
	bersejarah di London",
107	year = 1710,

108	image = R.drawable.stpaulcathedral,
109	link =
	"https://en.wikipedia.org/wiki/St_Paul%27s_Cathe
	dral"
110),
111	MyData(
112	nama = "Camden Market",
113	description = "Camden Market adalah
	salah satu tempat paling unik dan penuh karakter
	di London, ideal bagi kamu yang suka berburu
	barang-barang tak biasa dan merasakan suasana
	kota yang dinamis. Pasar ini dipenuhi toko-toko
	kecil yang menjual pakaian vintage, aksesoris
	handmade, dan berbagai barang nyentrik yang
	sering kali hanya ada satu di dunia. Suasananya
	ramai tapi seru, dengan musik jalanan, aroma
	makanan internasional, dan gaya busana
	pengunjung yang beragam.\n" +
114	"\n" + "Selain jadi pusat
	belanja, Camden juga merupakan ruang ekspresi
	subkultur alternatif seperti punk, goth, dan
	hippie—tempat di mana semua orang bisa tampil
	sesuai dirinya sendiri. Kalau lapar, pilihan
	kulinernya luar biasa banyak dan menggoda, dari
	makanan Asia, Latin, Timur Tengah, sampai fusion
	kreatif. Setiap kunjungan ke Camden terasa
	seperti eksplorasi baru bukan sekadar belanja,
	tapi pengalaman hidup kota London yang bebas,
	kreatif, dan penuh kejutan.",
115	descriptionsingkat = "Pasar terbesar
	di London",
116	year = 1974,

117	image = R.drawable.camdenmarket,
118	link =
	"https://en.wikipedia.org/wiki/Camden_Market"
119)
120)
121	
122	override fun onCreateView(inflater:
	LayoutInflater, container: ViewGroup?,
123	savedInstanceState: Bundle?
): View {
124	_binding =
	HomeFragmentBinding.inflate(inflater, container,
	false)
125	return binding.root
126	}
127	
128	override fun onViewCreated(view: View,
	savedInstanceState: Bundle?) {
129	super.onViewCreated(view,
	savedInstanceState)
130	
131	adapter = MyAdapter(myDataList)
	{ selectedItem ->
132	val bundle = bundleOf(
133	"imageResId" to
	selectedItem.image,
134	"nama" to selectedItem.nama,
135	"deskripsi" to
	selectedItem.description
136)
137	
138	findNavController().navigate(R.id.action_HomeFra

	<code>gment_to_detailFragment, bundle)</code>
139	<code> }</code>
140	
141	<code> binding.rvCharacter.layoutManager =</code>
	<code>LinearLayoutManager(requireContext())</code>
142	<code> binding.rvCharacter.adapter = adapter</code>
143	<code> }</code>
144	
145	<code> override fun onDestroyView() {</code>
146	<code> super.onDestroyView()</code>
147	<code> _binding = null</code>
148	<code> }</code>
149	<code>}</code>

Tabel 3. 3. Source Code Jawaban Soal 1

4. MyAdapter.kt

1	<code>package com.example.londondestination</code>
2	
3	<code>import android.content.Intent</code>
4	<code>import android.net.Uri</code>
5	<code>import android.view.LayoutInflater</code>
6	<code>import android.view.ViewGroup</code>
7	<code>import androidx.recyclerview.widget.RecyclerView</code>
8	<code>import</code>
	<code>com.example.londondestination.databinding.ItemLi</code>
	<code>stBinding</code>
9	
10	<code>class MyAdapter(</code>
11	<code> private val destinations: List<MyData>,</code>
12	<code> private val onDetailClick: (MyData) -> Unit</code>
	<code>) : RecyclerView.Adapter<MyAdapter.ViewHolder>()</code>
13	<code>{</code>

14	
15	inner class ViewHolder(val binding:
	ItemListBinding) :
16	RecyclerView.ViewHolder(binding.root)
17	
18	override fun onCreateViewHolder(parent:
	ViewGroup, viewType: Int): ViewHolder {
19	val binding = ItemListBinding.inflate(
20	LayoutInflater.from(parent.context),
21	parent,
22	false
23)
24	return ViewHolder(binding)
25	}
26	
27	override fun onBindViewHolder(holder:
	ViewHolder, position: Int) {
28	val destination = destinations[position]
29	with(holder.binding) {
30	textViewName.text = destination.nama
31	textViewYear.text =
	destination.year.toString()
32	textViewDesc.text =
	destination.descriptionsingkat
33	imageView.setImageResource(destination.image)
34	
35	buttonLink.setOnClickListener {
36	val context = it .context
37	val intent =
	Intent(Intent.ACTION_VIEW,
	Uri.parse(destination.link))

38	context.startActivity(intent)
39	}
40	
41	buttonDetail.setOnClickListener {
42	onDetailClick(destination)
43	}
44	}
45	}
46	
47	override fun getItemCount(): Int =
	destinations.size
48	}

Tabel 3. 4. Source Code Jawaban Soal 1

5. MyData.kt

1	package com.example.londondestination
2	
3	import android.os.Parcelable
4	import kotlinx.parcelize.Parcelize
5	
6	@Parcelize
7	data class MyData(
8	val nama: String,
9	val description: String,
10	val descriptionsingkat: String,
11	val year: Int,
12	val image: Int,
13	val link: String
14): Parcelable

Tabel 3. 5. Source Code Jawaban Soal 1

6. Detail_fragment.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<ScrollView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	android:id="@+id/detailScrollView"
5	android:layout_width="match_parent"
6	android:layout_height="match_parent"
7	android:padding="16dp">
8	
9	<androidx.cardview.widget.CardView
10	android:layout_width="match_parent"
11	android:layout_height="wrap_content"
12	app:cardCornerRadius="16dp"
13	app:cardElevation="8dp">
14	
15	<LinearLayout
16	android:layout_width="match_parent"
17	android:layout_height="wrap_content"
18	android:orientation="vertical">
19	
20	<ImageView
21	android:id="@+id/detailImage"
22	android:layout_width="match_parent"
23	android:layout_height="200dp"
24	android:scaleType="centerCrop"/>
25	
26	<TextView
27	android:id="@+id/detailTitle"
28	android:layout_width="match_parent"

29	android:layout_height="wrap_content"
30	android:text="Nama Tempat"
31	android:textStyle="bold"
32	android:textSize="20sp"
33	android:padding="16dp" />
34	
35	<TextView
36	android:id="@+id/detailDescription"
37	android:layout_width="match_parent"
38	android:layout_height="wrap_content"
39	android:text="Deskripsi lengkap tempat destinasi."
40	android:paddingStart="16dp"
41	android:paddingEnd="16dp"
42	android:paddingBottom="16dp"
43	android:textSize="16sp" />
44	
45	</LinearLayout>
46	</androidx.cardview.widget.CardView>
47	</ScrollView>

Tabel 3. 6. Source Code Jawaban Soal 1

7. activity_main.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayo
3	ut
	xmlns:android="http://schemas.android.com/apk/re
	s/android"
4	xmlns:app="http://schemas.android.com/apk/res-
	auto"
5	xmlns:tools="http://schemas.android.com/tools"

6	<code>android:id="@+id/main_layout"</code>
7	<code>android:layout_width="match_parent"</code>
8	<code>android:layout_height="match_parent"</code>
9	<code>tools:context=".MainActivity"></code>
10	
11	<code><androidx.fragment.app.FragmentContainerView</code>
12	<code>android:id="@+id/fragment_container_view"</code>
13	<code>android:name="androidx.navigation.fragment.NavHostFragment"</code>
14	<code>android:layout_width="match_parent"</code>
15	<code>android:layout_height="match_parent"</code>
16	<code>app:navGraph="@navigation/nav_graph"</code>
17	<code>app:defaultNavHost="true" /></code>
18	<code></androidx.constraintlayout.widget.ConstraintLayout></code>

Tabel 3. 7. Source Code Jawaban Soal 1

8. home_fragment.xml

1	<code><?xml version="1.0" encoding="utf-8"?></code>
2	<code><androidx.constraintlayout.widget.ConstraintLayout</code>
	<code>xmlns:android="http://schemas.android.com/apk/res/android"</code>
3	<code>xmlns:app="http://schemas.android.com/apk/res-auto"</code>
4	<code>xmlns:tools="http://schemas.android.com/tools"</code>
5	<code>android:layout_width="match_parent"</code>
6	<code>android:layout_height="match_parent"</code>
7	<code>tools:context=".HomeFragment"></code>
8	
9	<code><androidx.recyclerview.widget.RecyclerView</code>
10	<code>android:id="@+id/rv_character"</code>

11	android:layout_width="0dp"
12	android:layout_height="0dp"
13	app:layout_constraintBottom_toBottomOf="parent"
14	app:layout_constraintEnd_toEndOf="parent"
15	app:layout_constraintStart_toStartOf="parent"
16	app:layout_constraintTop_toTopOf="parent" />
17	</androidx.constraintlayout.widget.ConstraintLayout>

Tabel 3. 8. Source Code Jawaban Soal 1

9. item_list.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.cardview.widget.CardView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:layout_width="match_parent"
6	android:layout_height="wrap_content"
7	android:layout_margin="12dp"
8	app:cardCornerRadius="16dp"
9	app:cardElevation="6dp">
10	
11	<androidx.constraintlayout.widget.ConstraintLayout
	ut
12	android:layout_width="match_parent"
13	android:layout_height="wrap_content"
14	android:padding="12dp">
15	
16	<ImageView
17	android:id="@+id/imageView"

18	android:layout_width="100dp"
19	android:layout_height="150dp"
20	android:scaleType="centerCrop"
21	app:layout_constraintTop_toTopOf="parent"
22	app:layout_constraintStart_toStartOf="parent" />
23	
24	<TextView
25	android:id="@+id/textViewName"
26	android:layout_width="0dp"
27	android:layout_height="wrap_content"
28	android:text="Main Title"
29	android:textStyle="bold"
30	android:textSize="18sp"
31	android:textColor="#000000"
32	android:layout_marginStart="12dp"
33	android:layout_marginTop="8dp"
34	app:layout_constraintStart_toEndOf="@id/imageVie
	w"
35	app:layout_constraintEnd_toEndOf="parent"
36	tools:ignore="MissingConstraints" />
37	
38	<TextView
39	android:id="@+id/textViewYear"
40	android:layout_width="0dp"
41	android:layout_height="wrap_content"
42	android:text="1990"
43	android:textSize="16sp"
44	android:textColor="#555555"
45	android:layout_marginStart="12dp"
46	android:layout_marginTop="4dp"
47	app:layout_constraintStart_toEndOf="@id/imageVie
	w"

48	app:layout_constraintTop_toBottomOf="@id/textViewName"
49	app:layout_constraintEnd_toEndOf="parent" />
50	
51	<TextView
52	android:id="@+id/textViewDesc"
53	android:layout_width="0dp"
54	android:layout_height="wrap_content"
55	android:text="Secondary description text that can span multiple lines."
56	android:textSize="15sp"
57	android:textColor="#555555"
58	android:layout_marginStart="12dp"
59	android:layout_marginTop="8dp"
60	app:layout_constraintStart_toEndOf="@id/imageView"
61	app:layout_constraintTop_toBottomOf="@id/textViewYear"
62	app:layout_constraintEnd_toEndOf="parent" />
63	
64	<LinearLayout
65	android:id="@+id/buttonRow"
66	android:layout_width="wrap_content"
67	android:layout_height="wrap_content"
68	android:orientation="horizontal"
69	android:gravity="center"
70	android:layout_marginTop="16dp"
71	app:layout_constraintTop_toBottomOf="@id/imageView"
72	app:layout_constraintBottom_toBottomOf="parent"
73	app:layout_constraintStart_toStartOf="parent"
74	app:layout_constraintEnd_toEndOf="parent">

75	
76	<Button
77	android:id="@+id/buttonLink"
78	android:layout_width="wrap_content"
79	android:layout_height="wrap_content"
80	android:layout_marginEnd="8dp"
81	android:backgroundTint="@color/purple_200"
82	android:text="Detail"
83	android:textAllCaps="false" />
84	
85	<Button
86	android:id="@+id/buttonDetail"
87	android:layout_width="wrap_content"
88	android:layout_height="wrap_content"
89	android:layout_marginStart="8dp"
90	android:backgroundTint="@color/purple_200"
91	android:text="Info"
92	android:textAllCaps="false" />
93	</LinearLayout>
94	
95	</androidx.constraintlayout.widget.ConstraintLayout>
96	</androidx.cardview.widget.CardView>

Tabel 3. 9. Source Code Jawaban Soal 1

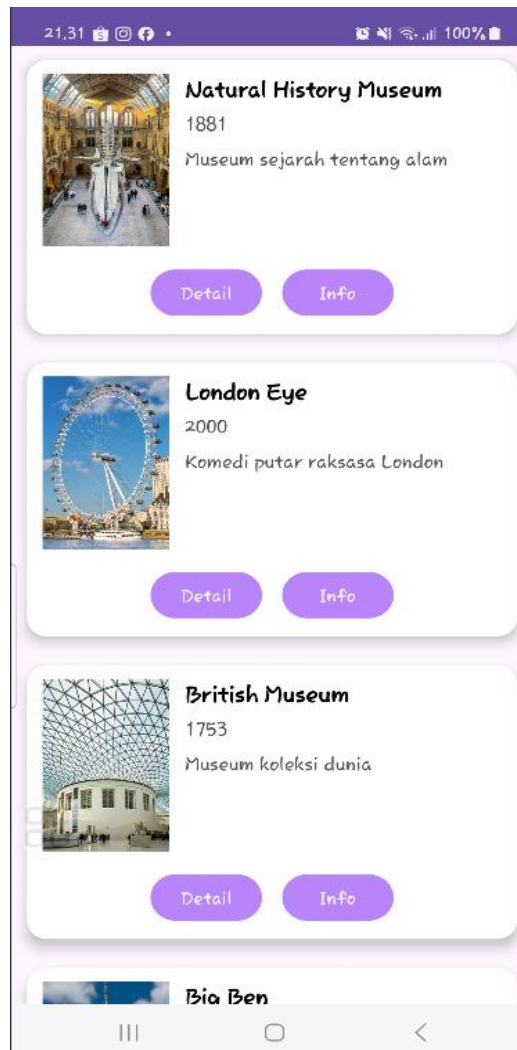
10. nav_graph.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<navigation
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"

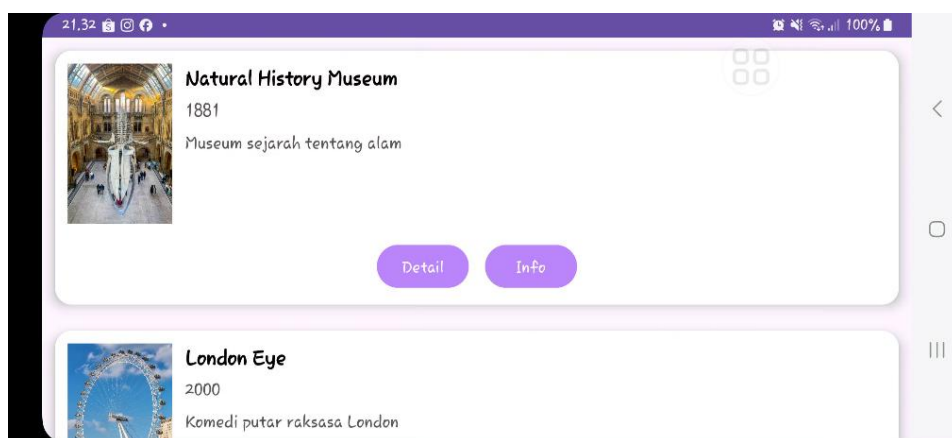
4	android:id="@+id/nav_graph"
5	app:startDestination="@id/HomeFragment">
6	
7	<fragment
8	android:id="@+id/HomeFragment"
9	android:name="com.example.londondestination.Home Fragment"
10	android:label="HomeFragment" >
11	<action
12	android:id="@+id/action_HomeFragment_to_detailFr agment"
13	app:destination="@id/detailFragment"
	/>
14	</fragment>
15	
16	<fragment
17	android:id="@+id/detailFragment"
18	android:name="com.example.londondestination.Frag mentGuweh"
19	android:label="DetailFragment" >
20	<argument
21	android:name="imageResId"
22	app:argType="integer" />
23	<argument
24	android:name="nama"
25	app:argType="string" />
26	<argument
27	android:name="deskripsi"
28	app:argType="string" />
29	</fragment>
30	
31	</navigation>

Tabel 3. 10. Source Code Jawaban Soal 1

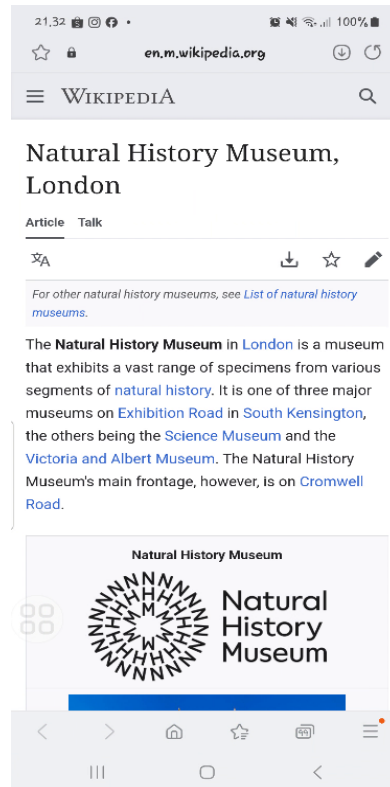
B. Output Program



Gambar 3. 3. Screenshot Hasil Jawaban Soal 1



Gambar 3. 4. Screenshot Hasil Jawaban Soal 1



Gambar 3. 5. Screenshot tombol Detail



Gambar 3. 6. Screenshot tombol Info

C. Pembahasan

1. MainActivity.kt

Di file ini ada `MainActivity` merupakan class utama dalam aplikasi Android yang digunakan untuk menampilkan antarmuka pengguna dari `activity_main.xml`. Dimana class ini mewarisi dari `AppCompatActivity` adalah versi aktivitas dengan dukungan fitur tambahan. Terdapat `onCreate()`, `ActivityMainBinding` digunakan untuk menghubungkan layout XML dengan kode Kotlin secara langsung melalui `View Binding`, sehingga memudahkan akses ke elemen UI tanpa menggunakan `findViewById`. Kemudian ada `setContentview(binding.root)` digunakan untuk menampilkan layout tersebut sebagai isi dari aktivitas yang digunakan.

2. FragmentGuweh.kt

Pada class `FragmentGuweh` berguna agar bisa menampilkan detail dari suatu destinasi di aplikasi. Dimana fragment ini memakai `View Binding (DetailFragmentBinding)` untuk mengakses elemen-elemen UI dari layout `detail_fragment.xml` tanpa perlu adanya `findViewById`. Dilihat bagian `onCreateView` yang mana layout di-*inflate* akan dihubungkan menuju fragment melalui binding. Juga, di `onViewCreated` nantinya data seperti gambar (`imageResId`), nama, dan deskripsi diambil dari argumen yang dikirim ke fragment dan ditampilkan ke elemen UI seperti `ImageView` dan `TextView`. Selain itu, ada fungsi `newInstance()` di bagian companion object berguna sebagai pembuatan instance fragment baru sekaligus mengisi argumen yang dibutuhkan. Terakhir, ada `onDestroyView` agar binding dihapus untuk mencegah kebocoran memori.

3. HomeFragment.kt

Pada class `HomeFragment` merupakan bagian dari aplikasi Android yang menampilkan daftar destinasi wisata di London dalam bentuk `RecyclerView`. Dimana fragment ini menggunakan `View Binding` (`HomeFragmentBinding`) agar bisa menghubungkan elemen layout dengan kode tanpa harus menggunakan `findViewById`. Kita lihat di `onCreateView` nantinya layout di-*inflate* dan dikembalikan menjadi tampilan utama. Dengan adanya `onViewCreated`, maka fragment dapat mengatur `RecyclerView` menggunakan `LinearLayoutManager` dan adapter (`MyAdapter`) berisi daftar objek `MyData`. Dimana setiap item berisi informasi lengkap seperti nama, deskripsi, gambar, dan link dari destinasi wisata. Saat kita mencoba mengklik salah satu item, maka data dari item tersebut akan terkirim melalui *Bundle* yang menuju fragment detail (`FragmentGuweh`) menggunakan `Navigation Component`. Terakhir, ada `onDestroyView` berguna untuk menghapus binding agar mencegah kebocoran memori. Jadi, fragment ini berfungsi sebagai tampilan utama yang menampilkan seluruh destinasi dalam bentuk daftar.

4. MyAdapter.kt

Pada class `MyAdapter` untuk bisa menampilkan daftar destinasi wisata dalam `RecyclerView`. Dimana adapter ini menerima dua parameter yaitu, daftar data (`destinations`) berupa objek `MyData`, dan sebuah fungsi lambda (`onDetailClick`) nantinya dijalankan saat tombol detail kita klik. Di dalamnya terdapat `ViewHolder` yang menggunakan `ItemListAdapter` untuk mengakses elemen-elemen dari layout item seperti nama, tahun, deskripsi singkat, gambar, dan dua tombol. Kemudian, di dalam `onBindViewHolder` setiap data diatur menuju tampilan sesuai posisinya, termasuk menampilkan teks dan gambar. Nah, ketika kita klik nantinya membuka halaman web sesuai URL destinasi melalui `Intent`. Sedangkan bagian tombol detail akan menjalankan fungsi

`onDetailClick` dan mengirim data yang dipilih ke fragment detail. Dengan adanya adapter bisa menjembatani data dan tampilan dalam daftar secara efisien, sekaligus menangani interaksi kita sebagai pengguna lebih baik.

5. **MyData.kt**

Pada data class bernama `MyData` yang berfungsi sebagai model data utama untuk aplikasi destinasi wisata London. Dimana class ini menggunakan anotasi `@Parcelize` yang memberi kemungkinan objek `MyData` bisa dikirim antar komponen Android seperti antar fragment atau activity dengan lebih mudah melalui `Bundle`. Dengan adanya class ini menyimpan enam properti penting untuk tiap destinasi seperti, nama (nama tempat wisata), `description` (deskripsi lengkap), `descriptionSingkat` (deskripsi singkat), `year` (tahun pendirian atau peresmian), `image` (ID dari gambar sumber daya), dan `link` (tautan ke halaman informasi lebih lanjut). Pentingnya struktur ini agar aplikasi bisa menampilkan informasi yang lengkap dan interaktif mengenai setiap lokasi wisata dalam bentuk daftar, serta meneruskan data dengan rapi ke tampilan detail saat kita memilih salah satu destinasi. Jadi, file ini menjadi pondasi data yang akan digunakan oleh adapter dan fragment lainnya di aplikasi ini.

6. **Detail_fragment.xml**

Pada bagian ini digunakan untuk menampilkan tampilan detail dari sebuah tempat wisata dalam aplikasi. Dimana seluruh konten dibungkus di dalam sebuah elemen `ScrollView` yang memungkinkan kita bisa menggulir layar ke bawah jika isi kontennya lebih panjang dari ukuran layar. Hal ini penting agar seluruh informasi tetap bisa diakses meskipun banyak atau Panjang isinya.

Di dalam `ScrollView` terdapat sebuah `CardView` yang berfungsi memberikan tampilan lebih menarik karena memiliki sudut melengkung (dengan `cardCornerRadius="16dp"`) dan

(`cardElevation="8dp"`), sehingga konten tampak seperti kartu sedikit terangkat dari latar belakang membuat tampilan lebih rapi dan enak dilihat.

Isi dari `CardView` diletakkan di dalam `LinearLayout` yang diatur secara vertikal. Di dalam layout ini terdapat tiga komponen utama yaitu, pertama ada `ImageView` menggunakan ID `detailImage` berguna untuk menampilkan gambar tempat wisata yang biasanya gambar ini nantinya ditampilkan dari file `drawable` atau dari sumber lain. Dengan gambar ini disetel menggunakan `scaleType="centerCrop"` agar mengisi seluruh ruang yang disediakan dengan proporsional.

Selanjutnya, dua `TextView` yang pertama ada `detailTitle` digunakan untuk menampilkan nama tempat dengan ukuran teks yang cukup besar (`20sp`) dan gaya teks tebal (**bold**), serta diberi `padding` agar teks tidak menempel langsung ke sisi layar. Yang kedua, `detailDescription` berfungsi untuk menampilkan deskripsi lengkap dari tempat tersebut dengan deskripsi ini menggunakan ukuran teks sedikit lebih kecil (`16sp`) dan diberi `padding` sisi kiri, kanan, dan bawah agar memudahkan saat membaca. Jadi, secara keseluruhan, layout ini dibuat untuk memberikan tampilan detail tempat wisata secara bersih, informatif, dan nyaman dilihat.

7. `activity_main.xml`

Pada bagian ini berguna untuk mengelola navigasi antar-fragment. Dimana seluruh tampilan dibungkus di dalam `ConstraintLayout` merupakan salah satu jenis layout fleksibel yang memungkinkan pengaturan posisi antar elemen secara dinamis dan efisien. Namun dalam kasus ini, hanya ada satu elemen di dalamnya, jadi `ConstraintLayout` tidak benar-benar dimanfaatkan secara penuh. Dengan elemen utamanya itu `FragmentContainerView` berfungsi sebagai wadah (container) agar bisa menampilkan fragment. Adanya `FragmentContainerView` ini memiliki ID `fragment_container_view` dan diatur untuk mengisi seluruh lebar dan tinggi layar (`match_parent`). Terdapat property berisi

`android:name` yang menunjuk ke bagian `androidx.navigation.fragment.NavHostFragment` berarti view ini berperan sebagai host fragment nantinya saat mengelola navigasi.

Adanya atribut `app:navGraph` menunjuk ke file `nav_graph` di direktori `res/navigation` berisi struktur navigasi fragment—seperti daftar tujuan (`destination`) dan hubungan antar fragment (misalnya aksi berpindah antar fragment). Sementara `app:defaultNavHost="true"` digunakan dalam menyatakan bahwa fragment ini adalah host default agar bisa menangani navigasi sistem, seperti tombol "Back" di Android. Jadi, keseluruhan bagian ini XML ini menyusun fondasi navigasi fragment berbasis Jetpack Navigation dengan melakukan penempatan satu `NavHostFragment` untuk mengatur transisi antar-fragment di dalam aplikasi tanpa perlu berpindah antar activity yang menjadikan navigasi lebih efisien dan lebih terstruktur.

8. `home_fragment.xml`

Pada bagian ini adalah layout untuk sebuah Fragment lebih tepatnya `HomeFragment` yang menggunakan `ConstraintLayout` sebagai layout utama. Dimana layout ini hanya memiliki satu elemen di dalamnya, yaitu sebuah `RecyclerView` dengan ID `rv_character` dengan fungsi utama dari `RecyclerView` ini untuk menampilkan daftar item secara efisien dan dapat discroll tergantung pada pengaturan adapter dan layout manager-nya.

Pada pengaturannya `RecyclerView` ini dibuat agar memenuhi seluruh ruang layar karena semua constraint-nya dihubungkan ke tepi-tepi parent layout. Hal ini terlihat dari penggunaan `0dp` untuk `layout_width` dan `layout_height` berarti ukuran akan disesuaikan berdasarkan constraint yang diberikan. Dilihat bagian Constraint-nya menghubungkan atas (Top), bawah (Bottom), kiri (Start), dan kanan (End) ke parent,

sehingga `RecyclerView` menutupi seluruh tampilan yang tersedia dalam fragment.

Sementara itu, kita lihat di atribut `tools:context=".HomeFragment"` berguna hanya untuk keperluan preview di Android Studio agar tampilan ini bisa dikenali sedang digunakan `HomeFragment` saat mendesain antarmuka. Jadi, secara keseluruhan layout bagian ini sangat sederhana tapi efektif demi bisa menampilkan daftar yang dinamis, seperti daftar tempat wisata, karakter, atau data lain nantinya diisi melalui adapter dalam kode program aplikasi ini.

9. `item_list.xml`

Pada layout file untuk komponen tampilan dalam aplikasi Android yang mana layout ini menggunakan `CardView` sebagai wadah utama yang memberikan efek tampilan seperti kartu, lengkap dengan sudut melengkung dan bayangan. Di dalam `CardView` terdapat `ConstraintLayout` digunakan sebagai layout utama agar bisa menyusun elemen-elemen UI dengan fleksibilitas tinggi sesuai posisi satu sama lain. Dimana bagian pertama itu `ImageView` berguna untuk menampilkan gambar, dengan ukurannya ditetapkan 100dp x 150dp dan letaknya di pojok kiri atas tampilan. Gambar diatur agar "crop" ke tengah (`centerCrop`) agar mengisi seluruh ruang.

Selanjutnya, ada tiga `TextView` yang masing-masing menampilkan nama (`textViewName`), tahun atau angkatan (`textViewYear`), dan deskripsi tambahan (`textViewDesc`). Dari semua `TextView` ini diletakkan di sebelah kanan `ImageView` yang disusun secara vertikal satu per satu dari atas ke bawah. Adanya `textViewName` yang menggunakan teks tebal dan ukuran huruf lebih besar cocok untuk judul atau nama utama. Sedangkan `textViewYear` dan `textViewDesc` memiliki ukuran teks yang lebih kecil dan warna abu-abu gelap (`#555555`) untuk memberikan perbedaan teks.

Di bagian paling bawah layout terdapat `LinearLayout` yang berisi dua tombol (`Button`), yaitu tombol "Detail" dan "Info". Dengan layout tombol ini diposisikan di bawah `ImageView` dan diberi margin atas agar tidak menempel langsung. Dimana warna latar tombol diambil dari `@color/purple_200` yang memberi warna ungu dari resources aplikasi. Kedua tombol ini disiapkan untuk tindakan lebih lanjut, misalnya "Detail" untuk membuka tampilan biodata lengkap, dan "Info" untuk menampilkan informasi tambahan.

Jadi, secara keseluruhan, XML ini digunakan untuk item dalam `RecyclerView` karena desainnya ringkas, informatif, dan responsif. Dengan struktur layout sudah rapi sesuai pengaturan `ConstraintLayout` yang fleksibel dan `CardView` mempercantik tampilan menjadi tipe layout biasanya digunakan agar bisa menampilkan daftar data dalam bentuk kartu.

10. nav_graph.xml

Pada file `navigation graph` fungsinya untuk mendefinisikan alur navigasi antar tampilan atau fragment dalam sebuah aplikasi. Di bagian dalamnya akan dideklarasikan dua buah fragment yaitu, `HomeFragment` dan `detailFragment`. Dimana layout navigasi ini untuk aplikasi dapat berpindah dari satu fragment menuju fragment lainnya secara terstruktur dan mudah dikelola.

Pertama, elemen `<navigation>` menyatakan bahwa ini bagian root dari `navigation graph` yang terdapat atribut ini `app:startDestination="@id/HomeFragment"` yang menandakan bahwa saat aplikasi dijalankan, maka tampilan awal (halaman pertama) nantinya ditampilkan itu `HomeFragment`. Dengan fragment ini diberi ID `@+id/HomeFragment` dan terhubung menuju bagian dari `class com.example.londondestination.HomeFragment`.

Di dalam HomeFragment terdapat elemen `<action>` dengan ID `action_HomeFragment_to_detailFragment` merupakan aksi navigasi yang berarti jika dipicu (misalnya kita klik tombol), maka aplikasi akan berpindah dari HomeFragment menuju detailFragment. Nama action ini bisa dipanggil di kode Kotlin atau Java untuk mentrigger perpindahan halaman ke selanjutnya.

Adanya fragment tujuan yaitu detailFragment dikaitkan dengan class Java atau Kotlin `com.example.londondestination.FragmentGuweh`. Dimana fragment ini nantinya menerima data melalui tiga buah argument seperti, `imageResId` (tipe integer, biasanya ID gambar dari resource drawable), nama (tipe string), dan deskripsi (juga string) dari ketiga argumen ini memberi kemungkinan fragment tujuan bisa menerima data yang dikirim dari HomeFragment, misalnya saat kita klik tombol "Detail". Jadi, file ini memastikan bahwa aplikasi punya alur navigasi yang rapi dari halaman utama ke halaman detail, dan sudah siap menerima data agar bisa ditampilkan.

SOAL 2

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

A. Pembahasan

Menurut saya RecyclerView masih banyak digunakan karena sebagian besar aplikasi Android lama yang masih pakai View XML, bukan Jetpack Compose. Nah, pada LazyColumn itu cuma bisa dipakai semisal kita udah pakai Compose. Jadi, kalau aplikasi yang dibuat itu masih pakai layout XML biasa, pilihan paling bagus dan umum untuk menampilkan daftar data tetap RecyclerView. Dengan konteks memang benar bahwa RecyclerView itu kodenya lebih panjang karena harus bikin Adapter, ViewHolder, dan file layout item-nya, dll. Tetapi, kelebihanannya itu sangat fleksibel dengan bisa menampilkan data yang banyak tanpa bikin aplikasi jadi lemot, dan bisa dikustomisasi banget, misalnya saat kita pakai tombol, gambar, atau animasi di tiap item. Jadi, meskipun lebih ribet karena RecyclerView masih dipakai namun memiliki beberapa kelebihan, sebagai berikut:

- Cocok untuk View XML,
- Sudah stabil dan teruji,
- Banyak tutorial dan contoh yang bisa kita ikuti.

Nah, selama belum pakai Jetpack Compose, maka RecyclerView tetap pilihan yang paling tepat buat menampilkan list data.

MODUL 4 : VIEWMODEL AND DEBUGGING

SOAL 1

Soal Praktikum:

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

1. MainActivity.kt

1	package com.example.londondestination
2	
3	import android.os.Bundle
4	import androidx.appcompat.app.AppCompatActivity
5	import
	com.example.londondestination.databinding.Activi

	tyMainBinding
6	
7	class MainActivity : AppCompatActivity() {
8	private lateinit var binding:
	ActivityMainBinding
9	override fun onCreate(savedInstanceState:
	Bundle?) {
10	
11	super.onCreate(savedInstanceState)
12	binding =
	ActivityMainBinding.inflate(layoutInflater)
13	setContentView(binding.root)
14	}
15	}

Tabel 4. 1. Source Code MainActivity

2. FragmentGuweh.kt

1	package com.example.londondestination
2	
3	import android.os.Bundle
4	import android.view.LayoutInflater
5	import android.view.View
6	import android.view.ViewGroup
7	import androidx.fragment.app.Fragment
8	import
	com.example.londondestination.databinding.Detail
	FragmentBinding
9	
10	
11	class FragmentGuweh : Fragment() {
12	
13	private var _binding: DetailFragmentBinding?

	= null
14	private val binding get() = _binding!!
15	
16	override fun onCreateView(
17	inflater: LayoutInflater, container:
	ViewGroup?,
18	savedInstanceState: Bundle?
): View {
19	_binding =
	DetailFragmentBinding.inflate(inflater,
	container, false)
20	return binding.root
21	}
22	
23	override fun onViewCreated(view: View,
	savedInstanceState: Bundle?) {
24	super.onViewCreated(view,
	savedInstanceState)
25	
26	val imageResId =
	arguments?.getInt("imageResId") ?:
	R.drawable.ic_launcher_background
27	val nama =
	arguments?.getString("nama") ?: "Nama tidak
	tersedia"
28	val deskripsi =
	arguments?.getString("deskripsi") ?: "Deskripsi
	tidak tersedia"
29	
30	
31	binding.detailImage.setImageResource(imageResId)
32	binding.detailTitle.text = nama

33	binding.detailDescription.text =
	deskripsi
34	}
35	
36	override fun onDestroyView() {
37	super.onDestroyView()
38	_binding = null
39	}
40	companion object {
41	fun newInstance(imageResId: Int, nama:
	String, deskripsi: String): FragmentGuweh {
42	val fragment = FragmentGuweh()
43	val args = Bundle()
44	args.putInt("imageResId",
	imageResId)
45	args.putString("nama", nama)
46	args.putString("deskripsi",
	deskripsi)
47	fragment.arguments = args
48	return fragment
49	}
50	}
51	}

Tabel 4. 2. Source Code FragmentGuweh

3. HomeFragment.kt

1	package com.example.londondestination
2	
3	import android.os.Bundle
4	import android.util.Log
5	import android.view.LayoutInflater
6	import android.view.View

7	import android.view.ViewGroup
8	import androidx.fragment.app.Fragment
9	import androidx.fragment.app.viewModels
10	import androidx.lifecycle. <i>lifecycleScope</i>
11	import
	androidx.navigation.fragment.findNavController
12	import androidx.core.os.bundleOf
13	import
	androidx.recyclerview.widget.LinearLayoutManager
14	import
	com.example.londondestination.databinding.HomeFr agmentBinding
15	import kotlinx.coroutines.flow.collectLatest
16	import kotlinx.coroutines.launch
17	
18	class HomeFragment : Fragment() {
19	
20	private var _binding: HomeFragmentBinding? = null
21	private val binding get() = _binding!!
22	
23	private lateinit var adapter: MyAdapter
24	
25	private val viewModel: HomeViewModel by viewModels {
26	HomeViewModelFactory()
27	}
28	
29	override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View {
30	_binding =

	HomeFragmentBinding.inflate(inflater, container, false)
31	return binding.root
32	}
33	
34	override fun onCreateView(view: View, savedInstanceState: Bundle?) {
35	super.onCreateView(view, savedInstanceState)
36	
37	adapter = MyAdapter(emptyList())
	{ selectedItem ->
38	viewModel.onItemClicked(selectedItem)
39	}
40	
41	binding.rvCharacter.layoutManager =
	LinearLayoutManager(requireContext())
42	binding.rvCharacter.adapter = adapter
43	
44	viewLifecycleOwner.lifecycleScope.launch
	{
45	viewModel.destinationList.collectLatest { list -
	>
46	adapter = MyAdapter(list)
	{ selectedItem ->
47	viewModel.onItemClicked(selectedItem)
48	}
49	binding.rvCharacter.adapter =
	adapter
50	}
51	}
52	

53	<code>viewLifecycleOwner.lifecycleScope.launch</code>
	<code>{</code>
54	<code>viewModel.selectedItem.collectLatest</code>
	<code>{ item -></code>
55	<code>item?.let {</code>
56	<code>Log.d("HomeFragment",</code>
	<code>"Navigasi ke DetailFragment untuk: \${it.nama},</code>
	<code>Tahun: \${it.year}, Deskripsi:</code>
	<code>\${it.description}")</code>
57	
58	<code>val bundle = bundleOf(</code>
59	<code>"imageResId" to</code>
	<code>it.image,</code>
60	<code>"nama" to it.nama,</code>
61	<code>"deskripsi" to</code>
	<code>it.description</code>
62	<code>)</code>
63	<code>findNavController().navigate(R.id.action_HomeFra</code>
	<code>gment_to_detailFragment, bundle)</code>
64	<code>viewModel.resetSelectedItem()</code>
65	<code>}</code>
66	<code>}</code>
67	<code>}</code>
68	
69	<code>}</code>
70	
71	<code>override fun onDestroyView() {</code>
72	<code>super.onDestroyView()</code>
73	<code>_binding = null</code>
74	<code>}</code>
75	<code>}</code>

Tabel 4. 3. Source Code HomeFragment

3. HomeViewModel.kt

```
1 package com.example.londondestination
2
3 import android.util.Log
4 import androidx.lifecycle.ViewModel
5 import kotlinx.coroutines.flow.MutableStateFlow
6 import kotlinx.coroutines.flow.StateFlow
7 import kotlinx.coroutines.flow.asStateFlow
8
9 class HomeViewModel : ViewModel() {
10
11     private val _destinationList =
12         MutableStateFlow<List<MyData>>(emptyList())
13     val destinationList: StateFlow<List<MyData>>
14         = _destinationList.asStateFlow()
15
16     private val _selectedItem =
17         MutableStateFlow<MyData?>(null)
18     val selectedItem: StateFlow<MyData?> =
19         _selectedItem.asStateFlow()
20
21     init {
22         val data = listOf(
23             MyData(
24                 nama = "Natural History Museum",
25                 description = "Natural History
26 Museum di London adalah destinasi wajib bagi
27 pecinta sains, sejarah alam, dan keluarga dengan
28 anak-anak. Museum ini memiliki lebih dari 80
29 juta spesimen, mencakup zoologi, paleontologi,
30 botani, dan geologi. Daya tarik utama termasuk
31 rangka dinosaurus, seperti Diplodocus dan model
```

22	<p>animatronik Tyrannosaurus Rex. Selain itu, pengunjung bisa melihat koleksi batuan langka, meteorit, serta kristal dan permata.\n" +</p> <p>"\n" + "Di Earth Galleries, ada pameran interaktif tentang geologi bumi, gunung berapi, dan gempa bumi. Museum ini juga memiliki zona edukasi yang menarik untuk anak-anak. Dibangun dengan arsitektur Romanesque bergaya Victoria, bangunannya memukau dengan interior yang megah. Yang menarik, museum ini gratis untuk dikunjungi, memberikan pengalaman yang tak terlupakan bagi semua pengunjung.",</p>
23	<p> descriptionsingkat = "Museum sejarah tentang alam",</p>
24	<p> year = 1881,</p>
25	<p> image =</p> <p>R.drawable.naturalhistorymuseum,</p>
26	<p> link =</p> <p>"https://en.wikipedia.org/wiki/Natural_History_Museum,_London"</p>
27	<p>),</p>
28	<p> MyData(</p>
29	<p> nama = "London Eye",</p>
30	<p> description = "London Eye adalah kincir raksasa setinggi 135 meter di tepi Sungai Thames, dan menjadi salah satu ikon paling terkenal di London. Dari dalam kapsul kacanya, pengunjung bisa menikmati panorama kota yang menakjubkan, termasuk pemandangan Big Ben, Gedung Parlemen, dan Sungai Thames. Pada hari cerah, jarak pandang bisa mencapai hampir 40</p>

31	<pre> kilometer.\n" + "\n" + "Waktu terbaik untuk naik adalah saat senja, ketika cahaya kota mulai menyala dan langit berwarna keemasan karena menciptakan suasana romantis dan tenang. Setiap putaran berlangsung sekitar 30 menit, memberikan cukup waktu untuk mengagumi pemandangan, mengambil foto, atau sekadar menikmati momen dari ketinggian. London Eye bukan hanya atraksi wisata, tetapi juga simbol kebanggaan kota yang terus memikat baik turis maupun warga lokal.", </pre>
32	<pre> descriptionsingkat = "Komedi putar raksasa London", </pre>
33	<pre> year = 2000, </pre>
34	<pre> image = R.drawable.londoneye, </pre>
35	<pre> link = "https://en.wikipedia.org/wiki/London_Eye" </pre>
36	<pre>), </pre>
37	<pre> MyData(</pre>
38	<pre> nama = "British Museum", </pre>
39	<pre> description = "British Museum adalah destinasi luar biasa bagi pecinta sejarah dan arkeologi, dengan koleksi global yang mencakup ribuan tahun peradaban manusia. Begitu masuk, suasana elegan langsung membawa pengunjung seakan menjelajahi masa lalu dari mumi Mesir, patung Yunani, artefak Mesopotamia, hingga keramik Tiongkok dan seni Islam. Batu Rosetta adalah salah satu sorotan utama, bersama Patung Ramses II dan reruntuhan Kuil Parthenon yang ikonik.\n" + </pre>

40	"\n" + "Setiap koleksi disertai penjelasan informatif yang memudahkan pengunjung memahami konteks sejarahnya. Great Court, dengan atap kaca yang terang dan desain modern, menjadi pusat bangunan yang menawan dan nyaman untuk bersantai. British Museum bukan sekadar tempat melihat artefak, tapi juga ruang kontemplatif yang menghubungkan kita dengan jejak panjang umat manusia dan semuanya bisa dinikmati tanpa biaya masuk.",
41	description singkat = "Museum koleksi dunia",
42	year = 1753,
43	image = R.drawable.british_museum,
44	link = "https://en.wikipedia.org/wiki/British_Museum"
45),
46	MyData(nama = "Big Ben",
47	description = "Big Ben adalah ikon tak tergantikan London, berdiri megah di samping Gedung Parlemen di tepi Sungai Thames. Meski banyak mengira namanya merujuk pada menaranya, Big Ben sebenarnya adalah lonceng besar seberat lebih dari 13 ton di dalam Elizabeth Tower adalah nama resmi menara tersebut, yang diberikan untuk menghormati Ratu Elizabeth II.\n" +
49	"\n" + "Dentang Big Ben punya makna emosional yang dalam, sering terdengar dalam momen penting seperti pergantian

	tahun atau peringatan nasional, dan bahkan disiarkan BBC sejak 1920-an. Arsitekturnya yang anggun menjadi latar favorit para wisatawan, baik saat disinari mentari pagi maupun diterangi lampu malam hari. Walau akses ke dalam menara terbatas, cukup berdiri di dekatnya sudah membuat pengunjung merasa terhubung dengan sejarah dan semangat kota London yang tak lekang waktu",
50	<pre> descriptionsingkat = "Jam besar London", </pre>
51	<pre> year = 1859, </pre>
52	<pre> image = R.drawable.bigben, </pre>
53	<pre> link = "https://en.wikipedia.org/wiki/Big_Ben" </pre>
54	<pre>), </pre>
55	<pre> MyData(</pre>
56	<pre> nama = "Buckingham Palace", </pre>
57	<pre> description = "Buckingham Palace adalah simbol monarki Inggris yang berdiri megah di pusat London, berfungsi sebagai kediaman resmi Raja dan pusat berbagai acara kenegaraan. Dengan lebih dari 700 ruangan, istana ini mencerminkan kemewahan dan sejarah yang hidup, bahkan dari luar pagar hitamnya yang ikonik.\n" + </pre>
58	<pre> "\n" + "Salah satu atraksi utama adalah Upacara Pergantian Penjaga, prosesi tradisional dengan seragam merah dan musik marching band yang menarik ribuan wisatawan setiap harinya. Jika bendera kerajaan berkibar di atas istana, itu menandakan Raja </pre>

	sedang berada di dalam momen sederhana yang membuat banyak orang merasa lebih dekat dengan sejarah kerajaan.\n" +
59	<pre> "\n" + "Pada musim panas, beberapa ruang kenegaraan dibuka untuk umum, menampilkan interior menawan lengkap dengan kristal, lukisan klasik, dan kemegahan khas kerajaan. Meski banyak pengunjung hanya melihat dari luar, pesona dan wibawa istana ini menjadikannya salah satu destinasi paling ikonik di London.", </pre>
60	<pre> descriptionsingkat = "Istana resmi Kerajaan Inggris", </pre>
61	<pre> year = 1703, </pre>
62	<pre> image = R.drawable.buckinghampalace, </pre>
63	<pre> link = "https://en.wikipedia.org/wiki/Buckingham_Palace " </pre>
64	<pre>), </pre>
65	<pre> MyData(</pre>
66	<pre> nama = "Tower of London", </pre>
67	<pre> description = "Tower of London adalah benteng bersejarah di tepi Sungai Thames yang menyimpan kisah dramatis tentang kekuasaan, pengkhianatan, dan warisan kerajaan Inggris. Dulu berfungsi sebagai penjara bagi bangsawan, termasuk Anne Boleyn yang dieksekusi di sana, tempat ini memancarkan nuansa mencekam sekaligus megah, terutama di lokasi-lokasi penting seperti halaman eksekusi.\n" + </pre>
68	<pre> "\n" + "Namun, sisi </pre>

	gelap itu berpadu dengan kemewahan karena di sinilah Permata Mahkota Inggris disimpan, termasuk mahkota dan tongkat kerajaan yang berkilau menakjubkan. Kontras antara sejarah kelam dan simbol kejayaan membuat kunjungan ke Tower of London terasa seperti menyusuri lorong waktu, menghadirkan pengalaman mendalam yang tak terlupakan.",
69	<pre> descriptionsingkat = "Benteng mempunyai banyak sejarah", </pre>
70	<pre> year = 1066, </pre>
71	<pre> image = R.drawable.toweroflondon, </pre>
72	<pre> link = "https://en.wikipedia.org/wiki/Tower_of_London" </pre>
73	<pre>), </pre>
74	<pre> MyData(</pre>
75	<pre> nama = "Warner Bros. Studio Tour London", </pre>
76	<pre> description = "Warner Bros. Studio Tour London adalah destinasi impian bagi penggemar Harry Potter, menawarkan pengalaman imersif ke dunia sihir yang sebelumnya hanya bisa dilihat di layar. Begitu masuk, kamu akan dibawa ke set asli seperti Great Hall, Diagon Alley, dan Privet Drive dimana semuanya penuh detail yang membuatmu merasa benar-benar berada di dunia Hogwarts.\n" + </pre>
77	<pre> "\n" + "Selain menjelajahi lokasi ikonik, pengunjung juga bisa melihat properti film seperti Horcrux, kostum rumah-rumah Hogwarts, hingga proses pembuatan </pre>

	<p>efek visual yang menghidupkan sihir di layar. Setiap sudut studio menyuguhkan keajaiban yang membuat kamu makin menghargai imajinasi dan kerja keras di balik film. Dan tentu saja, mencicipi Butterbeer jadi penutup manis dari kunjungan yang terasa seperti pulang ke dunia masa kecil yang penuh keajaiban.",</p>
78	<pre> descriptionsingkat = "Studio Harry Potter London", </pre>
79	<pre> year = 2012, </pre>
80	<pre> image = R.drawable.harrypotterstudio, </pre>
81	<pre> link = "https://www.wbstudiotour.co.uk/" </pre>
82	<pre>), </pre>
83	<pre> MyData(</pre>
84	<pre> nama = "Hyde Park", </pre>
85	<pre> description = "Hyde Park adalah oase hijau di tengah London yang menawarkan ketenangan dan ruang bebas bagi siapa saja, dari warga lokal hingga turis. Dengan luas lebih dari 140 hektar, taman ini menjadi tempat ideal untuk jogging, bersepeda, membaca buku, atau sekadar duduk santai di tepi danau Serpentine.\n" + </pre>
86	<pre> "\n" + "Suasananya santai dan cocok buat piknik, bermain, atau menikmati kopi di bawah rindangnya pepohonan. Salah satu sudut paling unik adalah Speaker's Corner, simbol kebebasan berpendapat di mana siapa pun bisa berbicara di depan umum. Selain sebagai tempat pelarian dari hiruk pikuk kota, Hyde Park juga kerap menjadi lokasi konser besar </pre>

	dan festival, menjadikannya ruang publik yang dinamis dan menyatu dengan jiwa kota London.",
87	description singkat = "Taman pusat kota",
88	year = 1637,
89	image = R.drawable.hydepark,
90	link = "https://en.wikipedia.org/wiki/Hyde_Park,_London"
91),
92	MyData(
93	nama = "The Sherlock Holmes Museum",
94	description = "The Sherlock Holmes Museum di 221B Baker Street adalah surga bagi penggemar detektif legendaris ini, membawa pengunjung langsung ke dunia fiksi era Victoria yang terasa hidup. Interiornya merekonstruksi ruang kerja dan rumah Holmes secara detail-lengkap dengan perapian, kaca pembesar, dan barang-barang pribadi khas karakter dalam cerita.\n" +
95	"\n" + "Setiap sudut museum dirancang agar kamu seolah benar-benar berada di tengah kisah misteri bersama Holmes dan Watson. Lebih dari sekadar pameran, museum ini menawarkan pengalaman yang memuaskan rasa ingin tahu para penggemar dan pencinta cerita klasik.",
96	description singkat = "Museum seorang detektif yang ikonik",
97	year = 1990,

98	image =
	R.drawable.thesherlockholmesmuseum,
99	link = "https://www.sherlock-
	holmes.co.uk/"
100),
101	MyData(
102	nama = "St Paul's Cathedral",
103	description = "St Paul's
	Cathedral adalah salah satu ikon arsitektur
	paling menakjubkan di London, dikenal dengan
	kubah raksasanya yang mendominasi cakrawala
	kota. Dari luar terlihat megah, tapi keindahan
	sejatinya baru benar-benar terasa saat kamu
	melangkah masuk—ruang dalamnya hening, agung,
	dan sarat nuansa spiritual. Langit-langit
	tinggi, kaca patri indah, dan cahaya alami
	menciptakan atmosfer yang membuat siapa pun
	terdiam dalam kekaguman.\n" +
104	"\n" + "Salah satu
	pengalaman paling tak terlupakan di sini adalah
	menaiki ratusan anak tangga menuju puncak kubah.
	Dari atas, kamu bisa menikmati panorama kota
	London yang luas dan penuh sejarah, dari Sungai
	Thames hingga gedung-gedung modern yang berdiri
	berdampingan dengan bangunan bersejarah. Tak
	heran, St Paul's sering menjadi lokasi berbagai
	momen penting nasional, karena tempat ini bukan
	hanya gereja, tapi simbol kekuatan dan keindahan
	yang hidup modern di kota.",
105	descriptionsingkat = "Katedral
	bersejarah di London",
106	year = 1710,

107	image =
	R.drawable.stpaulcathedral,
108	link =
	"https://en.wikipedia.org/wiki/St_Paul%27s_Cathe dral"
109),
110	MyData(
111	nama = "Camden Market",
112	description = "Camden Market adalah salah satu tempat paling unik dan penuh karakter di London, ideal bagi kamu yang suka berburu barang-barang tak biasa dan merasakan suasana kota yang dinamis. Pasar ini dipenuhi toko-toko kecil yang menjual pakaian vintage, aksesori handmade, dan berbagai barang nyentrik yang sering kali hanya ada satu di dunia. Suasananya ramai tapi seru, dengan musik jalanan, aroma makanan internasional, dan gaya busana pengunjung yang beragam.\n" +
113	"\n" + "Selain jadi pusat belanja, Camden juga merupakan ruang ekspresi subkultur alternatif seperti punk, goth, dan hippie—tempat di mana semua orang bisa tampil sesuai dirinya sendiri. Kalau lapar, pilihan kulinernya luar biasa banyak dan menggoda, dari makanan Asia, Latin, Timur Tengah, sampai fusion kreatif. Setiap kunjungan ke Camden terasa seperti eksplorasi baru bukan sekadar belanja, tapi pengalaman hidup kota London yang bebas, kreatif, dan penuh kejutan.",
114	descriptionsingkat = "Pasar terbesar di London",

115	year = 1974,
116	image = R.drawable.camdenmarket,
117	link =
	"https://en.wikipedia.org/wiki/Camden_Market"
118)
119	
120)
121	_destinationList.value = data
122	Log.d("HomeViewModel", "List data berhasil dimuat sebanyak \${data.size} item")
123	}
124	
125	fun onItemClick(item: MyData) {
126	_selectedItem.value = item
127	}
128	
129	fun resetSelectedItem() {
130	_selectedItem.value = null
131	}
132	}

Tabel 4. 4. Source Code HomeViewModel

5. HomeViewModelFactory.kt

1	package com.example.londondestination
2	
3	import androidx.lifecycle.ViewModel
4	import androidx.lifecycle.ViewModelProvider
5	
6	class HomeViewModelFactory :
	ViewModelProvider.Factory {
7	override fun <T : ViewModel>
	create(modelClass: Class<T>): T {

8	if
	(modelClass.isAssignableFrom(HomeViewModel::class.java)) {
9	return HomeViewModel() as T
10	}
11	throw IllegalArgumentException("Unknown ViewModel class")
12	}
13	}

Tabel 4. 5. Source Code HomeViewModelFactory

6. MyAdapter.kt

1	package com.example.londondestination
2	
3	import android.content.Intent
4	import android.net.Uri
5	import android.util.Log
6	import android.view.LayoutInflater
7	import android.view.ViewGroup
8	import androidx.recyclerview.widget.RecyclerView
9	import
	com.example.londondestination.databinding.ItemListBinding
10	
11	class MyAdapter(
12	private val destinations: List<MyData>,
13	private val onDetailClick: (MyData) -> Unit
14) : RecyclerView.Adapter<MyAdapter.ViewHolder>()
	{
15	
16	inner class ViewHolder(val binding:
	ItemListBinding) :

17	RecyclerView.ViewHolder(binding.root)
18	
19	override fun onCreateViewHolder(parent:
	ViewGroup, viewType: Int): ViewHolder {
20	val binding = ItemListBinding.inflate(
21	LayoutInflater.from(parent.context),
22	parent,
23	false
24)
25	return ViewHolder(binding)
26	}
27	
28	override fun onBindViewHolder(holder:
	ViewHolder, position: Int) {
29	val destination = destinations[position]
30	with(holder.binding) {
31	textViewName.text = destination.nama
32	textViewYear.text =
	destination.year.toString()
33	textViewDesc.text =
	destination.descriptionsingkat
34	imageView.setImageResource(destination.image)
35	
36	buttonLink.setOnClickListener {
37	val context = it .context
38	val intent =
	Intent(Intent.ACTION_VIEW,
	Uri.parse(destination.link))
39	context.startActivity(intent)
40	Log.d("MyAdapter", "Tombol Link
	ditekan untuk: \${destination.nama}")
41	}

42	
43	buttonDetail.setOnClickListener {
44	onDetailClick(destination)
45	Log.d("MyAdapter", "Tombol
	Detail ditekan untuk: \${destination.nama}")
46	}
47	}
48	}
49	
50	override fun getItemCount(): Int =
	destinations.size
51	}

Tabel 4. 6. Source Code MyAdapter

7. MyData.kt

1	package com.example.londondestination
2	
3	import android.os.Parcelable
4	import kotlinx.parcelize.Parcelize
5	
6	@Parcelize
7	data class MyData(
8	val nama: String,
9	val description: String,
10	val descriptionsingkat: String,
11	val year: Int,
12	val image: Int,
13	val link: String
14): Parcelable

Tabel 4. 7. Source Code MyData

8. Detail_fragment.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<ScrollView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	android:id="@+id/detailScrollView"
5	android:layout_width="match_parent"
6	android:layout_height="match_parent"
7	android:padding="16dp">
8	
9	<androidx.cardview.widget.CardView
10	android:layout_width="match_parent"
11	android:layout_height="wrap_content"
12	app:cardCornerRadius="16dp"
13	app:cardElevation="8dp">
14	
15	<LinearLayout
16	android:layout_width="match_parent"
17	android:layout_height="wrap_content"
18	android:orientation="vertical">
19	
20	<ImageView
21	android:id="@+id/detailImage"
22	android:layout_width="match_parent"
23	android:layout_height="200dp"
24	android:scaleType="centerCrop"/>
25	
26	<TextView
27	android:id="@+id/detailTitle"
28	android:layout_width="match_parent"

29	android:layout_height="wrap_content"
30	android:text="Nama Tempat"
31	android:textStyle="bold"
32	android:textSize="20sp"
33	android:padding="16dp" />
34	
35	<TextView
36	android:id="@+id/detailDescription"
37	android:layout_width="match_parent"
38	android:layout_height="wrap_content"
39	android:text="Deskripsi lengkap
	tempat destinasi."
40	android:paddingStart="16dp"
41	android:paddingEnd="16dp"
42	android:paddingBottom="16dp"
43	android:textSize="16sp" />
44	
45	</LinearLayout>
46	</androidx.cardview.widget.CardView>
47	</ScrollView>

Tabel 4. 8. Source Code detail_fragment

9. activity_main.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
3	xmlns:android="http://schemas.android.com/apk/res/android"
4	xmlns:app="http://schemas.android.com/apk/res-auto"
5	xmlns:tools="http://schemas.android.com/tools"
6	android:id="@+id/main_layout"

7	android:layout_width="match_parent"
8	android:layout_height="match_parent"
9	tools:context=".MainActivity">
10	
11	<androidx.fragment.app.FragmentContainerView
12	android:id="@+id/fragment_container_view"
13	android:name="androidx.navigation.fragment.NavHostFragment"
14	android:layout_width="match_parent"
15	android:layout_height="match_parent"
16	app:navGraph="@navigation/nav_graph"
17	app:defaultNavHost="true" />
18	</androidx.constraintlayout.widget.ConstraintLayout>

Tabel 4. 9. Source Code activity_main

10. home_fragment.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:layout_width="match_parent"
6	android:layout_height="match_parent"
7	tools:context=".HomeFragment">
8	
9	<androidx.recyclerview.widget.RecyclerView
10	android:id="@+id/rv_character"
11	android:layout_width="0dp"

12	android:layout_height="0dp"
13	app:layout_constraintBottom_toBottomOf="parent"
14	app:layout_constraintEnd_toEndOf="parent"
15	app:layout_constraintStart_toStartOf="parent"
16	app:layout_constraintTop_toTopOf="parent"
	/>
17	</androidx.constraintlayout.widget.ConstraintLayout>

Tabel 4. 10. Source Code home_fragment

11. item_list.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.cardview.widget.CardView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:layout_width="match_parent"
6	android:layout_height="wrap_content"
7	android:layout_margin="12dp"
8	app:cardCornerRadius="16dp"
9	app:cardElevation="6dp">
10	
11	<androidx.constraintlayout.widget.ConstraintLayout
	ut
12	android:layout_width="match_parent"
13	android:layout_height="wrap_content"
14	android:padding="12dp">
15	
16	<ImageView
17	android:id="@+id/imageView"

18	android:layout_width="100dp"
19	android:layout_height="150dp"
20	android:scaleType="centerCrop"
21	app:layout_constraintTop_toTopOf="parent"
22	app:layout_constraintStart_toStartOf="parent" />
23	
24	<TextView
25	android:id="@+id/textViewName"
26	android:layout_width="0dp"
27	android:layout_height="wrap_content"
28	android:text="Main Title"
29	android:textStyle="bold"
30	android:textSize="18sp"
31	android:textColor="#000000"
32	android:layout_marginStart="12dp"
33	android:layout_marginTop="8dp"
34	app:layout_constraintStart_toEndOf="@id/imageVie
	w"
35	app:layout_constraintEnd_toEndOf="parent"
36	tools:ignore="MissingConstraints" />
37	
38	<TextView
39	android:id="@+id/textViewYear"
40	android:layout_width="0dp"
41	android:layout_height="wrap_content"
42	android:text="1990"
43	android:textSize="16sp"
44	android:textColor="#555555"
45	android:layout_marginStart="12dp"
46	android:layout_marginTop="4dp"
47	app:layout_constraintStart_toEndOf="@id/imageVie
	w"

48	app:layout_constraintTop_toBottomOf="@id/textViewName"
49	app:layout_constraintEnd_toEndOf="parent" />
50	
51	<TextView
52	android:id="@+id/textViewDesc"
53	android:layout_width="0dp"
54	android:layout_height="wrap_content"
55	android:text="Secondary description text that can span multiple lines."
56	android:textSize="15sp"
57	android:textColor="#555555"
58	android:layout_marginStart="12dp"
59	android:layout_marginTop="8dp"
60	app:layout_constraintStart_toEndOf="@id/imageView"
61	app:layout_constraintTop_toBottomOf="@id/textViewYear"
62	app:layout_constraintEnd_toEndOf="parent" />
63	
64	<LinearLayout
65	android:id="@+id/buttonRow"
66	android:layout_width="wrap_content"
67	android:layout_height="wrap_content"
68	android:orientation="horizontal"
69	android:gravity="center"
70	android:layout_marginTop="16dp"
71	app:layout_constraintTop_toBottomOf="@id/imageView"
72	app:layout_constraintBottom_toBottomOf="parent"
73	app:layout_constraintStart_toStartOf="parent"
74	app:layout_constraintEnd_toEndOf="parent">

75	
76	<Button
77	android:id="@+id/buttonLink"
78	android:layout_width="wrap_content"
79	android:layout_height="wrap_content"
80	android:layout_marginEnd="8dp"
81	android:backgroundTint="@color/purple_200"
82	android:text="Detail"
83	android:textAllCaps="false" />
84	
85	<Button
86	android:id="@+id/buttonDetail"
87	android:layout_width="wrap_content"
88	android:layout_height="wrap_content"
89	android:layout_marginStart="8dp"
90	android:backgroundTint="@color/purple_200"
91	android:text="Info"
92	android:textAllCaps="false" />
93	</LinearLayout>
94	
95	</androidx.constraintlayout.widget.ConstraintLayout>
96	</androidx.cardview.widget.CardView>

Tabel 4. 11. Source Code item_list

12. nav_graph.xml

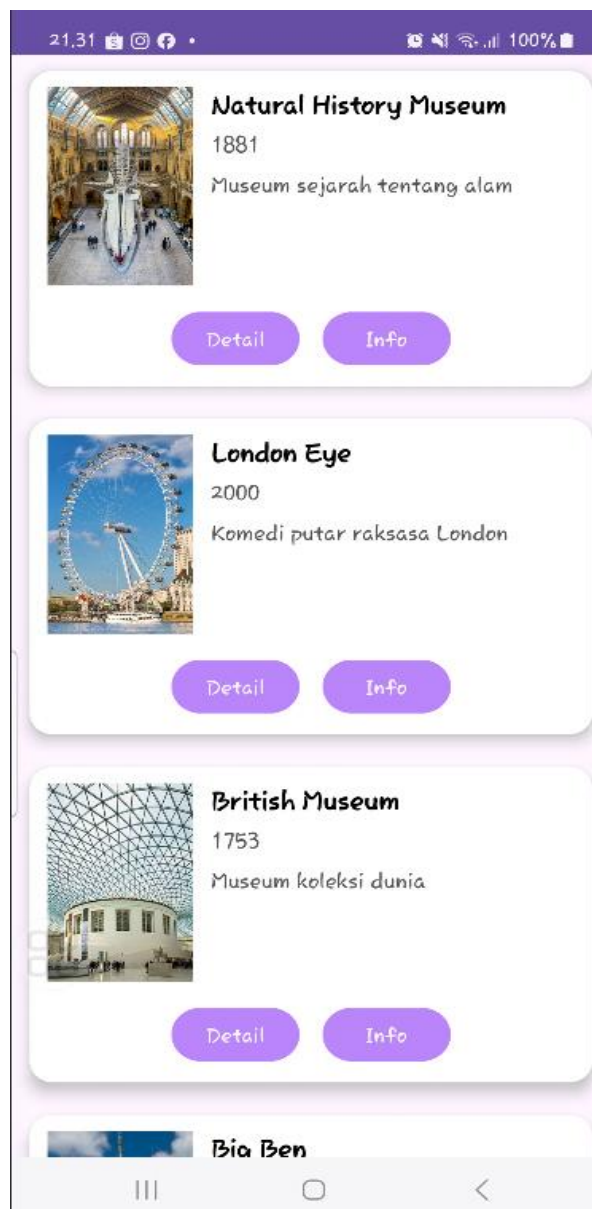
1	<?xml version="1.0" encoding="utf-8"?>
2	<navigation
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-

	auto"
4	android:id="@+id/nav_graph"
5	app:startDestination="@id/HomeFragment">
6	
7	<fragment
8	android:id="@+id/HomeFragment"
9	android:name="com.example.londondestination.Home Fragment"
10	android:label="HomeFragment" >
11	<action
12	android:id="@+id/action_HomeFragment_to_detailFr agment"
13	app:destination="@id/detailFragment"
	/>
14	</fragment>
15	
16	<fragment
17	android:id="@+id/detailFragment"
18	android:name="com.example.londondestination.Frag mentGuweh"
19	android:label="DetailFragment" >
20	<argument
21	android:name="imageResId"
22	app:argType="integer" />
23	<argument
24	android:name="nama"
25	app:argType="string" />
26	<argument
27	android:name="deskripsi"
28	app:argType="string" />
29	</fragment>
30	

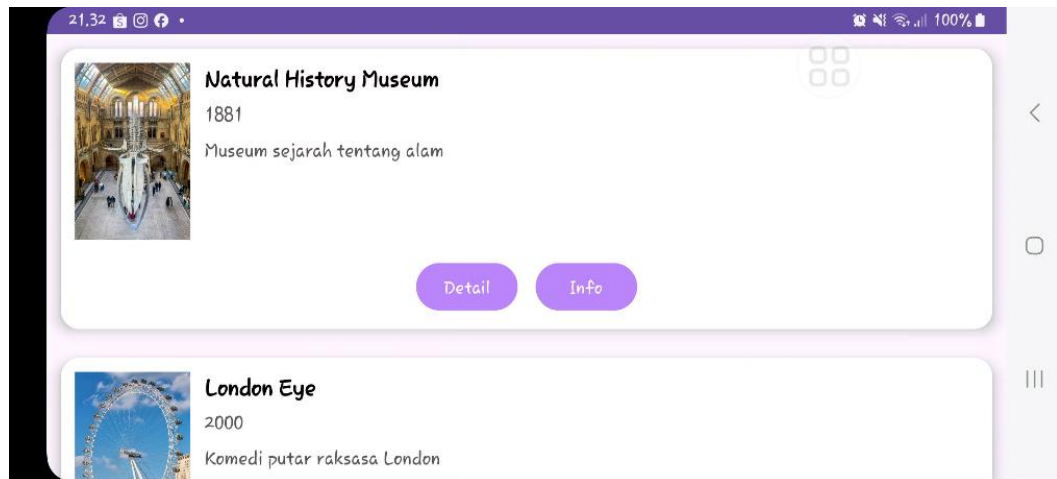
31	</navigation>
----	---------------

Tabel 4. 12. Source Code nav_graph

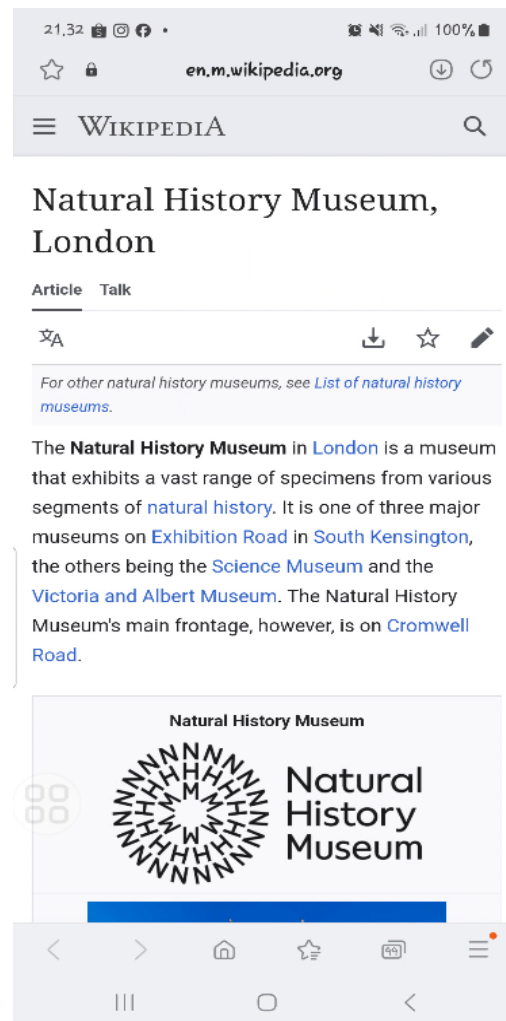
B. Output Program



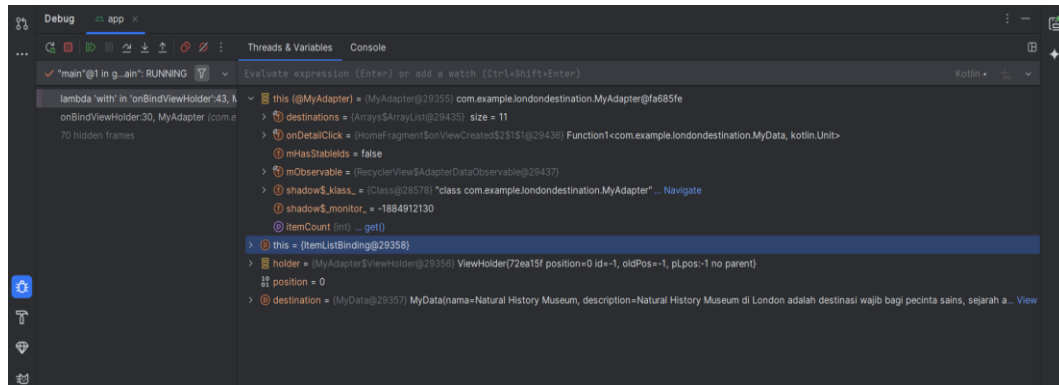
Gambar 4. 1. Screenshot Hasil Jawaban Soal 1



Gambar 4. 2. Screenshot Hasil Jawaban Soal 1



Gambar 4. 3. Screenshot tombol Detail



Gambar 4. 8. Screenshot Debugger Tombol Detail

C. Pembahasan

1. MainActivity.kt

Pada file MainActivity ini merupakan activity utama dalam aplikasi Android yang menggunakan view binding melalui class ActivityMainBinding agar bisa mengakses tampilan yang sudah didefinisikan di activity_main.xml. Dalam metode onCreate disini activity akan diinisialisasi dengan cara memanggil `super.onCreate(savedInstanceState)`, kemudian binding diatur menggunakan `ActivityMainBinding.inflate(layoutInflater)` dan tampilan activity diatur dengan `setContentView(binding.root)`.

Kita lihat dari segi implementasi logging, meskipun file MainActivity yang ditampilkan tidak secara eksplisit menampilkan proses terkait list data, tombol, atau navigasi ke halaman detail, tetapi MainActivity berperan sebagai kontainer utama dari fragment atau komponen lain yang memuat logika tersebut. Mari kita lihat soal d bagian Log saat data item masuk ke dalam list (a), log saat tombol Detail dan tombol Explicit Intent ditekan (b), dan log data dari list yang dipilih ketika berpindah ke halaman Detail (c) kemungkinan besar diatur oleh fragment atau komponen lain yang berada di dalam MainActivity, seperti HomeFragment. Oleh karena itu, meskipun MainActivity

tidak secara langsung menangani logging untuk ketiga event atau permintaan soal tersebut, semua proses tersebut terjadi dalam lifecycle dari MainActivity. Maka, jika diperlukan logging untuk aktivitas aplikasi dapat pula dilakukan di sini, misalnya dengan menambahkan log di dalam onCreate agar bisa mencatat bahwa aktivitas utama dimulai atau bahwa aplikasi dalam keadaan aktif dan siap menampilkan konten yang akan dimasukkan.

2. FragmentGuweh.kt

Pada class FragmentGuweh merupakan implementasi dari sebuah fragment yang berguna untuk menampilkan halaman detail dari suatu item destinasi di aplikasi. Dimana fragment ini menggunakan view binding melalui DetailFragmentBinding agar bisa mengakses komponen UI yang didefinisikan dalam layout detail_fragment.xml. Adanya metode onCreateView layout di-inflate dan binding disiapkan. Selanjutnya, di bagian onViewCreated fragment mengambil data dari argument bundle yang dikirim saat navigasi yaitu imageResId, nama, dan deskripsi, dengan nilai default jika tidak ditemukan. Nilai-nilai ini kemudian digunakan untuk mengatur konten tampilan seperti, gambar, judul, dan deskripsi.

Dilihat terkait implementasi logging untuk event yang disebutkan seperti, log saat data item masuk ke dalam list seperti soal d bagian (a) tidak terjadi di dalam FragmentGuweh karena fragment ini hanya bertugas menampilkan detail satu item bukannya menangani seluruh list. Namun, log saat tombol Detail dan tombol Explicit Intent ditekan seperti soal d bagian (b) dapat dihubungkan menuju fragment ini karena fragment ini dipanggil akibat salah satu dari tombol tersebut ditekan. Maka, saat metode onViewCreated dijalankan dan data ditampilkan dapat dipastikan bahwa salah satu tombol telah ditekan, serta saat itu bisa dilakukan logging. Kita lihat lagi di Log data dari list yang dipilih ketika berpindah ke halaman Detail seperti soal d bagian (c) karena fragment

ini menerima data dari item yang dipilih saja dengan demikian, di dalam `onViewCreated` dapat ditambahkan logging yang mencatat data `imageResId`, nama, dan deskripsi sebagai bentuk pencatatan informasi dari item dipilih saat berpindah menuju halaman detail. Jadi, meskipun log tidak dituliskan secara eksplisit dalam kode yang ditampilkan, `FragmentGuweh` merupakan tempat yang tepat agar bisa mencatat event log terkait pemilihan dan penampilan data detail.

3. HomeFragment.kt

Pada class `HomeFragment` berfungsi sebagai UI utama yang menampilkan daftar destinasi wisata menggunakan `RecyclerView`. Di sini fragment ini dengan arsitektur MVVM yang mana data destinasi diperoleh dari `HomeViewModel` melalui objek `destinationList` merupakan `Flow`. Ketika data dikirim oleh `ViewModel` dan fragment ini mengumpulkannya menggunakan `collectLatest`, lalu menginisialisasi kembali `MyAdapter` dengan data tersebut dan menyetelnya menuju `RecyclerView`. Nah, ini bagian dari alur masuknya data ke dalam list.

Kita lihat di bagian logging untuk event ada saat tombol `Detail` dan tombol `Explicit Intent` karena pencatatan dilakukan di dalam class `MyAdapter` yang diinisialisasi dalam `HomeFragment`. Ketika kita (user) menekan tombol `Detail` atau `Link` pada setiap item daftar, adapter nantinya mencetak log melalui `Log.d` merekam aktivitas kita (user) terhadap item yang ditekan.

Selanjutnya, saat sebuah item dipilih dengan data tersebut diteruskan melalui `selectedItem` di `ViewModel`. Di dalam fragment, data ini dikumpulkan menggunakan `collectLatest` dan jika item tidak null, maka nantinya dicetak log berisi informasi lengkap mengenai item yang dipilih seperti, nama, tahun, dan deskripsi. Setelah log tercetak, fragment melakukan navigasi menuju `DetailFragment` menggunakan

`findNavController().navigate()` dengan membawa data yang diperlukan dalam bentuk `Bundle`.

Jadi, kode `HomeFragment` udah mencakup logging untuk tiga poin penting seperti, saat data masuk ke daftar melalui proses pengumpulan dari `Flow`, saat tombol ditekan melalui adapter, dan saat item dipilih dan digunakan untuk navigasi ke halaman detail.

4. `HomeViewModel.kt`

Pada class `HomeViewModel` ini berperan sebagai bagian dari arsitektur `MVVM` yang berguna dalam menyimpan dan mengelola data dari destinasi wisata. Dimana data tersebut disimpan dalam `_destinationList` sebuah objek `MutableStateFlow` bertipe `List<MyData>` dan kemudian diekspose ke luar sebagai `destinationList` bersifat `StateFlow`. Nah, saat kita inisialisasi `ViewModel` memuat daftar destinasi menuju `_destinationList` dengan membuat list berisi objek-objek `MyData`, lalu menetapkan data ini menjadi nilai `StateFlow`. Di momen data dimuat ke dalam list tersebut, maka dilakukan logging menggunakan `Log.d` dengan "`HomeViewModel`" mencatat jumlah item yang berhasil dimuat ke dalam daftar (`Log.d("HomeViewModel", "List data berhasil dimuat sebanyak ${data.size} item")`). Nah, logging ini menjawab kebutuhan untuk kode ini karena mencatat event saat data item masuk ke dalam list termasuk ke poin soal d bagian (a).

Kemudian, kode juga menyediakan fungsi `onItemClicked` berguna untuk menetapkan sebuah item yang dipilih dari list sebagai nilai `selectedItem`. Ketika fungsi ini dipanggil oleh `HomeFragment` melalui interaksi kita atau pengguna (seperti menekan tombol `Detail` atau `Explicit Intent` di dalam `MyAdapter`), item yang ditekan nantinya disimpan di dalam `_selectedItem`. Dari tombol-tombol tersebut ditangani langsung di adapter (`MyAdapter`) yang sudah disiapkan

di `HomeFragment` dan logging untuk event saat tombol Detail dan tombol Explicit Intent ditekan seperti poin soal d bagian (b) dicatat menggunakan `Log.d` dari dalam adapter saat fungsi-fungsi tersebut dipanggil.

Selain itu, jika item yang dipilih diambil dari `selectedItem` oleh `HomeFragment` dan digunakan agar bisa berpindah menuju halaman detail dimana data dari item tersebut akan dicetak melalui `log`. Nah, logging ini mencakup informasi seperti nama tempat, tahun, dan deskripsi singkat dari item yang dipilih. Hal ini menjawab kebutuhan event `log` data dari list yang dipilih ketika berpindah ke halaman Detail sesuai poin soal d bagian (c). Dengan begitu, proses pencatatan aktivitas penting dalam alur aplikasi telah diimplementasikan sesuai kebutuhan menggunakan `Log.d` dari `ViewModel`, `Adapter`, hingga `Fragment`.

5. `HomeViewModelFactory.kt`

Pada class `HomeViewModelFactory` merupakan implementasi dari `ViewModelProvider.Factory` berguna agar bisa menghasilkan instance dari `HomeViewModel`. Dengan metode `create` yang dilakukan sebagai pemeriksaan apakah `modelClass` turunan dari `HomeViewModel`. Jika benar, maka `HomeViewModel` nantinya dibuat dan dikembalikan sebagai instance dari `T`. Jika tidak sesuai, maka akan dilemparkan exception `IllegalArgumentException` dengan pesan "`Unknown ViewModel class`".

Meskipun kode `HomeViewModelFactory` ini tidak secara langsung memuat data ataupun menangani aksi kita (sebagai pengguna saat mengklik tombol), namun perannya sangat penting dalam memastikan bahwa instance `HomeViewModel` dapat dibuat dan dipergunakan oleh komponen seperti `Fragment` atau `Activity`. Saat proses logging yang berkaitan dengan log saat data item masuk ke dalam list di bagian poin soal d bagian (a) dilakukan di dalam `HomeViewModel` diciptakan melalui factory ini. Jadi, ketika `HomeViewModelFactory` memproduksi

HomeViewModel, maka secara tidak langsung proses logging dari pemuatan data akan terjadi karena sudah tertulis di blok init dalam HomeViewModel.

Kesimpulan pada file ini adalah meskipun HomeViewModelFactory tidak berisi kode logging secara langsung, tetapi peran utamanya itu sebagai penyedia ViewModel yang menjalankan log pada bagian-bagian yang relevan dalam alur aplikasi atau alur data aplikasinya.

6. MyAdapter.kt

Pada class MyAdapter merupakan class untuk RecyclerView dalam aplikasi Android yang menampilkan daftar destinasi wisata di London. Dimana adapter ini menerima dua parameter utama seperti daftar data destinations bertipe List<MyData> dan onDetailClick nantinya dipanggil saat kita klik tombol Detail. Nah, Logging event disini diimplementasikan menggunakan Log.d() agar bisa mencatat aktivitas kita selama run aplikasi atau mengklik salah satu tombol dan alur datanya yang sangat berguna saat kita melakukan debugging.

Pertama di poin soal d bagian (a), kode MyAdapter adalah adapter untuk RecyclerView berguna saat menampilkan daftar objek MyData ke dalam tampilan daftar (list). Di dalam onBindViewHolder itu setiap item dari daftar destinations diproses dan diikat menuju tampilan item_list menggunakan view binding dari ItemListBinding. Saat proses pengikatan ini terjadi berarti data item akan masuk menuju list dan tampil di layar, sehingga event ini bisa dianggap sebagai pencatatan saat data sudah dimasukkan ke dalam daftar.

Kedua di poin soal d bagian (b), logging event sudah diimplementasikan saat kita (user) menekan dua tombol yaitu, tombol Link

dan tombol Detail. Dimana bagian `buttonLink.setOnClickListener` saat kita (user) mengklik tombol membuka URL menggunakan `Intent.ACTION_VIEW` dan mencatat aksi tersebut dengan `Log.d("MyAdapter", "Tombol Link ditekan untuk: ${destination.nama}")`. Begitu juga dengan tombol Detail saat kita klik fungsi `onDetailClick(destination)` dijalankan dan aplikasi mencatat interaksi pengguna dengan `Log.d("MyAdapter", "Tombol Detail ditekan untuk: ${destination.nama}")`. Nah, ini sepenuhnya memenuhi permintaan agar bisa mencatat log pada tombol-tombol tersebut ketika di klik.

Ketiga di poin soal d bagian (c), data dari list yang dipilih ketika berpindah menuju halaman detail dicatat melalui pemanggilan fungsi `onDetailClick(destination)` juga sudah dilengkapi dengan log. Dari fungsi ini berguna untuk membuka halaman baru (misalnya `DetailActivity`) dan membawa data item dipilih. Dengan mencatat `Log.d("MyAdapter", "Tombol Detail ditekan untuk: ${destination.nama}")`, maka data yang dipilih saat pindah halaman pun sudah ter-log dengan jelas.

Jadi, kode ini sudah mengimplementasikan logging untuk ketiga kebutuhan yaitu, pencatatan item yang tampil dalam list, interaksi kita saat klik tombol, dan data yang dipilih untuk detail.

7. MyData.kt

Pada data class bernama `MyData` yang berfungsi sebagai model data utama untuk aplikasi destinasi wisata London. Dimana class ini menggunakan anotasi `@Parcelize` yang memberi kemungkinan objek `MyData` bisa dikirim antar komponen Android seperti antar fragment atau activity dengan lebih mudah melalui `Bundle`. Dengan adanya class ini menyimpan enam properti penting untuk tiap destinasi seperti, nama (nama

tempat wisata), `description` (deskripsi lengkap), `descriptionSingkat` (deskripsi singkat), `year` (tahun pendirian atau peresmian), `image` (ID dari gambar sumber daya), dan `link` (tautan ke halaman informasi lebih lanjut). Pentingnya struktur ini agar aplikasi bisa menampilkan informasi yang lengkap dan interaktif mengenai setiap lokasi wisata dalam bentuk daftar, serta meneruskan data dengan rapi ke tampilan detail saat kita memilih salah satu destinasi. Jadi, file ini menjadi pondasi data yang akan digunakan oleh adapter dan fragment lainnya di aplikasi ini.

8. `Detail_fragment.xml`

Pada bagian ini digunakan untuk menampilkan tampilan detail dari sebuah tempat wisata dalam aplikasi. Dimana seluruh konten dibungkus di dalam sebuah elemen `ScrollView` yang memungkinkan kita bisa menggulir layar ke bawah jika isi kontennya lebih panjang dari ukuran layar. Hal ini penting agar seluruh informasi tetap bisa diakses meskipun banyak atau Panjang isinya.

Di dalam `ScrollView` terdapat sebuah `CardView` yang berfungsi memberikan tampilan lebih menarik karena memiliki sudut melengkung (dengan `cardCornerRadius="16dp"`) dan (`cardElevation="8dp"`), sehingga konten tampak seperti kartu sedikit terangkat dari latar belakang membuat tampilan lebih rapi dan enak dilihat.

Isi dari `CardView` diletakkan di dalam `LinearLayout` yang diatur secara vertikal. Di dalam layout ini terdapat tiga komponen utama yaitu, pertama ada `ImageView` menggunakan ID `detailImage` berguna untuk menampilkan gambar tempat wisata yang biasanya gambar ini nantinya ditampilkan dari file `drawable` atau dari sumber lain. Dengan gambar ini disetel menggunakan `scaleType="centerCrop"` agar mengisi seluruh ruang yang disediakan dengan proporsional.

Selanjutnya, dua `TextView` yang pertama ada `detailTitle` digunakan untuk menampilkan nama tempat dengan ukuran teks yang cukup besar (20sp) dan gaya teks tebal (bold), serta diberi `padding` agar teks tidak menempel langsung ke sisi layar. Yang kedua,

`detailDescription` berfungsi untuk menampilkan deskripsi lengkap dari tempat tersebut dengan deskripsi ini menggunakan ukuran teks sedikit lebih kecil (16sp) dan diberi padding sisi kiri, kanan, dan bawah agar memudahkan saat membaca. Jadi, secara keseluruhan, layout ini dibuat untuk memberikan tampilan detail tempat wisata secara bersih, informatif, dan nyaman dilihat.

9. `activity_main.xml`

Pada bagian ini berguna untuk mengelola navigasi antar-fragment. Dimana seluruh tampilan dibungkus di dalam `ConstraintLayout` merupakan salah satu jenis layout fleksibel yang memungkinkan pengaturan posisi antar elemen secara dinamis dan efisien. Namun dalam kasus ini, hanya ada satu elemen di dalamnya, jadi `ConstraintLayout` tidak benar-benar dimanfaatkan secara penuh. Dengan elemen utamanya itu `FragmentContainerView` berfungsi sebagai wadah (container) agar bisa menampilkan fragment. Adanya `FragmentContainerView` ini memiliki ID `fragment_container_view` dan diatur untuk mengisi seluruh lebar dan tinggi layar (`match_parent`). Terdapat property berisi `android:name` yang menunjuk ke bagian `androidx.navigation.fragment.NavHostFragment` berarti view ini berperan sebagai host fragment nantinya saat mengelola navigasi.

Adanya atribut `app:navGraph` menunjuk ke file `nav_graph` di direktori `res/navigation` berisi struktur navigasi fragment—seperti daftar tujuan (destination) dan hubungan antar fragment (misalnya aksi berpindah antar fragment). Sementara `app:defaultNavHost="true"` digunakan dalam menyatakan bahwa fragment ini adalah host default agar bisa menangani navigasi sistem, seperti tombol "Back" di Android. Jadi, keseluruhan bagian ini XML ini menyusun fondasi navigasi fragment berbasis Jetpack Navigation dengan melakukan penempatan satu `NavHostFragment` untuk mengatur

transisi antar-fragment di dalam aplikasi tanpa perlu berpindah antar activity yang menjadikan navigasi lebih efisien dan lebih terstruktur.

10. home_fragment.xml

Pada bagian ini adalah layout untuk sebuah Fragment lebih tepatnya `HomeFragment` yang menggunakan `ConstraintLayout` sebagai layout utama. Dimana layout ini hanya memiliki satu elemen di dalamnya, yaitu sebuah `RecyclerView` dengan ID `rv_character` dengan fungsi utama dari `RecyclerView` ini untuk menampilkan daftar item secara efisien dan dapat discroll tergantung pada pengaturan adapter dan layout manager-nya.

Pada pengaturannya `RecyclerView` ini dibuat agar memenuhi seluruh ruang layar karena semua constraint-nya dihubungkan ke tepi-tepi parent layout. Hal ini terlihat dari penggunaan `0dp` untuk `layout_width` dan `layout_height` berarti ukuran akan disesuaikan berdasarkan constraint yang diberikan. Dilihat bagian Constraint-nya menghubungkan atas (Top), bawah (Bottom), kiri (Start), dan kanan (End) ke parent, sehingga `RecyclerView` menutupi seluruh tampilan yang tersedia dalam fragment.

Sementara itu, kita lihat di atribut `tools:context=".HomeFragment"` berguna hanya untuk keperluan preview di Android Studio agar tampilan ini bisa dikenali sedang digunakan `HomeFragment` saat mendesain antarmuka. Jadi, secara keseluruhan layout bagian ini sangat sederhana tapi efektif demi bisa menampilkan daftar yang dinamis, seperti daftar tempat wisata, karakter, atau data lain nantinya diisi melalui adapter dalam kode program aplikasi ini.

11. item_list.xml

Pada layout file untuk komponen tampilan dalam aplikasi Android yang mana layout ini menggunakan `CardView` sebagai wadah utama yang memberikan efek tampilan seperti kartu, lengkap dengan sudut melengkung dan bayangan. Di dalam `CardView` terdapat `ConstraintLayout` digunakan sebagai layout utama agar bisa menyusun elemen-elemen UI dengan fleksibilitas tinggi sesuai posisi satu sama lain. Dimana bagian pertama itu `ImageView` berguna untuk menampilkan gambar, dengan ukurannya ditetapkan 100dp x 150dp dan letaknya di pojok kiri atas tampilan. Gambar diatur agar "crop" ke tengah (`centerCrop`) agar mengisi seluruh ruang.

Selanjutnya, ada tiga `TextView` yang masing-masing menampilkan nama (`textViewName`), tahun atau angkatan (`textViewYear`), dan deskripsi tambahan (`textViewDesc`). Dari semua `TextView` ini diletakkan di sebelah kanan `ImageView` yang disusun secara vertikal satu per satu dari atas ke bawah. Adanya `textViewName` yang menggunakan teks tebal dan ukuran huruf lebih besar cocok untuk judul atau nama utama. Sedangkan `textViewYear` dan `textViewDesc` memiliki ukuran teks yang lebih kecil dan warna abu-abu gelap (`#555555`) untuk memberikan perbedaan teks.

Di bagian paling bawah layout terdapat `LinearLayout` yang berisi dua tombol (`Button`), yaitu tombol "Detail" dan "Info". Dengan layout tombol ini diposisikan di bawah `ImageView` dan diberi margin atas agar tidak menempel langsung. Dimana warna latar tombol diambil dari `@color/purple_200` yang memberi warna ungu dari resources aplikasi. Kedua tombol ini disiapkan untuk tindakan lebih lanjut, misalnya "Detail" untuk membuka tampilan biodata lengkap, dan "Info" untuk menampilkan informasi tambahan.

Jadi, secara keseluruhan, XML ini digunakan untuk item dalam `RecyclerView` karena desainnya ringkas, informatif, dan responsif. Dengan struktur layout sudah rapi sesuai pengaturan

`ConstraintLayout` yang fleksibel dan `CardView` mempercantik tampilan menjadi tipe layout biasanya digunakan agar bisa menampilkan daftar data dalam bentuk kartu.

12. `nav_graph.xml`

Pada file `navigation graph` fungsinya untuk mendefinisikan alur navigasi antar tampilan atau fragment dalam sebuah aplikasi. Di bagian dalamnya akan dideklarasikan dua buah fragment yaitu, `HomeFragment` dan `detailFragment`. Dimana layout navigasi ini untuk aplikasi dapat berpindah dari satu fragment menuju fragment lainnya secara terstruktur dan mudah dikelola.

Pertama, elemen `<navigation>` menyatakan bahwa ini bagian root dari `navigation graph` yang terdapat atribut `app:startDestination="@id/HomeFragment"` yang menandakan bahwa saat aplikasi dijalankan, maka tampilan awal (halaman pertama) nantinya ditampilkan itu `HomeFragment`. Dengan fragment ini diberi ID `@+id/HomeFragment` dan terhubung menuju bagian dari `class com.example.londondestination.HomeFragment`.

Di dalam `HomeFragment` terdapat elemen `<action>` dengan ID `action_HomeFragment_to_detailFragment` merupakan aksi navigasi yang berarti jika dipicu (misalnya kita klik tombol), maka aplikasi akan berpindah dari `HomeFragment` menuju `detailFragment`. Nama action ini bisa dipanggil di kode Kotlin atau Java untuk mentrigger perpindahan halaman ke selanjutnya.

Adanya fragment tujuan yaitu `detailFragment` dikaitkan dengan class Java atau Kotlin `com.example.londondestination.FragmentGuweh`. Dimana fragment ini nantinya menerima data melalui tiga buah argument seperti, `imageResId` (tipe integer, biasanya ID gambar dari resource drawable), nama (tipe string), dan deskripsi (juga string) dari ketiga argumen ini

memberi kemungkinan fragment tujuan bisa menerima data yang dikirim dari HomeFragment, misalnya saat kita klik tombol "Detail". Jadi, file ini memastikan bahwa aplikasi punya alur navigasi yang rapi dari halaman utama ke halaman detail, dan sudah siap menerima data agar bisa ditampilkan.

13. Debugger

Pada debugger saya menggunakan file MyAdapter karena breakpoint dipasang di dua lokasi penting, yaitu `buttonLink.setOnClickListener` dan `buttonDetail.setOnClickListener`. Dimana breakpoint berguna untuk memastikan bahwa kedua tombol berfungsi sebagaimana mestinya dan bahwa data yang ditampilkan atau dikirim sesuai yang diharapkan. Sebagai contoh, ketika saya klik tombol "Link", maka debugger dapat memperlihatkan nilai `destination.link` dan memastikan intent menuju browser terbentuk dengan benar. Begitu pula saat tombol "Detail" ditekan, maka debugger dapat menunjukkan bahwa fungsi `onDetailClick(destination)` menerima data yang benar seperti nama dan deskripsi destinasi.

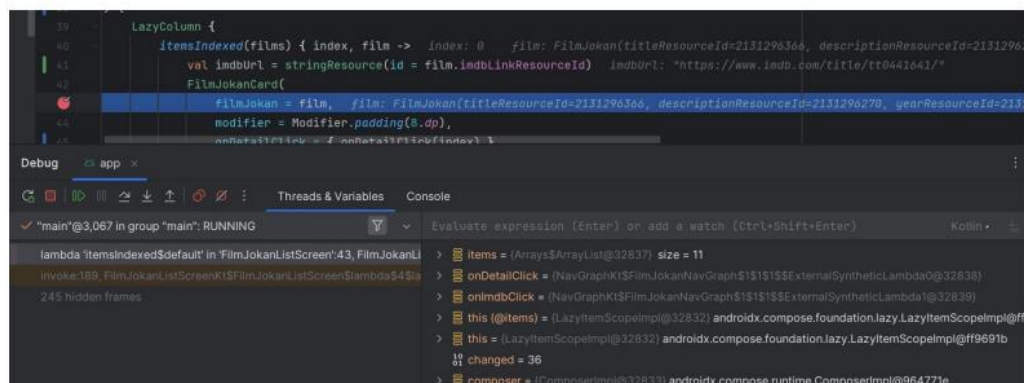
Selama proses debugging, Android Studio menyediakan tiga fitur utama yaitu, Step Into, Step Over, dan Step Out. Dari fitur Step Into digunakan untuk masuk ke dalam detail implementasi sebuah metode atau fungsi yang sedang dipanggil, misalnya masuk ke dalam `onDetailClick(destination)`. Hal ini sangat berguna saat kita ingin mengetahui lebih dalam bagaimana suatu fungsi bekerja. Sementara itu, Step Over digunakan untuk melompati baris kode saat ini tanpa perlu masuk ke dalam fungsi, cocok digunakan jika kita sudah yakin bahwa fungsi tersebut bekerja dengan baik dan tidak perlu ditelusuri lagi. Lalu, ada Step Out digunakan untuk keluar dari fungsi yang sedang ditelusuri dan kembali ke pemanggilnya, sangat membantu ketika kita sudah masuk terlalu dalam menuju struktur kode dan ingin kembali ke bagian level sebelumnya.

Dengan cara memanfaatkan breakpoint dan fitur langkah demi langkah tersebut, proses penelusuran bug menjadi jauh lebih mudah dan efisien. Kemudian, dilihat lagi *gambar output program 7 dan 8* yang memperlihatkan bahwa proses debugging berhasil menghentikan eksekusi program di dalam `onBindViewHolder` dan nilai objek `destination` telah ditampilkan dengan benar di jendela debugger, menandakan bahwa data yang digunakan saat itu sesuai dengan yang diharapkan.

SOAL 2

Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

Aplikasi harus dapat mempertahankan fitur-fitur yang sudah dibuat pada modul sebelumnya. Berikut adalah contoh debugging dalam Android Studio.



Gambar 4. 9. Contoh Penggunaan Debugger

A. Pembahasan

Pada Application berfungsi sebagai titik awal dan pusat konfigurasi aplikasi Android sebelum Activity atau Service pertama dipanggil. Untuk menggunakannya, pengembang cukup membuat class yang mewarisi dari `android.app.Application` dan mendaftarkannya di file `AndroidManifest.xml`.

Nah, kita masuk ke bagian proses debugging pada kode MyAdapter dimana saya sudah menempatkan breakpoint di dua bagian penting, yaitu `buttonLink.setOnClickListener` dan `buttonDetail.setOnClickListener`. Dari kedua bagian kode ini menangani interaksi kita (user) saat tombol ditekan pada tiap item RecyclerView. Ketika debugging mencapai kode tombol "Link", maka debugger nantinya memperlihatkan bahwa nilai `destination.link` sudah benar dan menghasilkan intent dengan `ACTION_VIEW` agar bisa membuka tautan

di browser. Hal ini memastikan bahwa aplikasi akan mengarahkan kita ke situs yang sesuai. Sedangkan pada tombol "Detail" ini debugger menunjukkan bahwa fungsi `onDetailClick(destination)` berhasil dipanggil dengan objek `destination` yang benar. Misalnya nama, `description` singkat, dan properti lainnya sesuai data dikirimkan. Informasi inilah yang sangat penting untuk memastikan bahwa interaksi kita (user) diproses sesuai yang diharapkan.

Penggunaan debugger seperti ini memungkinkan kita (user) agar bisa menelusuri alur program lebih teliti. Saat breakpoint aktif, saya mulai menggunakan fitur `Step Into` untuk masuk ke dalam implementasi fungsi (misalnya `onDetailClick`), `Step Over` untuk melewati eksekusi fungsi tanpa masuk ke dalamnya, dan `Step Out` untuk keluar dari fungsi saat ini dan kembali ke pemanggilnya. Dari seluruh fitur ini (`Step Into`, `Step Over`, dan `Step Out`) sangat berguna dalam memahami secara mendalam tentang bagaimana cara aplikasi bekerja dan memastikan bahwa tidak ada kesalahan logika atau data yang salah selama proses interaksi kita (saat kita mencoba mengklik salah satu tombol bagik itu detail maupun info) dengan tampilan aplikasi yang dimunculkan di layar.

MODUL 5 : Connect to the Internet

SOAL 1

Soal Praktikum:

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

1. MyApiResponse.kt

1	package com.example.myapi_test.data.api
2	
3	// DTOs that exactly match the JSON response from the API.
4	
5	data class BookApiResponse(6 val data: List<BookData> 7)
8	
9	data class BookData(10 val attributes: BookAttributes 11)
12	
13	data class BookAttributes(14 val author: String, 15 val cover: String, 16 val release_date: String, 17 val title: String, 18 val wiki: String, 19 val summary: String 20)

Tabel 5. 1. Source Code MyApiResponse

2. MyInstance

1	package com.example.myapi_test.data.api
2	
3	import retrofit2.Retrofit
4	import retrofit2.converter.gson.GsonConverterFactory
5	

6	object MyInstance {
7	private const val BASE_URL =
	"https://api.potterdb.com/"
8	
9	val api: MyService by lazy {
10	Retrofit.Builder()
11	.baseUrl(BASE_URL)
12	.addConverterFactory(GsonConverterFactory.create())
13	.build()
14	.create(MyService::class.java)
15	}
16	}

Tabel 5. 2. Source Code MyApiResponse

3. MyService

1	package com.example.myapi_test.data.api
2	
3	import retrofit2.http.GET
4	
5	interface MyService {
6	@GET("v1/books")
7	suspend fun getMessage(): BookApiResponse
8	}

Tabel 5. 3. Source Code MyService

4. BookDao

1	package com.example.myapi_test.data.database.dao
2	
3	import androidx.room.Dao
4	import androidx.room.Insert

5	import androidx.room.OnConflictStrategy
6	import androidx.room.Query
7	import
	com.example.myapi_test.data.database.entity.Book
	DbEntity
8	
9	@Dao
10	interface BookDao {
11	/**
12	<i>* Inserts a list of books into the</i>
	<i>database. If a book with the same</i>
13	<i>* primary key already exists, it will be</i>
	<i>replaced.</i>
14	*/
15	@Insert(onConflict =
	OnConflictStrategy.REPLACE)
16	suspend fun insertBooks(books:
	List<BookDbEntity>)
17	
18	/**
19	<i>* Retrieves all books from the database,</i>
	<i>ordered by title.</i>
20	<i>* @return A list of all BookDbEntity</i>
	<i>objects.</i>
21	*/
22	@Query("SELECT * FROM books ORDER BY title
	ASC")
23	suspend fun getAllBooks():
	List<BookDbEntity>
24	
25	/**
26	<i>* Deletes all books from the database.</i>

27	<code>*/</code>
28	<code>@Query("DELETE FROM books")</code>
29	<code>suspend fun clearAllBooks()</code>
30	<code>}</code>

Tabel 5. 4. Source Code BookDao

5. BookDbEntity

1	<code>package</code>
	<code>com.example.myapi_test.data.database.entity</code>
2	
3	<code>import androidx.room.Entity</code>
4	<code>import androidx.room.PrimaryKey</code>
5	
6	<code>/**</code>
7	<code> * Defines the schema for the 'books' table in</code>
	<code>the Room database.</code>
8	<code> * Each instance of this class represents a row</code>
	<code>in the table.</code>
9	<code>*/</code>
10	<code>@Entity(tableName = "books")</code>
11	<code>data class BookDbEntity(</code>
12	<code> // We use the title as the primary key,</code>
	<code>assuming it's unique.</code>
13	<code> // For real-world apps, a unique ID from the</code>
	<code>API is preferable.</code>
14	<code> @PrimaryKey</code>
15	<code> val title: String,</code>
16	<code> val author: String,</code>
17	<code> val cover: String,</code>
18	<code> val releaseDate: String,</code>
19	<code> val summary: String,</code>

20	val wiki: String
21)

Tabel 5. 5. Source Code BookDbEntity

6. AppDatabase

1	package com.example.myapi_test.data.database
2	
3	import android.content.Context
4	import androidx.room.Database
5	import androidx.room.Room
6	import androidx.room.RoomDatabase
7	import
	com.example.myapi_test.data.database.dao.BookDao
8	import
	com.example.myapi_test.data.database.entity.Book
	DbEntity
9	
10	@Database(entities = [BookDbEntity::class],
	version = 1, exportSchema = false)
11	abstract class AppDatabase : RoomDatabase() {
12	
13	abstract fun bookDao(): BookDao
14	
15	companion object {
16	// Volatile ensures that the INSTANCE is
	always up-to-date and the same for all execution
	threads.
17	@Volatile
18	private var INSTANCE: AppDatabase? =
	null
19	
20	fun getDatabase(context: Context):

	AppDatabase {
21	// Return the existing instance if it exists, otherwise create a new one.
22	return INSTANCE ?:
	synchronized(this) {
23	val instance =
	Room.databaseBuilder(
24	context.applicationContext,
25	AppDatabase::class.java,
26	"book_database"
27)
28	.fallbackToDestructiveMigrat
	ion() // Strategy for handling version changes
29	.build()
30	INSTANCE = instance
31	instance
32	}
33	}
34	
35	}

Tabel 5. 6. Source Code AppDatabase

7. BookMapper.kt

1	package com.example.myapi_test.data.mappers
2	
3	import com.example.myapi_test.data.api.BookData
4	import com.example.myapi_test.domain.model.Book
5	import
	com.example.myapi_test.presentation.model.BookUi
6	import
	com.example.myapi_test.data.api.BookAttributes
7	import

	com.example.myapi_test.data.database.entity.Book
	DbEntity
8	/**
9	* Converts the data layer's BookData (DTO) into
	a domain layer Book entity.
10	* This is a key part of separating the data and
	domain layers.
11	*/
12	fun BookData.toDomain(): Book {
13	return Book(
14	title = this.attributes.title,
15	author = this.attributes.author,
16	cover = this.attributes.cover,
17	releaseDate =
	this.attributes.release_date,
18	summary = this.attributes.summary,
19	wiki = this.attributes.wiki
20)
21	}
22	fun Book.toUiModel(): BookUi {
23	return BookUi(
24	title = this.title,
25	author = this.author,
26	cover = this.cover,
27	release_date = this.releaseDate,
28	summary = this.summary,
29	wiki = this.wiki
30)
31	}
32	/**
33	* Converts the network DTO (BookAttributes) to
	a database entity (BookDbEntity).

34	<code>*/</code>
35	<code>fun BookAttributes.toDbEntity(): BookDbEntity {</code>
36	<code> return BookDbEntity(</code>
37	<code> title = this.title,</code>
38	<code> author = this.author,</code>
39	<code> cover = this.cover,</code>
40	<code> releaseDate = this.release_date,</code>
41	<code> summary = this.summary,</code>
42	<code> wiki = this.wiki</code>
43	<code>)</code>
44	<code>}</code>
45	
46	<code>/**</code>
47	<code> * Converts a database entity (BookDbEntity) to</code>
	<code>a domain model (Book).</code>
48	<code>*/</code>
49	<code>fun BookDbEntity.toDomain(): Book {</code>
50	<code> return Book(</code>
51	<code> title = this.title,</code>
52	<code> author = this.author,</code>
53	<code> cover = this.cover,</code>
54	<code> releaseDate = this.releaseDate,</code>
55	<code> summary = this.summary,</code>
56	<code> wiki = this.wiki</code>
57	<code>)</code>
58	<code>}</code>

Tabel 5. 7. Source Code BookMapper.kt

8. BookRepositoryImpl

1	package com.example.myapi_test.data.repository
2	
3	import android.content.Context
4	import android.util.Log
5	import
	com.example.myapi_test.data.api.MyInstance
6	import
	com.example.myapi_test.data.database.AppDatabase
7	import
	com.example.myapi_test.data.mappers.toDbEntity
8	import
	com.example.myapi_test.data.mappers.toDomain
9	import com.example.myapi_test.domain.model.Book
10	import
	com.example.myapi_test.domain.repository.BookRepository
11	import kotlinx.coroutines.Dispatchers
12	import kotlinx.coroutines.withContext
13	
14	/**
15	* Implements the BookRepository. It now manages
	two data sources:
16	* 1. Remote: MyInstance (Retrofit API)
17	* 2. Local: AppDatabase (Room DAO)
18	*/
19	class BookRepositoryImpl(context: Context) :
	BookRepository {
20	
21	private val apiService = MyInstance.api
22	private val bookDao =
	AppDatabase.getDatabase(context).bookDao()

23	
24	override suspend fun getBooks():
	Result<List<Book>> {
25	return withContext(Dispatchers.IO) {
26	try {
27	// 1. Fetch fresh data from the
	API
28	Log.d("BookRepository",
	"Fetching books from API...")
29	val remoteBooks =
	apiService.getMessage().data
30	
31	// 2. Clear old data from the
	database
32	bookDao.clearAllBooks()
33	
34	// 3. Map network DTOs to
	database entities and insert them
35	val dbEntities = remoteBooks.map
	{ it.attributes.toDbEntity() }
36	bookDao.insertBooks(dbEntities)
37	Log.d("BookRepository",
	"Successfully inserted \${dbEntities.size} books
	into database.")
38	
39	} catch (e: Exception) {
40	// 4. If the network call fails,
	log the error.
41	// The function will proceed to
	load data from the cache.
42	Log.e("BookRepository", "Failed
	to fetch from API. Loading from cache.", e)

43	}
44	
45	// 5. Always return data from the database (Single Source of Truth)
46	// If the network call succeeded, this is the fresh data.
47	// If it failed, this is the old, cached data.
48	try {
49	val cachedBooks = bookDao.getAllBooks()
50	Log.d("BookRepository", "Loaded \${cachedBooks.size} books from database.")
51	Result.success(cachedBooks.map { it.toDomain() })
52	} catch (e: Exception) {
53	Log.e("BookRepository", "Failed to read from database.", e)
54	Result.failure(e)
55	}
56	}
57	}
58	}

Tabel 5. 8. *Source Code BookRepositoryImpl*

9. Book

1	package com.example.myapi_test.domain.model
2	
3	/**
4	* Represents the core business object (Entity). It is a pure data class,
5	* independent of any data source or UI

	<i>implementation details.</i>
6	<i>*/</i>
7	<code>data class Book(</code>
8	<code> val title: String,</code>
9	<code> val author: String,</code>
10	<code> val cover: String,</code>
11	<code> val releaseDate: String,</code>
12	<code> val summary: String,</code>
13	<code> val wiki: String</code>
14	<code>)</code>

Tabel 5. 9. Source Code Book

10. BookRepository

1	<code>package com.example.myapi_test.domain.repository</code>
2	
3	<code>import com.example.myapi_test.domain.model.Book</code>
4	
5	<i>/**</i>
6	<i> * Defines a contract for data operations that</i>
	<i>the Data layer must implement.</i>
7	<i> * The domain layer uses this interface to</i>
	<i>access data, keeping it decoupled</i>
8	<i> * from the data source's implementation (e.g.,</i>
	<i>Retrofit, Room).</i>
9	<i>*/</i>
10	<code>interface BookRepository {</code>
11	<i>/**</i>
12	<i> * Fetches a list of books from the data</i>
	<i>source.</i>
13	<i> * @return A Result wrapper containing</i>
	<i>either the list of books on success or an</i>
	<i>exception on failure.</i>

14	<code>*/</code>
15	<code>suspend fun getBooks(): Result<List<Book>></code>

Tabel 5. 10. Source Code BookRepository

11. GetBooksUseCase

1	<code>package com.example.myapi_test.domain.usecase</code>
2	
3	<code>import com.example.myapi_test.domain.model.Book</code>
4	<code>import</code> <code>com.example.myapi_test.domain.repository.BookRepository</code>
5	
6	<code>/**</code>
7	<code> * A Use Case that encapsulates the business</code> <code>logic for fetching the list of books.</code>
8	<code> * It depends on the BookRepository interface,</code> <code>not its implementation.</code>
9	<code>*/</code>
10	<code>class GetBooksUseCase(private val</code> <code>bookRepository: BookRepository) {</code>
11	
12	<code> /**</code>
13	<code> * Executes the use case. The 'invoke'</code> <code>operator allows this class to be called</code>
14	<code> * like a function, e.g., getBooksUseCase().</code>
15	<code> */</code>
16	<code>suspend operator fun invoke():</code> <code>Result<List<Book>> {</code>
17	<code> return bookRepository.getBooks()</code>
18	<code>}</code>
19	<code>}</code>

Tabel 5. 11. Source Code GetBooksUseCase

12. BookUi

1	package
	com.example.myapi_test.presentation.model
2	
3	import android.os.Parcelable
4	import kotlinx.parcelize.Parcelize
5	
6	/**
7	* A data model specifically for the UI
	(Presentation Layer).
8	* It implements Parcelable to be passed between
	Android components like Fragments.
9	*/
10	@Parcelize
11	data class BookUi(
12	val author: String,
13	val cover: String,
14	val release_date: String,
15	val title: String,
16	val wiki: String,
17	val summary: String
18) : Parcelable

Tabel 5. 12. Source Code BookUi

13. BookUi

1	package com.example.myapi_test.presentation.ui
2	
3	import android.os.Build
4	import android.os.Bundle
5	import android.view.LayoutInflater
6	import android.view.View
7	import android.view.ViewGroup

8	import androidx.appcompat.app.AppCompatActivity
9	import androidx.fragment.app.Fragment
10	import com.bumptech.glide.Glide
11	import com.example.myapi_test.R
12	import
	com.example.myapi_test.databinding.DetailFragmen
	tBinding
13	import
	com.example.myapi_test.presentation.model.BookUi
14	
15	class DetailFragment : Fragment() {
16	
17	private var _binding: DetailFragmentBinding?
	= null
18	private val binding get() = _binding!!
19	
20	private var currentBook: BookUi? = null
21	
22	override fun onCreate(savedInstanceState: Bundle?) {
	super.onCreate(savedInstanceState)
23	arguments?.let {
24	currentBook = if
25	(Build.VERSION.SDK_INT >=
	Build.VERSION_CODES.TIRAMISU) {
26	it.getParcelable(ARG_BOOK,
	BookUi::class.java)
27	} else {
28	@Suppress("DEPRECATION")
29	it.getParcelable(ARG_BOOK)
30	}
31	}

32	}
33	
34	override fun onCreateView(
35	inflater: LayoutInflater, container:
36	ViewGroup?,
37	savedInstanceState: Bundle?
38): View {
39	_binding =
40	DetailFragmentBinding.inflate(inflater,
41	container, false)
42	return binding.root
43	}
44	
45	override fun onViewCreated(view: View,
46	savedInstanceState: Bundle?) {
47	super.onViewCreated(view,
48	savedInstanceState)
49	
50	val toolbar:
51	androidx.appcompat.widget.Toolbar =
52	view.findViewById(R.id.detailToolbar)
53	val activity = requireActivity() as
	AppCompatActivity
	activity.supportActionBar(toolbar)
	activity.supportActionBar?.setDisplayHomeAsUpEnabled(true)
	activity.supportActionBar?.setDisplayHomeAsUpEnabled(true)
	toolbar.setNavigationOnClickListener {
	// Handle back press
	activity.onBackPressed()

54	}
55	currentBook?.let { book ->
56	binding.detailTitle.text =
	book.title
57	binding.detailDescription.text =
	book.summary
58	
59	if (book.cover.isNotBlank()) {
60	Glide.with(this)
61	.load(book.cover)
62	.into(binding.detailImage)
63	} else {
64	binding.detailImage.setImageResource(R.drawable.
	ic_launcher_background)
65	}
66	}
67	}
68	
69	override fun onDestroyView() {
70	super.onDestroyView()
71	_binding = null
72	}
73	
74	companion object {
75	private const val ARG_BOOK =
	"book_ui_parcel"
76	
77	@JvmStatic
78	fun newInstance(book: BookUi):
	DetailFragment {
79	return DetailFragment().apply {

80	<code>arguments = Bundle().apply {</code>
81	<code>putParcelable(ARG_BOOK,</code>
	<code>book)</code>
82	<code>}</code>
83	<code>}</code>
84	<code>}</code>
85	<code>}</code>
86	<code>}</code>

Tabel 5. 13. Source Code DetailFragment

14. HomeFragment

1	<code>package com.example.myapi_test.presentation.ui</code>
2	
3	<code>import android.content.Intent</code>
4	<code>import android.net.Uri</code>
5	<code>import android.os.Bundle</code>
6	<code>import android.util.Log</code>
7	<code>import android.view.LayoutInflater</code>
8	<code>import android.view.View</code>
9	<code>import android.view.ViewGroup</code>
10	<code>import android.widget.Toast</code>
11	<code>import androidx.fragment.app.Fragment</code>
12	<code>import androidx.lifecycle.ViewModelProvider</code>
13	<code>import</code>
	<code>androidx.recyclerview.widget.LinearLayoutManager</code>
14	<code>import com.example.myapi_test.R</code>
15	<code>import</code>
	<code>com.example.myapi_test.databinding.HomeFragmentB</code>
	<code>inding</code>
16	<code>import</code>
	<code>com.example.myapi_test.presentation.model.BookUi</code>
17	<code>import</code>

	com.example.myapi_test.presentation.viewmodel.BookViewModel
18	
19	class HomeFragment : Fragment() {
20	
21	private var _binding: HomeFragmentBinding? =
	null
22	private val binding get() = _binding!!
23	
24	private lateinit var bookViewModel:
	BookViewModel
25	private lateinit var myAdapter: MyAdapter
26	
27	override fun onCreateView(
28	inflater: LayoutInflater, container:
	ViewGroup?,
29	savedInstanceState: Bundle?
30): View {
31	_binding =
	HomeFragmentBinding.inflate(inflater, container,
	false)
32	return binding.root
33	}
34	
35	override fun onViewCreated(view: View,
	savedInstanceState: Bundle?) {
36	super.onViewCreated(view,
	savedInstanceState)
37	
38	// Share the ViewModel with the hosting
	Activity
39	bookViewModel =

	<pre> ViewModelProvider(requireActivity()).get(BookView viewModel::class.java) 40 41 setupRecyclerView() 42 observeViewModel() 43 44 // Fetch data only if the list is empty 45 if (bookViewModel.booksLiveData.value.isNullOrEmpty ()) { 46 bookViewModel.fetchBooks() 47 } 48 } 49 50 private fun setupRecyclerView() { 51 myAdapter = MyAdapter(52 mutableListOf(), 53 onDetailButtonClicked = { book -> navigateToDetail(book) }, 54 onInfoButtonClicked = { book -> openWikiLink(book) }, 55 onItemRootClicked = { book -> navigateToDetail(book) } 56) 57 58 binding.rvCharacter.apply { 59 layoutManager = LinearLayoutManager(requireContext()) 60 adapter = myAdapter 61 } 62 } 63 </pre>
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

64	private fun observeViewModel() {
65	// Observe the list of books and update the adapter
66	bookViewModel.booksLiveData.observe(<i>viewLifecycleOwner</i>) { books ->
67	books?.let
68	{ myAdapter.updateBooks(it) }
69	}
70	// The MainActivity handles observing isLoading and errorMessage for global UI feedback
71	}
72	
73	private fun navigateToDetail(book: BookUi) {
74	Log.d("HomeFragment", "Navigating to detail for: \${book.title}")
75	val detailFragment = DetailFragment.newInstance(book)
76	<i>parentFragmentManager</i> .beginTransaction()
77	.replace(R.id. <i>fragmentContainer</i> , detailFragment)
78	.addToBackStack(null)
79	.commit()
80	}
81	
82	private fun openWikiLink(book: BookUi) {
83	Log.d("HomeFragment", "Info button clicked for: \${book.title}")
84	if (book.wiki.isNotBlank()) {
85	try {
86	val intent =

	Intent(Intent.ACTION_VIEW, Uri.parse(book.wiki))
87	startActivity(intent)
88	} catch (e: Exception) {
89	Log.e("HomeFragment", "Could not
	open URL: \${book.wiki}", e)
90	Toast.makeText(requireContext(),
	"Could not open link.",
	Toast.LENGTH_SHORT).show()
91	}
92	} else {
93	Toast.makeText(requireContext(), "No
	info link available.",
	Toast.LENGTH_SHORT).show()
94	}
95	}
96	
97	override fun onDestroyView() {
98	super.onDestroyView()
99	_binding = null
100	}
101	}

Tabel 5. 14. Source Code HomeFragment

15. MainActivity

1	package com.example.myapi_test.presentation.ui
2	
3	import android.os.Bundle
4	import android.view.View
5	import android.widget.Toast
6	import androidx.appcompat.app.AppCompatActivity
7	import
	androidx.core.splashscreen.SplashScreen.Companion

	n.installSplashScreen // <-- Import this
8	import androidx.lifecycle.ViewModelProvider
9	import com.example.myapi_test.R
10	import com.example.myapi_test.databinding.ActivityMainB inding
11	import com.example.myapi_test.presentation.viewmodel.Bo okViewModel
12	import com.example.myapi_test.presentation.viewmodel.Bo okViewModelFactory
13	
14	class MainActivity : AppCompatActivity() {
15	
16	private lateinit var binding: ActivityMainBinding
17	private lateinit var bookViewModel: BookViewModel
18	
19	override fun onCreate(savedInstanceState: Bundle?) {
20	// 1. Handle the splash screen transition. Must be called before super.onCreate().
21	val splashScreen = installSplashScreen()
22	super.onCreate(savedInstanceState)
23	
24	// Standard view binding setup
25	binding = ActivityMainBinding.inflate(layoutInflater)
26	setContentView(binding.root)

27	
28	<pre> // Initialize ViewModel using the factory </pre>
29	<pre> val viewModelFactory = BookViewModelFactory(application) </pre>
30	<pre> bookViewModel = ViewModelProvider(this, viewModelFactory)[BookViewModel::class.java] </pre>
31	
32	<pre> // 2. Keep the splash screen visible until the initial data is ready. </pre>
33	<pre> // This links the splash screen's duration to your app's actual loading state. </pre>
34	<pre> splashScreen.setKeepOnScreenCondition { </pre>
35	<pre> bookViewModel.isLoadingLiveData.value == true </pre>
36	<pre> } </pre>
37	
38	<pre> // Add the main fragment only if the activity is newly created </pre>
39	<pre> if (savedInstanceState == null) { </pre>
40	<pre> supportFragmentManager.beginTransaction() </pre>
41	<pre> .replace(R.id.fragmentContainer, HomeFragment()) </pre>
42	<pre> .commitNow() </pre>
43	<pre> } </pre>
44	
45	<pre> observeViewModel() </pre>
46	<pre> } </pre>
47	
48	<pre> /** </pre>
49	<pre> * Sets up observers for global UI states </pre>

	<i>like loading indicators and error messages.</i>
50	*/
51	private fun observeViewModel() {
52	// Observe loading state to show/hide
	the ProgressBar
53	bookViewModel.isLoadingLiveData.observe(this)
	{ isLoading ->
54	binding.progressBar.visibility = if
	(isLoading) View.VISIBLE else View.GONE
55	}
56	
57	// Observe errors to show a Toast
	message
58	bookViewModel.errorLiveData.observe(this)
	{ errorMessage ->
59	errorMessage?.let {
60	Toast.makeText(this, it,
	Toast.LENGTH_LONG).show()
61	}
62	}
63	}
64	}

Tabel 5. 15. Source Code MainActivity

16. MyAdapter

1	package com.example.myapi_test.presentation.ui
2	
3	import android.util.Log
4	import android.view.LayoutInflater
5	import android.view.ViewGroup
6	import androidx.recyclerview.widget.RecyclerView
7	import com.bumptech.glide.Glide

8	import com.example.myapi_test.R
9	import
	com.example.myapi_test.databinding.ItemListBindi
	ng
10	import
	com.example.myapi_test.presentation.model.BookUi
11	
12	class MyAdapter(
13	private var books: MutableList<BookUi>,
14	private val onDetailButtonClicked: (book:
	BookUi) -> Unit,
15	private val onInfoButtonClicked: (book:
	BookUi) -> Unit,
16	private val onItemRootClicked: (book:
	BookUi) -> Unit
17) :
	RecyclerView.Adapter<MyAdapter.BookViewHolder>()
	{
18	
19	inner class BookViewHolder(val binding:
	ItemListBinding) :
20	RecyclerView.ViewHolder(binding.root) {
21	
22	init {
23	// Listener for the "Detail" button
24	binding.buttonInfo.setOnClickListener {
25	val position = adapterPosition
26	if (position !=
	RecyclerView.NO_POSITION) {
27	onDetailButtonClicked(books[position])
28	}

29	}
30	
31	// Listener for the "Info" button
32	binding.buttonDetail.setOnClickListener {
33	val position = <i>adapterPosition</i>
34	if (position !=
	RecyclerView.NO_POSITION) {
35	onInfoButtonClicked(books[position])
36	}
37	}
38	
39	// Listener for the entire item
	click
40	binding.root.setOnClickListener {
41	val position = <i>adapterPosition</i>
42	if (position !=
	RecyclerView.NO_POSITION) {
43	onItemRootClicked(books[position])
44	}
45	}
46	}
47	
48	fun bind(book: BookUi) {
49	binding.apply {
50	textViewName.text = book.title
51	textViewYear.text =
	book.release_date
52	textViewAuthor.text =
	book.author
53	

54	<code>Glide.with(itemView.context)</code>
55	<code>.load(book.cover)</code>
56	<code>.into(imageView)</code>
57	<code>}</code>
58	<code>}</code>
59	<code>}</code>
60	
61	<code>override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookViewHolder {</code>
62	<code>val binding =</code>
	<code>ItemListBinding.inflate(LayoutInflater.from(pare</code>
	<code>nt.context), parent, false)</code>
63	<code>return BookViewHolder(binding)</code>
64	<code>}</code>
65	
66	<code>override fun getItemCount() = books.size</code>
67	
68	<code>override fun onBindViewHolder(holder: BookViewHolder, position: Int) {</code>
69	<code>holder.bind(books[position])</code>
70	<code>}</code>
71	
72	<code>fun updateBooks(newBooks: List<BookUi>) {</code>
73	<code>this.books.clear()</code>
74	<code>this.books.addAll(newBooks)</code>
75	<code>notifyDataSetChanged() // For production</code>
	<code>apps, consider using DiffUtil for better</code>
	<code>performance</code>
76	<code>}</code>
77	<code>}</code>

Tabel 5. 16. Source Code MyAdapter

17. BookViewModel

```
1 package
  com.example.myapi_test.presentation.viewmodel
2
3 import android.util.Log
4 import android.app.Application
5 import androidx.lifecycle.AndroidViewModel
6 import androidx.lifecycle.LiveData
7 import androidx.lifecycle.MutableLiveData
8 import androidx.lifecycle.ViewModel
9 import androidx.lifecycle.viewModelScope
10 import
  com.example.myapi_test.data.mappers.toUiModel
11 import
  com.example.myapi_test.data.repository.BookRepositoryImpl
12 import
  com.example.myapi_test.domain.usecase.GetBooksUseCase
13 import
  com.example.myapi_test.presentation.model.BookUi
14 import kotlinx.coroutines.launch
15
16 class BookViewModel(application: Application) :
  AndroidViewModel(application) {
17     // --- Dependencies ---
18     // In a real app, use Dependency Injection
  (e.g., Hilt) to provide these dependencies.
19     // We manually create the instances here for
  simplicity.
20     private val bookRepository =
  BookRepositoryImpl(application.applicationContext)
```

	t)
21	private val getBooksUseCase =
	GetBooksUseCase(bookRepository)
22	
23	// --- LiveData ---
24	private val _books =
	MutableLiveData<List<BookUi>>()
25	val booksLiveData: LiveData<List<BookUi>>
	get() = _books
26	
27	private val _isLoading =
	MutableLiveData<Boolean>()
28	val isLoadingLiveData: LiveData<Boolean>
	get() = _isLoading
29	
30	private val _errorMessage =
	MutableLiveData<String>()
31	val errorLiveData: LiveData<String> get() =
	_errorMessage
32	
33	private val TAG = "BookViewModel"
34	
35	/**
36	<i> * Fetches the list of books by executing</i>
	<i> the use case.</i>
37	<i> */</i>
38	fun fetchBooks() {
39	_isLoading.value = true
40	viewModelScope.launch {
41	// Execute the use case to get the
	result
42	val result = getBooksUseCase()

43	
44	<pre> // Handle the result: onSuccess or onFailure </pre>
45	<pre> result.onSuccess { domainBooks -> </pre>
46	<pre> // Map domain models to UI models before posting to LiveData </pre>
47	<pre> _books.postValue(domainBooks.map { it.toUiModel() }) </pre>
48	<pre> Log.d(TAG, "Successfully fetched \${domainBooks.size} books.") </pre>
49	<pre> }.onFailure { exception -> </pre>
50	<pre> _errorMessage.postValue("Failed to fetch books: \${exception.message}") </pre>
51	<pre> Log.e(TAG, "Error fetching books", exception) </pre>
52	<pre> } </pre>
53	<pre> // Update loading state regardless of outcome </pre>
54	<pre> _isLoading.postValue(false) </pre>
55	<pre> } </pre>
56	<pre> } </pre>
57	<pre> } </pre>

Tabel 5. 17. Source Code BookViewModel

18. BookViewModelFactory

1	<pre> package com.example.myapi_test.presentation.viewmodel </pre>
2	
3	<pre> import android.app.Application </pre>
4	<pre> import androidx.lifecycle.ViewModel </pre>
5	<pre> import androidx.lifecycle.ViewModelProvider </pre>
6	

7	<i>/**</i>
8	<i> * A factory class for creating BookViewModel</i> <i>instances.</i>
9	<i> * It's required because our BookViewModel has a</i> <i>constructor with parameters (Application).</i>
10	<i>*/</i>
11	class BookViewModelFactory(private val application: Application) : ViewModelProvider.Factory {
12	
13	<i>/**</i>
14	<i> * Creates a new instance of the given</i> <i>`Class`.</i>
15	<i> * @param modelClass a `Class` whose</i> <i>instance is requested</i>
16	<i> * @return a newly created ViewModel</i>
17	<i>*/</i>
18	override fun <T : ViewModel> create(modelClass: Class<T>): T {
19	// Check if the requested ViewModel class is our BookViewModel
20	if (modelClass.isAssignableFrom(BookViewModel::clas s.java)) {
21	// If it is, create and return an instance of it, passing the application context.
22	// The unchecked cast is safe because of the isAssignableFrom check.
23	@Suppress("UNCHECKED_CAST")
24	return BookViewModel(application) as
25	T }

26	// If it's a different ViewModel, throw an exception.
27	throw IllegalArgumentException("Unknown ViewModel class: \${modelClass.name}")
28	}
29	}

Tabel 5. 18. Source Code BookViewModelFactory

19. detail_fragment.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<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:layout_width="match_parent" android:layout_height="match_parent" tools:context=".presentation.ui.DetailFragment">
3	
4	
5	
6	
7	
8	
9	<!-- This Toolbar is specific to the DetailFragment for the back button and title -->
10	<androidx.appcompat.widget.Toolbar
11	android:id="@+id/detailToolbar"
12	android:layout_width="match_parent"
13	android:layout_height="?attr/actionBarSize"
14	android:background="?attr/colorPrimary"
15	android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
16	app:layout_constraintTop_toTopOf="parent"
17	app:titleTextColor="@android:color/white" />

18	
19	<pre> <!-- NestedScrollView makes the content below the toolbar scrollable --> <androidx.core.widget.NestedScrollView android:layout_width="0dp" android:layout_height="0dp" app:layout_constraintTop_toBottomOf="@id/detailT oolbar" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintEnd_toEndOf="parent"> <!-- A LinearLayout organizes the image and text vertically --> <LinearLayout android:layout_width="match_parent" android:layout_height="wrap_content" android:orientation="vertical"> <!-- The ImageView with a fixed height of 600dp as requested --> <ImageView android:id="@+id/detailImage" android:layout_width="match_parent" android:layout_height="600dp" android:scaleType="centerCrop" android:contentDescription="@string/book_cover_i mage_large" tools:src="@tools:sample/backgrounds/scenic" /> <!-- A container for the text content with padding for better readability --> </pre>
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	

44	<LinearLayout
45	android:layout_width="match_parent"
46	android:layout_height="wrap_content"
47	android:orientation="vertical"
48	android:padding="16dp">
49	
50	<TextView
51	android:id="@+id/detailTitle"
52	android:layout_width="match_parent"
53	android:layout_height="wrap_content"
54	android:layout_marginBottom="8dp"
55	android:textAppearance="@style/TextAppearance.Ma terialComponents.Headline5"
56	tools:text="Nama Buku" />
57	
58	<TextView
59	android:id="@+id/detailDescription"
60	android:layout_width="match_parent"
61	android:layout_height="wrap_content"
62	android:layout_marginTop="8dp"
63	android:lineSpacingMultiplier="1.2"
64	android:textAppearance="@style/TextAppearance.Ma terialComponents.Body1"
65	tools:text="Deskripsi lengkap." />
66	</LinearLayout>
67	
68	</LinearLayout>
69	
70	</androidx.core.widget.NestedScrollView>
71	

72	</androidx.constraintlayout.widget.ConstraintLayout>
----	------------------------------------------------------

Tabel 5. 19. Source Code detail_fragment.xml

20. activity_main.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:id="@+id/main_layout"
6	android:layout_width="match_parent"
7	android:layout_height="match_parent"
8	
9	android:background="?android:attr/colorBackground"
10	
11	tools:context=".presentation.ui.MainActivity">
12	
13	<FrameLayout
14	android:id="@+id/fragmentContainer"
15	android:layout_width="0dp"
16	android:layout_height="0dp"
17	app:layout_constraintTop_toTopOf="parent"
18	app:layout_constraintBottom_toBottomOf="parent"
19	app:layout_constraintStart_toStartOf="parent"
20	app:layout_constraintEnd_toEndOf="parent" />
21	
22	<ProgressBar

23	android:id="@+id/progressBar"
24	style="?android:attr/progressBarStyle"
25	android:layout_width="wrap_content"
26	android:layout_height="wrap_content"
27	android:visibility="gone"
28	app:layout_constraintTop_toTopOf="parent"
29	app:layout_constraintBottom_toBottomOf="parent"
30	app:layout_constraintStart_toStartOf="parent"
31	app:layout_constraintEnd_toEndOf="parent"
32	tools:visibility="visible" />
33	
34	</androidx.constraintlayout.widget.ConstraintLayout>

Tabel 5. 20. Source Code activity_main

21. home_fragment.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.constraintlayout.widget.ConstraintLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:layout_width="match_parent"
6	android:layout_height="match_parent"
7	tools:context=".presentation.ui.HomeFragment">
8	
9	<com.google.android.material.appbar.AppBarLayout
10	android:id="@+id/homeAppBarLayout"
11	android:layout_width="match_parent"
12	android:layout_height="wrap_content"

13	app:layout_constraintTop_toTopOf="parent"
14	app:layout_constraintStart_toStartOf="parent"
15	app:layout_constraintEnd_toEndOf="parent">
16	
17	<androidx.appcompat.widget.Toolbar
18	android:id="@+id/homeToolbar"
19	android:layout_width="match_parent"
20	android:layout_height="?attr/actionBarSize"
21	android:background="?attr/colorPrimary"
22	app:title="@string/app_name"
23	app:titleTextColor="@android:color/white"
24	app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
25	
26	</com.google.android.material.appbar.AppBarLayout>
27	
28	<androidx.recyclerview.widget.RecyclerView
29	android:id="@+id/rv_character"
30	android:layout_width="0dp"
31	android:layout_height="0dp"
32	android:padding="4dp"
33	android:clipToPadding="false"
34	app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
35	app:layout_constraintBottom_toBottomOf="parent"
36	app:layout_constraintEnd_toEndOf="parent"
37	app:layout_constraintStart_toStartOf="parent"
38	app:layout_constraintTop_toBottomOf="@id/homeAppBarLayout"
39	tools:listitem="@layout/item_list" />
40	

41	</androidx.constraintlayout.widget.ConstraintLayout>
----	------------------------------------------------------

Tabel 5. 21. Source Code home_fragment

22. item_list.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<androidx.cardview.widget.CardView
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	xmlns:tools="http://schemas.android.com/tools"
5	android:layout_width="match_parent"
6	android:layout_height="wrap_content"
7	android:layout_margin="8dp"
8	app:cardBackgroundColor="?attr/colorSurface"
9	app:cardCornerRadius="16dp"
10	app:cardElevation="4dp">
11	
12	<androidx.constraintlayout.widget.ConstraintLayout
	ut
13	android:layout_width="match_parent"
14	android:layout_height="wrap_content"
15	android:padding="16dp">
16	
17	<ImageView
18	android:id="@+id/imageView"
19	android:layout_width="100dp"
20	android:layout_height="150dp"
21	android:contentDescription="@string/book_cover_image"
22	android:scaleType="centerCrop"

23	app:layout_constraintStart_toStartOf="parent"
24	app:layout_constraintTop_toTopOf="parent"
25	tools:src="@tools:sample/avatars" />
26	
27	<TextView
28	android:id="@+id/textViewName"
29	android:layout_width="0dp"
30	android:layout_height="wrap_content"
31	android:layout_marginStart="16dp"
32	android:ellipsize="end"
33	android:maxLines="2"
34	android:textAppearance="@style/TextAppearance.MaterialComponents.Subtitle1"
35	android:textStyle="bold"
36	app:layout_constraintEnd_toEndOf="parent"
37	app:layout_constraintStart_toEndOf="@id/imageView"
38	app:layout_constraintTop_toTopOf="parent"
39	tools:text="Main Title" />
40	
41	<TextView
42	android:id="@+id/textViewAuthor"
43	android:layout_width="0dp"
44	android:layout_height="wrap_content"
45	android:layout_marginTop="4dp"
46	android:ellipsize="end"
47	android:maxLines="1"
48	android:textAppearance="@style/TextAppearance.MaterialComponents.Body2"
49	app:layout_constraintEnd_toEndOf="@id/textViewName"

50	app:layout_constraintStart_toStartOf="@id/textVi ewName"
51	app:layout_constraintTop_toBottomOf="@id/textVie wName"
52	tools:text="Author" />
53	
54	<TextView
55	android:id="@+id/textViewYear"
56	android:layout_width="0dp"
57	android:layout_height="wrap_content"
58	android:layout_marginTop="2dp"
59	android:textAppearance="@style/TextAppearance.Ma terialComponents.Caption"
60	app:layout_constraintEnd_toEndOf="@id/textViewNa me"
61	app:layout_constraintStart_toStartOf="@id/textVi ewName"
62	app:layout_constraintTop_toBottomOf="@id/textVie wAuthor"
63	tools:text="1990" />
64	
65	<LinearLayout
66	android:id="@+id/buttonRow"
67	android:layout_width="0dp"
68	android:layout_height="wrap_content"
69	android:layout_marginTop="8dp"
70	android:gravity="end"
71	android:orientation="horizontal"
72	app:layout_constraintBottom_toBottomOf="parent"
73	app:layout_constraintEnd_toEndOf="parent"
74	app:layout_constraintStart_toStartOf="@id/textVi ewName"

75	app:layout_constraintTop_toBottomOf="@id/textVie wYear"
76	app:layout_constraintVertical_bias="1.0">
77	
78	<com.google.android.material.button.MaterialButt on
79	android:id="@+id/buttonInfo"
80	style="@style/Widget.MaterialComponents.Button.T extButton"
81	android:layout_width="wrap_content"
82	android:layout_height="wrap_content"
83	android:layout_marginEnd="8dp"
84	android:text="@string/info_button_text"
85	android:textAllCaps="false" />
86	
87	<com.google.android.material.button.MaterialButt on
88	android:id="@+id/buttonDetail"
89	style="@style/Widget.MaterialComponents.Button"
90	android:layout_width="wrap_content"
91	android:layout_height="wrap_content"
92	android:text="@string/detail_button_text"
93	android:textAllCaps="false" />
94	</LinearLayout>
95	
96	</androidx.constraintlayout.widget.ConstraintLay out>
97	</androidx.cardview.widget.CardView>

Tabel 5. 22. Source Code item_list

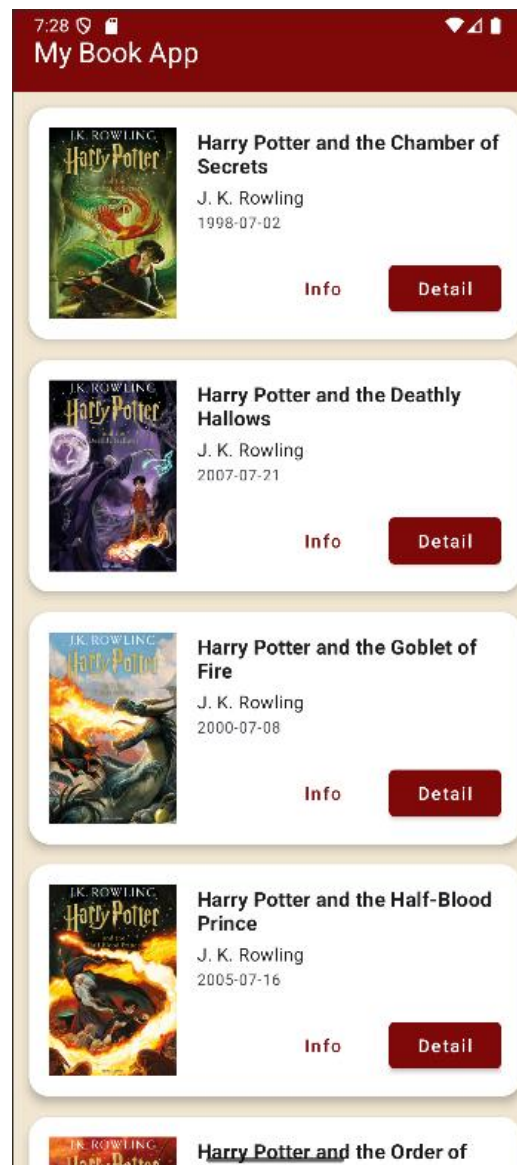
23. nav_graph.xml

1	<?xml version="1.0" encoding="utf-8"?>
2	<navigation
	xmlns:android="http://schemas.android.com/apk/res/android"
3	xmlns:app="http://schemas.android.com/apk/res-auto"
4	android:id="@+id/nav_graph"
5	app:startDestination="@id/HomeFragment">
6	
7	<fragment
8	android:id="@+id/HomeFragment"
9	android:name="com.example.myapi_test.presentation.ui.HomeFragment"
10	android:label="HomeFragment" >
11	<action
12	android:id="@+id/action_HomeFragment_to_detailFragment"
13	app:destination="@id/detailFragment"
	/>
14	</fragment>
15	
16	<fragment
17	android:id="@+id/detailFragment"
18	android:name="com.example.myapi_test.presentation.ui.DetailFragment"
19	android:label="DetailFragment" >
20	<argument
21	android:name="imageResId"
22	app:argType="integer" />
23	<argument
24	android:name="nama"

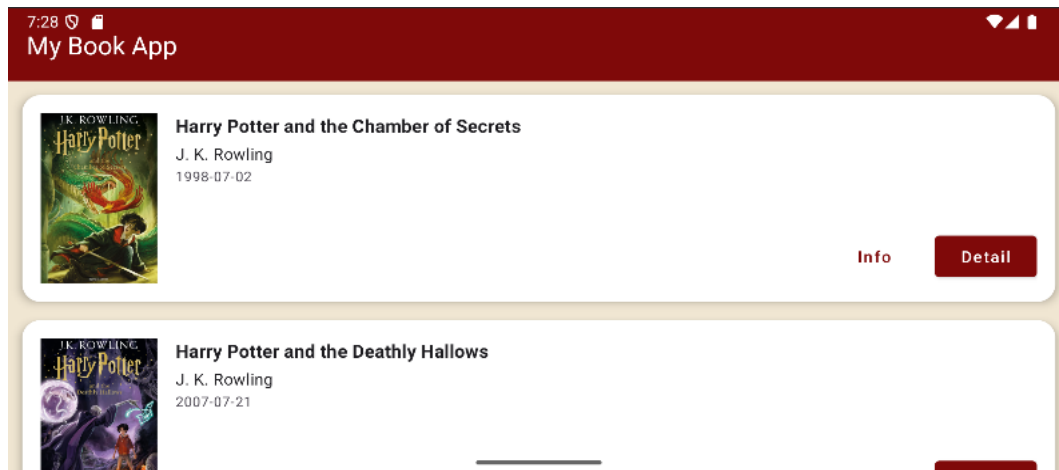
25	app:argType="string" />
26	<argument
27	android:name="deskripsi"
28	app:argType="string" />
29	</fragment>
30	
31	</navigation>

Tabel 5. 23. Source Code nav_graph

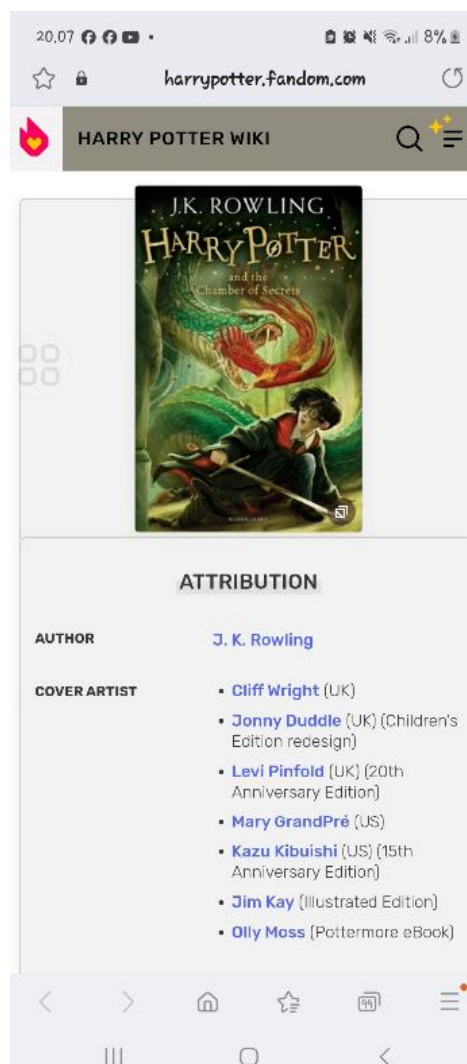
B. Output Program



Gambar 5. 1. Screenshot Hasil Jawaban Soal 1



Gambar 5. 2. Screenshot Hasil Jawaban Soal 1



Gambar 5. 3. Screenshot tombol Detail



Gambar 5. 4. creenshot tombol Info

C. Pembahasan

Jakak

1. MyApiResponse.kt

Pada file ini terutama bagian paket `com.example.myapi_test.data.api` berguna dalam struktur proyek Android agar bisa mengelompokkan class yang berhubungan saat kita melakukan pemanggilan API eksternal. Dimana file ini berfungsi sebagai Data Transfer Object (DTO), yaitu class-class yang mencerminkan

struktur data JSON yang diterima dari API untuk diolah dengan mudah dalam aplikasi Android.

Dimulai dari bagian pertama dari kode mendefinisikan class `BookApiResponse` sebagai class utama mewakili respons JSON dari API. Di dalamnya terdapat properti data yang berupa daftar (list) dari objek `BookData`. Nah, artinya respons JSON dari API berisi array data buku dan setiap item di dalam array tersebut nantinya dipetakan menjadi objek `BookData`. Kemudian, class `BookData` berguna untuk merepresentasikan setiap entri data buku satu persatu dalam daftar dengan class ini hanya mempunyai satu properti bernama `attributes` berisi objek dari tipe `BookAttributes`. Jadi, seluruh informasi buku (seperti judul, penulis, tanggal rilis, dll) disimpan di dalam objek `attributes`.

Selanjutnya, ada class `BookAttributes` berguna untuk menyimpan detail lengkap dari sebuah buku. Disini terdapat enam properti yang menjadi informasi buku yaitu, `author` (nama penulis), `cover` (URL gambar sampul), `release_date` (tanggal rilis), `title` (judul buku), `wiki` (tautan ke halaman wiki buku), dan `summary` (ringkasan isi buku). Dimana semua properti tipe datanya `String` sesuai dengan tipe data pada JSON API yang digunakan.

Kesimpulan dari file ini bertujuan untuk memetakan struktur JSON dari API menuju bagian objek Kotlin agar bisa digunakan lebih mudah dalam proses parsing dan manipulasi data di aplikasi Android yang biasanya dibantu dengan library seperti Retrofit atau Moshi untuk aplikasi saya buat ini.

2. MyInstance

Pada file ini terutama bagian paket `com.example.myapi_test.data.api` berguna untuk menginisialisasi dan menyediakan instance Retrofit nantinya saya gunakan saat melakukan permintaan ke API. Retrofit merupakan library

populer di Android yang menangani komunikasi HTTP dengan cara sederhana dan efisien. Di file ini juga berperan sebagai singleton object berarti hanya akan dibuat satu instance Retrofit yang digunakan secara bersama-sama di seluruh aplikasi.

Bagian pertama dari file ini kita perlu mendefinisikan object `MyInstance`. Dimana object adalah cara agar bisa mendefinisikan singleton, yaitu objek yang hanya memiliki satu instance. Di dalam objek ini, terdapat sebuah konstanta `BASE_URL` yang menyimpan alamat dasar dari API nantinya diakses yang mana di sini `https://api.potterdb.com/`. Dari alamat ini atau URL utama nantinya menjadi dasar semua endpoint API yang digunakan aplikasi ini.

Kemudian, ada properti api bertipe `MyService` yang dideklarasikan dengan kata kunci `val` dan menggunakan `by lazy`. Nah, `by lazy` berarti instance ini hanya akan dibuat satu kali saat pertama kali kita gunakan. Hal ini untuk efisiensi memori dan performa karena Retrofit tidak dibuat sebelum benar-benar dibutuhkan. Dalam blok `lazy` ini dibuatlah instance Retrofit menggunakan `Retrofit.Builder()`. Dengan bagian pertamanya itu ada `baseUrl(BASE_URL)` agar menetapkan URL dasar untuk semua permintaan API. Lalu, `addConverterFactory(GsonConverterFactory.create())` digunakan agar Retrofit bisa mengubah (convert) respons JSON dari API menjadi objek Kotlin menggunakan library `Gson`. Nah, setelah konfigurasi selesai bagian `build()` nantinya membangun instance Retrofit dan ada `create(MyService::class.java)` berfungsi saat membuat implementasi interface `MyService` berisi definisi endpoint-endpoint dari API.

Kesimpulan dari file ini untuk menyediakan cara yang paling rapi dan lebih efisien membuat dan mengakses instance Retrofit di seluruh aplikasi, sehingga kita sebagai pengembang aplikasi tidak perlu mengatur ulang koneksi API di setiap tempat yang berbeda.

3. MyService

Pada file ini terutama bagian paket `com.example.myapi_test.data.api` berfungsi sebagai interface Retrofit yang mendefinisikan endpoint dari API ingin digunakan dalam aplikasi di buat. Dengan Retrofit yang menggunakan antarmuka seperti ini agar bisa memetakan HTTP request menjadi fungsi Kotlin, sehingga memudahkan komunikasi antara aplikasi dan server.

Kemudian, disini ada import terhadap anotasi `@GET` dari library Retrofit. Dari anotasi ini digunakan untuk menunjukkan bahwa fungsi di bawahnya akan melakukan permintaan HTTP GET menuju endpoint tertentu di API.

Lalu, didefinisikan sebuah interface bernama `MyService`. Dimana interface ini bertindak sebagai kontrak berisi deklarasi fungsi-fungsi mewakili permintaan dari HTTP. Di dalamnya terdapat satu fungsi yaitu `getMessage()` nantinya mengakses endpoint `"v1/books"` dari API. Nah, fungsi ini diberi anotasi `@GET("v1/books")` artinya saat fungsi ini dipanggil dengan Retrofit nantinya melakukan HTTP GET ke URL `https://api.potterdb.com/v1/books` (mengacu `BASE_URL` dari `MyInstance.kt`).

Selanjutnya, ada fungsi `getMessage()` ditandai dengan kata kunci `suspend` yang menunjukkan bahwa ini itu adalah `suspend function` dan harus dipanggil dari dalam `coroutine`. Hal ini menjadi bagian untuk pemrograman asinkron (non-blokir), sehingga operasi jaringan bisa dilakukan tanpa perlu membekukan UI aplikasi. Dengan fungsi ini akan mengembalikan objek `BookApiResponse` yaitu, model data yang telah didefinisikan sebelumnya untuk mencerminkan struktur JSON dari respons API.

Kesimpulan dari file `MyService.kt` bertugas menjembatani antara Retrofit dengan endpoint yang tersedia di API eksternal. Dengan hanya menuliskan fungsi yang disertai anotasi HTTP, maka saya sebagai

pengembang aplikasi bisa melakukan permintaan API tanpa harus menulis kode jaringan secara manual.

4. BookDao

Pada file ini terutama bagian paket ada `com.example.myapi_test.data.database.dao` merupakan bagian dari struktur arsitektur aplikasi Android berguna untuk menangani akses menuju database lokal menggunakan Room. Dengan file inilah bisa mendefinisikan sebuah Data Access Object (DAO) yang digunakan oleh Room agar bisa mengakses data ke dalam database secara efisien dan lebih aman.

Bagian pertama dari kode mengimpor anotasi dan class-class penting dari Room seperti `@Dao`, `@Insert`, `@Query`, dan `OnConflictStrategy`, serta untuk mengimpor `BookDbEntity`, yaitu entitas (struktur tabel) yang merepresentasikan data buku di dalam database. Kemudian, didefinisikan sebuah interface `BookDao` sebagai tanda anotasi `@Dao` yang menunjukkan bahwa interface ini nantinya digunakan Room untuk meng-generate implementasi kode database secara otomatis. Juga, interface ini berisi tiga fungsi utama yang menangani operasi dasar dari tabel buku di database.

Selanjutnya, fungsi ada `insertBooks()` berguna untuk menyisipkan daftar buku menuju dalam database. Dimana fungsi ini menggunakan anotasi `@Insert` dengan `OnConflictStrategy.REPLACE` artinya jika ada entri dengan primary key yang sama, maka data lama nantinya digantikan dengan yang baru. Nah, fungsi ini bersifat suspend, sehingga dijalankan dalam coroutine agar tidak bisa memblokir thread utama.

Kemudian, ada fungsi `getAllBooks()` yang diberi anotasi `@Query("SELECT * FROM books ORDER BY title ASC")`. Nah, fungsi ini digunakan untuk mengambil semua data buku dari tabel `books` dan mengurutkannya sesuai judul urutannya dari A sampai Z.

Hasilnya muncul daftar dari objek `BookDbEntity`. Karena ini juga suspend, maka pemanggilannya harus berada dalam coroutine.

Berikutnya, ada fungsi ketiga `clearAllBooks()` menggunakan anotasi `@Query` untuk menjalankan perintah `SQL DELETE FROM books`. Hal ini nantinya menghapus seluruh isi tabel `books` di database. Oleh karena itu, fungsi ini berguna misalnya saat saya ingin mererefresh data dengan yang baru dari API.

Kesimpulan dari file `BookDao.kt` merupakan komponen penting dalam sistem database lokal menggunakan Room yang menyediakan antarmuka untuk melakukan penyimpanan, pembacaan, dan penghapusan data buku secara terstruktur dan aman di dalam aplikasi ini.

5. BookDbEntity

Pada file ini terutama bagian paket `com.example.myapi_test.data.database.entity` berguna untuk mendefinisikan struktur tabel dalam database lokal yang digunakan oleh library Room di Android. Dimana file ini adalah entitas database yang mewakili setiap baris (record) dalam tabel `books`. Nah, Room akan menggunakan class data seperti ini agar bisa memetakan data antara database SQLite dan objek Kotlin secara otomatis.

Dengan class ini diberi anotasi `@Entity(tableName = "books")` artinya Room nantinya membuat atau menggunakan tabel bernama `books` di database dan setiap objek dari class ini nantinya direpresentasikan sebagai satu baris dalam tabel tersebut. Nama tabel bisa disesuaikan dan dalam project aplikasi ini disesuaikan menjadi `"books"` agar sesuai dengan konteks data yang ingin disimpan.

Kemudian, nama class `BookDbEntity` menggunakan bentuk data (data class) karena class ini hanya bertugas menyimpan data dan menyediakan fungsi-fungsi dasar seperti `equals()`, `hashCode()`, dan `toString()` yang otomatis dibuat oleh Kotlin. Di dalam class ini terdapat enam property seperti, `title`, `author`, `cover`, `releaseDate`,

summary, dan wiki yang seluruhnya bertipe data `String`. Berikutnya, properti `title` diberi anotasi `@PrimaryKey` yang menjadikannya kunci utama (primary key) dari tabel `books`. Hal ini berarti nilai `title` harus unik di setiap baris tabel. Dengan saya tambahkan komentar dalam kode ini juga menjelaskan bahwa penggunaan judul sebagai primary key hanya untuk keperluan sederhana atau percobaan dalam aplikasi nyata, biasanya digunakan ID unik yang berasal dari API agar lebih bagus dan terhindar dari duplikasi data.

Kesimpulan dari file `BookDbEntity.kt` adalah representasi dari satu entri buku dalam database Room dan menjadi jembatan antara data lokal dan logika aplikasi ini. Kita lihat lagi dari struktur ini memungkinkan developer bisa menyimpan data hasil dari API (di sini buku dari API Harry Potter) secara lokal, sehingga dapat diakses walaupun dalam kondisi offline.

6. AppDatabase

Pada file ini terutama bagian paket ada `com.example.myapi_test.data.database` berfungsi untuk mendefinisikan database utama menggunakan Room karena library ini dari Android Jetpack yang menyediakan abstraksi di atas `SQLite`. Dimana file ini bagian class database utama tempat seluruh entitas dan DAO dihubungkan, sehingga Room bisa membangun dan mengelola database lokal di aplikasi tersebut.

Kemudian, bagian pertama dari anotasi `@Database` menyatakan bahwa class `AppDatabase` merupakan sebuah database dengan di dalam anotasi tersebut terdapat parameter `entities = [BookDbEntity::class]` berarti database ini mempunyai satu tabel, yaitu tabel `books` yang diwakili oleh class `BookDbEntity`. `version = 1` yang menunjukkan versi pertama dari database ini dan `exportSchema = false` artinya Room nantinya tidak bisa mengekspor

skema database ke file metadata. Dari fitur inilah biasanya digunakan untuk dokumentasi atau pengujian migrasi.

Selanjutnya, ada class `AppDatabase` adalah abstract class yang mewarisi dari `RoomDatabase`. Karena abstrak, maka kita tidak membuat instance-nya secara langsung, tetapi Room nantinya yang mengelolanya. Di dalam class ini terdapat fungsi abstrak `bookDao()` yang mengembalikan objek `BookDao`. Dari fungsi ini adalah titik masuk utama untuk mengakses data buku melalui operasi yang telah didefinisikan di `BookDao.kt`. Bagian berikutnya ada companion object berguna untuk mengelola instance tunggal (singleton) dari database yang di dalamnya terdapat properti `INSTANCE` bersifat `@Volatile` berarti nilainya akan selalu diperbarui dan terlihat oleh semua thread. Hal ini penting agar bisa menghindari pembuatan banyak instance database dalam aplikasi yang sama dapat menyebabkan crash.

Lalu, ada fungsi `getDatabase(context: Context): AppDatabase` berguna untuk mendapatkan instance database. Apabila `INSTANCE` belum dibuat, maka fungsi ini nanti yang membuatnya menggunakan `Room.databaseBuilder(...)`. Dimana pembuatan instance ini dibungkus dalam `synchronized(this)` agar aman terhadap akses dari banyak thread secara bersamaan. Terdapat konfigurasi yang diberikan meliputi `context.applicationContext` referensi class `AppDatabase` dan nama database `"book_database"`. Selain itu, `fallbackToDestructiveMigration()` digunakan sebagai strategi migrasi versi database yang mana database lama nanti dihapus dan dibuat ulang jika versi berubah dengan biasanya hanya dipergunakan saat adanya pengembangan karena akan menghapus data lama.

Kesimpulan dari file `AppDatabase.kt` merupakan pondasi utama agar bisa pengelolaan database lokal menggunakan Room yang menghubungkan entitas data (`BookDbEntity`) dengan akses datanya (`BookDao`) dan menyediakan cara yang lebih aman serta efisien untuk

membangun instance database dalam aplikasi Android yang sedang kita buat.

7. BookMapper.kt

Pada file ini terutama bagian paket ada `com.example.myapi_test.data.mappers` berperan penting dalam bagaimana cara pemrosesan dan konversi data antar layer (data layer, domain layer, dan presentation layer) di bagian arsitektur aplikasi Android yang terstruktur. Ada fungsi-fungsi di file ini dikenal sebagai mapper yang bertugas mengubah satu jenis objek data menjadi jenis objek lain yang lebih sesuai dengan konteks penggunaannya (misalnya dari format API ke model domain, atau dari database ke tampilan UI).

Kemudian, bagian pertama dari file ini ada fungsi `BookData.toDomain()`. Dimana fungsi ini berguna untuk mengubah objek `BookData` karena hasil deserialisasi dari API (DTO atau *Data Transfer Object*) menjadi objek `Book` yang merupakan gambaran dari domain layer. Di sini pula ada konversi karena penting agar domain layer tidak selalu bergantung langsung pada struktur data dari API dan bisa bekerja dengan bentuk data lebih stabil.

Selanjutnya, terdapat fungsi `Book.toUiModel()` bertugas untuk mengubah objek `Book` dari domain layer menjadi `BookUi`, yaitu model yang digunakan oleh presentation layer (biasanya digunakan oleh `ViewModel` atau langsung oleh UI). Hal ini memungkinkan pemisahan antara logika aplikasi (domain) dan tampilan (UI), sehingga setiap lapisan bisa berubah tanpa mempengaruhi yang lain secara langsung.

Berikutnya, ada fungsi `BookAttributes.toDbEntity()` berguna saat ingin mengubah `BookAttributes` yang merupakan bagian dari respons API menjadi `BookDbEntity`, yaitu entitas untuk disimpan menuju database Room. Dimana fungsi ini penting dalam proses caching data dari API ke database lokal agar bisa digunakan kembali tanpa perlu koneksi internet lagi.

Lalu, fungsi `BookDbEntity.toDomain()` berguna untuk mengubah data dari database (`BookDbEntity`) menjadi objek domain (`Book`). Dimana fungsi ini digunakan saat kita ingin mengambil data dari database dan mau memprosesnya di dalam domain layer tanpa perlu melibatkan struktur data database secara langsung.

Kesimpulan dari file `BookMapper.kt` berfungsi sebagai jembatan untuk konversi data antar lapisan aplikasi sampai ke data dari API, database, dan domain dipisahkan dengan jelas. Yang mana saat kita praktikkan di aplikasi ini mendukung prinsip *separation of concerns* dan meningkatkan fleksibilitas, kemudahan dalam membaca kodenya, serta *maintainability* dari kode aplikasi secara keseluruhan.

8. BookRepositoryImpl

Pada file ini terutama bagian paket ada `com.example.myapi_test.data.repository` merupakan implementasi dari interface `BookRepository` berguna saat didefinisikan dalam domain layer. Dimana `BookRepositoryImpl` bertugas untuk mengelola pengambilan data buku dari dua sumber remote (API) dan lokal (database Room) bertujuan agar bisa menyediakan data yang konsisten untuk lapisan-lapisan lain dalam arsitektur aplikasi, seperti `ViewModel` dan `UI`. Jadi, di file ini pula sebenarnya

Di file ini pula untuk mengimplementasikan sebuah pola arsitektur dalam pengembangan aplikasi Android, yaitu "Single Source of Truth" bertujuan agar bisa membuat aplikasi tetap fungsional baik saat online maupun offline dengan mengandalkan database lokal sebagai sumber data utama bagi `UI` aplikasi ini.

Lalu, ada deklarasi class dan inisialisasi sumber data di baris [19] – [22] dan [57] adalah fondasi dari repositori. Karena class `BookRepositoryImpl` dibuat untuk menjalankan fungsi yang didefinisikan dalam interface `BookRepository`. Dari class ini mengelola dua sumber data `apiService` yang digunakan agar mengambil data dari

internet melalui `Retrofit` (network), dan `bookDao` yang digunakan untuk berinteraksi dengan database lokal (`Room`). Adanya inisialisasi keduanya di sini, maka repositori siap menjadi perantara antara sumber data remote dan lokal.

Selanjutnya, ada fungsi utama `getBooks()` dan penggunaan coroutine di baris [24] – [56] karena fungsi utama nantinya dipanggil oleh bagian lain dari aplikasi (misalnya, `ViewModel`) agar bisa mendapatkan daftar buku. Dimana fungsi ini ditandai sebagai `suspend` berarti ia adalah fungsi coroutine yang dapat berjalan di latar belakang tanpa memblokir interface pengguna (UI). Dengan penggunaan `withContext(Dispatchers.IO)` secara eksplisit memindahkan semua operasi di dalamnya (seperti panggilan API dan akses database) ke *thread* IO yang memang dirancang untuk tugas-tugas berat tersebut, sehingga aplikasi tetap masih responsif.

Kemudian, di baris [26] – [43] sinkronisasi data dengan mengambil dari API dan menyimpan ke database yang mana inti dari blok `try` ini adalah proses sinkronisasi. Dimulai aplikasi pertama-tama mencoba mengambil data buku terbaru dari API. Jika berhasil nantinya membersihkan semua data lama yang ada di database lokal (`clearAllBooks`) untuk memastikan tidak ada data terduplikat. Setelah itu, data dari API diubah formatnya (`map`) agar sesuai struktur tabel database (`toDbEntity`) dan lanjutannya itu dimasukkan menuju database lokal. Adanya proses ini memastikan bahwa database lokal selalu berisi data paling terbaru jika koneksi internet sudah tersedia.

Berikutnya, di baris [39] – [43] untuk penanganan `Network Failure` karena blok `catch` ini adalah mekanisme penanganan kesalahan (error handling) jika terjadi masalah saat mengambil data dari API, misalnya karena tidak ada koneksi internet. Alih-alih membuat aplikasi crash, maka kode ini hanya akan mencatat error tersebut (`Log.e`). Hal terpenting, program tidak berhenti di sini, melainkan nanti melanjutkan lagi eksekusi

ke langkah berikutnya. Ini memungkinkan aplikasi tetap berfungsi dengan menggunakan data yang sudah ada di database (*cache*).

Setelah itu, di baris [48] – [55] bagian prinsip "Single Source of Truth" selalu mengembalikan data dari database karena terlepas dari apakah pengambilan data dari API berhasil ataupun gagal, fungsi ini nantinya selalu mengambil data dari database lokal (`bookDao.getAllBooks()`). Apabila API berhasil, maka data diambil adalah data baru yang baru saja disimpan. Jika API gagal, maka data diambil adalah data lama yang tersimpan sebelumnya (*cache*). Nah, data dari database ini kemudian diubah formatnya (`toDomain`) menjadi model data yang akan digunakan oleh lapisan UI dengan dibungkus dalam objek `Result.success` sebagai penanda keberhasilan operasi secara keseluruhan. Jika bahkan membaca dari database pun gagal, ia akan mengembalikannya ke `Result.failure`.

9. Book

Pada file ini terutama bagian paket ada `com.example.myapi_test.domain.model` berguna untuk mendefinisikan class data `Book` yang berfungsi sebagai model inti (*domain entity*) dalam arsitektur aplikasi. Maksud dari konteks arsitektur bersih (*clean architecture*) file ini mewakili lapisan domain, yaitu pusat dari logika bisnis aplikasi. Dengan model ini tidak memiliki ketergantungan terhadap sumber data (seperti API atau database) maupun terhadap elemen UI berarti model ini bersifat murni dan dapat digunakan di mana saja dalam aplikasi.

Kemudian, di baris [7] – [14] untuk deklarasi data class `Book` berguna saat mendeklarasikan sebuah class `Book` menggunakan kata kunci data class. Dimana penggunaan data class di sini sudah pilihan yang tepat karena tujuannya hanya untuk menampung data. Dengan data class, maka kotlin secara otomatis membuatkan fungsi-fungsi standar yang sangat berguna di belakang layar, seperti `toString()`, `equals()`,

`hashCode()`, dan `copy()`. Ini membuat kode lebih ringkas dan bebas dari *boilerplate*. Intinya, baris ini mendefinisikan `Book` sebagai sebuah objek model data yang sederhana dan efisien.

Selanjutnya, di baris [8] – [13] bagian properti atau atribut yang mendefinisikan sebuah `Book`. Di setiap properti dideklarasikan dengan `val` berarti nilainya tidak bisa diubah setelah objek `Book` dibuat (*immutable*). Nah, ini membuat data lebih aman dan mudah diprediksi. Dengan setiap atribut (`title`, `author`, `cover`, dll.) yang mempunyai tipe data `String` menunjukkan bahwa semua itu nantinya menyimpan informasi dalam bentuk teks. Jadi, bagian ini menetapkan bahwa setiap objek `Book` dalam aplikasi yang saya buat ini akan selalu menampilkan judul, penulis, sampul, tanggal rilis, ringkasan, dan tautan wiki.

10. BookRepository

Pada file ini terutama bagian paket `com.example.myapi_test.domain.repository` berguna untuk mendefinisikan interface `BookRepository` karena dalam arsitektur aplikasi, interface ini bertindak sebagai jembatan antara lapisan domain (*domain layer*, tempat logika bisnis) dan lapisan data (*data layer*, tempat sumber data seperti API atau database). Dimana tujuannya untuk memisahkan logika bisnis dari detail teknis pengambilan data. Dengan lapisan domain hanya peduli apa yang bisa dilakukan (yaitu mendapatkan buku), bukan bagaimana caranya (apakah dari internet atau dari cache lokal). Ini membuat kode lebih bersih dan fleksibel.

Selanjutnya, di baris [15] terdapat `suspend fun getBooks()` : `Result<List<Book>>` merupakan satu-satunya aturan atau fungsi yang didefinisikan dalam `BookRepository`. Lebih tepatnya di bagian `suspend fun` sebagai penanda bahwa `getBooks` itu fungsi *asynchronous* yang dapat dijeda dan dilanjutkan, dirancang untuk berjalan di latar belakang (menggunakan *coroutines*) agar tidak mengganggu antarmuka pengguna (UI) dengan `getBooks()` menjadi nama fungsi yang jelas agar mendapatkan daftar buku melibatkan :

`Result<List<Book>>` adalah tipe data yang akan dikembalikan karena `Result` ini pembungkus (wrapper) yang sangat berguna karena bisa berisi salah satu dari dua kemungkinan yaitu, sebuah daftar objek `Book` jika berhasil (`Success`) ataupun sebuah `Exception` jika terjadi kegagalan (`Failure`). Jadi, ini cara yang aman dan lebih modern dalam menangani hasil operasi yang bisa saja gagal.

11. GetBooksUseCase

Pada file ini terutama bagian paket `com.example.myapi_test.domain.usecase` berguna untuk mendefinisikan sebuah class `GetBooksUseCase` merupakan bagian dari lapisan domain. Nah, class ini menjadi representasi dari satu kasus penggunaan atau logika bisnis tunggal yang ada dalam aplikasi kita buat. Lalu, perhatikan bahwa class ini membutuhkan `BookRepository` dalam konstruktornya. Hal ini menjadi poin kunci dimana `GetBooksUseCase` bergantung dengan kontrak (`BookRepository` interface), bukan hanya di implementasi detailnya (`BookRepositoryImpl`). Disini membuat `UseCase` sangat fleksibel dan mudah diuji secara terpisah.

Selanjutnya, di baris [16] – [18] ada suspend operator `fun invoke() : Result<List<Book>>` merupakan inti dari `UseCase` dengan penggunaan suspend operator `fun invoke` bagian trik khusus. Disini ada operator `fun invoke` berguna dalam memberi kemungkinan bahwa objek dari class `GetBooksUseCase` untuk dipanggil seolah-olah itu sebuah fungsi. Jadi, daripada menulis `getBooksUseCase.execute()`, saya bisa langsung menulis `getBooksUseCase()` dan membuat pemanggilan dari `ViewModel` menjadi lebih bersih dan ringkas menggunakan suspend sebagai penanda operasi ini bersifat asinkron juga harus dijalankan di dalam sebuah `Coroutine`, sehingga tidak akan memblokir UI. Nah, isi Fungsi menjadi satu-satunya hal yang perlu dilakukan oleh `return bookRepository.getBooks()` yang mana hanya meneruskan

perintah menuju repositori. Tugasnya bukan untuk mengetahui bagaimana cara mendapatkan buku, tetapi hanya untuk meminta buku dari repositori yang telah diberikan API.

12. BookUi

Pada file ini terutama bagian paket `com.example.myapi_test.presentation.model` berguna saat ingin mendefinisikan class data `BookUi`. Dimana class ini merupakan model data khusus untuk lapisan presentasi (UI) berarti data ini sudah dipersiapkan dalam format yang sudah siap ditampilkan ke layar user, biasanya hasil transformasi dari domain model menggunakan mapper. Dengan adanya model khusus untuk UI seperti ini, kita dapat menjaga agar interface ke user tidak langsung bergantung dengan detail dari data atau domain layer.

Selanjutnya, di baris [10] – [18] adalah bagian terpenting dari kode ini karena ada anotasi `@Parcelize` dan implementasi : `Parcelable`. Dimana dalam Android, jika saya ingin mengirim objek kustom (seperti `BookUi`) dari satu layar (`Activity/Fragment`) menuju layar lainnya dengan objek tersebut harus "`parcelable`". Lebih jelasnya : `Parcelable` merupakan interface Android yang menandakan bahwa objek dari kelas ini dapat diubah menjadi format yang bisa dikirim dan diterima antar komponen melibatkan `@Parcelize` adalah jalan pintas canggih dari bahasa `Kotlin` karena disini saya harus menulis banyak kode *boilerplate* yang rumit agar bisa mengimplementasikan `Parcelable` secara manual. Adanya `@Parcelize` membuat `Kotlin` secara otomatis membuat semua kode tersebut untuk kita di belakang layar dan membuat kode jauh lebih *clean* yang memudahkan pengelolaan data kompleks dalam navigasi antar layar.

Berikutnya, di baris [12] – [17] perlu mendeklarasikan sebuah data class `BookUi` yang fungsinya adalah sebagai wadah data. Dimana penggunaan nama `BookUi` agar bisa membedakannya dari model `Book` di lapisan domain. Dari class ini secara spesifik disiapkan untuk lapisan

presentasi (UI) berarti data di dalamnya mungkin sudah diformat dan ditampilkan. Properti di dalamnya (enam properti utama bertipe String yaitu, `author`, `cover`, `release_date`, `title`, `wiki`, dan `summary`) menjadi atribut-atribut nantinya saya tampilkan langsung di layar kepada user.

13. DetailFragment

Pada file terdapat class `DetailFragment` bagian sebuah komponen UI dalam arsitektur Android bertanggung jawab agar bisa menampilkan detail dari satu buku. Dimana class ini turunan dari `Fragment` berarti bagian dari interface user yang dapat digabungkan ke dalam `Activity` dengan fungsinya untuk menampilkan informasi mendetail berdasarkan data `BookUi` yang dikirimkan dari layar sebelumnya.

Selanjutnya, di file ini juga ada `binding` dan `currentBook` dengan `Fragment` ini menggunakan `ViewBinding` melalui objek `_binding` nantinya properti `binding` yang diinisialisasi di `onCreateView()` dan dibersihkan pada `onDestroyView()` menghindari memory leak. Dari variabel `currentBook` bertipe `BookUi` akan menyimpan data buku yang dipilih. Dimana data ini diambil dari argumen `Bundle` di `onCreate()` menggunakan metode `getParcelable()` dengan pengecekan versi Android agar kompatibel dengan API terbaru dan lama. Lalu, `onCreateView()` ini bertanggung jawab untuk meng-inflate layout dari `DetailFragmentBinding`, yaitu tampilan XML yang dikaitkan `fragment` ini. Dan view ini kemudian dikembalikan agar bisa ditampilkan ke user.

Kemudian, ada `onViewCreated()` membantu dalam mengatur tampilan setelah layout berhasil dibuat. Di sini peran toolbar nantinya dikonfigurasi agar bisa menampilkan tombol "Back" dengan memanggil `activity.supportActionBar()`. Dari tombol ini dikaitkan dengan `activity.onBackPressed()` untuk kembali ke layar

sebelumnya. Lanjutannya itu jika `currentBook` tidak null, fragment akan menampilkan data buku menuju tampilan seperti judul, deskripsi, dan gambar sampul. Apabila URL sampul tidak kosong, maka gambar nantinya dimuat menggunakan library `Glide` jika kosong, digunakan gambar default dari resource lokal API. Berikutnya, `onDestroyView()` membersihkan `_binding` agar referensi terhadap view dilepas ketika fragment dihancurkan. Hal ini terjadi di praktik aplikasi saya buat karena penting dalam pengelolaan lifecycle fragment untuk mencegah memory leak.

Juga, di file ini ada `companion object` menyediakan fungsi `newInstance()` yang digunakan untuk membuat instance baru dari `DetailFragment` dengan menyisipkan objek `BookUi` menuju `Bundle` sebagai `parcelable`. Nah, konstanta `ARG_BOOK` digunakan sebagai key untuk menyimpan dan mengambil data buku dalam bundle ini.

Kesimpulan di file `DetailFragment` adalah implementasi fragment yang mengikuti praktik Android modern termasuk penggunaan `ViewBinding`, `Parcelable`, `Fragment Arguments`, dan `Glide`. Dan tujuan dari file ini agar bisa menampilkan detail buku secara dinamis dan aman sambil menjaga struktur kode tetap masih modular, serta terpisah dari logika bisnis dan data (*clean architecture*).

14. HomeFragment

Pada class `HomeFragment` merupakan fragment yang bertanggung jawab agar bisa menampilkan daftar buku di halaman utama aplikasi ini. Dimana fragment ini menghubungkan `ViewModel` (`BookViewModel`) dengan UI melalui `ViewBinding` dan `RecyclerView`. Dari struktur fragment ini mengikuti arsitektur MVVM (`Model-View-ViewModel`) yang memisahkan logika data dan tampilan.

Kemudian, ada properti `_binding` adalah objek nullable dari `HomeFragmentBinding` yang digunakan untuk mengakses komponen

UI secara efisien. Dimana binding non-nullable menjamin akses hanya ketika binding sudah dibuat (selama fragment aktif). Nah, binding diinisialisasi di `onCreateView()` dan dibersihkan di `onDestroyView()` untuk mencegah memory leak pada project sudah dibuat.

Selanjutnya, bagian `onViewCreated()` karena `ViewModel` diinisialisasi menggunakan `ViewModelProvider` yang terikat dengan `requireActivity()`. Hal ini memungkinkan `ViewModel` dibagikan antara fragment dan activity induk agar menjaga konsistensi data. Peran `ViewModel` disini untuk mengamati data buku yang dikemas dalam `LiveData`. Lalu, fungsi `setupRecyclerView()` saat mengatur tampilan daftar buku menggunakan `LinearLayoutManager` dan `MyAdapter`. Dari adapter ini menerima tiga lambda untuk menangani klik pada tombol detail, tombol info (Wikipedia), dan klik di item root semua diarahkan menuju fungsi navigasi atau pembuka link. Berikutnya, ada fungsi `observeViewModel()` mengamati perubahan di `booksLiveData`. Apabila terdapat data baru, maka daftar buku di adapter diperbarui. Observasi ini menjaga agar UI tetap sinkron dengan adanya perubahan data di `ViewModel` secara reaktif.

Setelah itu, perlu melakukan pengambilan data awal karena masih di `onViewCreated()` dilakukan pengecekan apakah daftar buku masih kosong. Jika iya, maka `bookViewModel.fetchBooks()` dipanggil agar bisa memuat data buku dari repository, sehingga menghindari pemanggilan ulang saat konfigurasi ulang fragment (misalnya saat rotasi layar). Juga, ada fungsi `navigateToDetail()` membantu saat ingin menangani perpindahan menuju layar detail ketika item buku kita klik. Menggunakan

`parentFragmentManager.beginTransaction()`, maka fragment diganti ke `DetailFragment` yang dikirim argumen `BookUi` dan transaksi ditambahkan ke back stack agar user bisa kembali

dengan cara klik tombol back. Serta, ada fungsi `openWikiLink()` sebagai percobaan saat membuka URL dari properti `wiki` buku dengan `Intent`. Apabila URL valid, nantinya terbuka di browser. Bila tidak valid atau gagal, maka aplikasi dibuat itu menampilkan pesan kesalahan melalui `Toast`. Nah, hal ini menunjukkan implementasi yang defensif terhadap kemungkinan URL kosong atau format salah. Terakhir, ada fungsi `onDestroyView()` bertugas mengosongkan `_binding` agar tidak terjadi memory leak setelah `view fragment` dihancurkan. Dalam praktik dari project aplikasi yang saya buat ini sangat penting agar `fragment` tidak menyimpan referensi terhadap `view` yang sudah tidak diperlukan.

Kesimpulan dari file `HomeFragment` adalah bagian komponen presentasi utama yang mengelola tampilan daftar buku, merekam semisal ada perubahan data melalui `ViewModel`, dan menyediakan interaksi user seperti navigasi ke detail dan membuka info tambahan. Dengan melakukan implementasi di project ini sudah mencerminkan arsitektur `MVVM` yang baik dan menjaga *clean code* dengan pemisahan tanggung jawab antara logika UI dan data.

15. MainActivity

Pada file `MainActivity` berguna sebagai titik utama aplikasi Android karena aktivitas ini bertanggung jawab dalam mengatur tampilan awal, menginisialisasi `ViewModel` yang kita punya di project aplikasi, serta memuat `fragment` utama (`HomeFragment`). Dengan aktivitas ini juga membantu saya dalam menangani `splash screen` dan memantau status loading serta error.

Kemudian, di file ini ada bagian `onCreate()`, `installSplashScreen()` dipanggil sebelum `super.onCreate()` agar bisa memastikan *splash screen* compatible dengan Android 12+ ditampilkan saat kita klik aplikasi dan muncul *splash screen* saat dimulai aplikasinya. Nah, *splash screen* dijaga tetap terlihat melalui

`setKeepOnScreenCondition` nantinya tetap bisa menampilkan layar *splash* selama `isLoadingLiveData` bernilai `true`, yaitu saat data awal masih dimuat oleh `ViewModel`. Lalu, ada `ActivityMainBinding` berguna untuk menghubungkan kode bahas Kotlin sudah dibuat dengan layout XML (`activity_main.xml`) dengan cara perlu memanggil `ActivityMainBinding.inflate(layoutInflater)` dan memberikan `setContentView(binding.root)` yang mana semua elemen UI dalam layout bisa diakses melalui properti binding yang membuat kode lebih aman dan bebas dari adanya kesalahan runtime akibat kesalahan ID.

Selanjutnya, ada `ViewModel` lebih tepatnya `BookViewModel` dibuat menggunakan `ViewModelProvider` dan `BookViewModelFactory` dengan penggunaan `factory` diperlukan karena `BookViewModel` membutuhkan parameter (objek `Application`). Dimana `ViewModel` ini disiapkan di level `activity` agar bisa dibagikan ke `HomeFragment` dan `DetailFragment`. Berikutnya, ada kode `if (savedInstanceState == null)` untuk memastikan bahwa `HomeFragment` hanya bisa dimuat sekali saja saat `activity` pertama kali dibuat oleh programnya. Apabila `activity` dipulihkan ketika kita melakukan rotasi layar atau pembaruan konfigurasi lainnya, `fragment` tidak akan dimuat ulang secara manual karena sistem Android sudah memulihkannya kembali.

Setelah itu, observasi `Loading` dan `Error` dengan fungsi `observeViewModel()` yang mengatur dua `observer` seperti, `isLoadingLiveData` perlu diamati agar bisa menampilkan atau menyembunyikan `ProgressBar` di UI sesuai status `loading` dan `errorLiveData` diamati untuk menampilkan pesan kesalahan dalam bentuk `Toast` jika terjadi error dalam program saat memuat data.\

Kesimpulan file `MainActivity` adalah bagian pusat kendali utama yang dalam mengelola UI dan menghubungkan `ViewModel` dengan

berbagai elemen tampilan seperti XML . Juga disini mengatur *splash screen* yang adaptif terhadap status loading data dan memberikan feedback error kepada user agar tau permasalahan dari aplikasi yang sedang dijalankan. Dimana penggunaan arsitektur MVVM, ViewBinding, serta fragment manajemen modern menjadikan struktur aplikasi ini *clean architecture* dan modular.

16. MyAdapter

Pada file `MyAdapter` merupakan adapter yang menghubungkan data buku (`BookUi`) dengan tampilan daftar (`RecyclerView`) di aplikasi ini. Dimana adapter bertugas dalam membuat tampilan setiap item dengan mengikat data menuju tampilan tersebut dan menangani interaksi yang dilakukan oleh user seperti klik tombol atau item yang ada di aplikasi. Lalu, ada konstruktor dan parameter yang mana konstruktor dari `MyAdapter` menerima empat parameter penting seperti, `books` sebagai daftar data buku nantinya ditampilkan, `onDetailButtonClicked`, `onInfoButtonClicked`, dan `onItemRootClicked` menjadi function (callback) yang mengatur aksi saat user mengklik tombol "Detail", "Info", ataupun seluruh item. Hal ini memberikan fleksibilitas dan menjaga adapter tetap terpisah dari logika tampilan aplikasi.

Kemudian, class `ViewHolder` ada `BookViewHolder` merupakan class dalam (*inner class*) yang bertanggung jawab mengelola satu item tampilan di daftar dengan menyimpan objek `ItemListAdapter` berisi referensi langsung menuju elemen-elemen UI seperti `textViewName`, `imageView`, dll. Dalam `init`, setiap elemen diberi `setOnClickListener` yang mengeksekusi callback sesuai posisi item di dalam daftar. Setelah itu, ada binding data dari fungsi `bind()` dipanggil saat `RecyclerView` nantinya menampilkan data di item tertentu terutama di dalam fungsi ini, nilai properti `BookUi` seperti

`title`, `release_date`, dan `author` diatur ke `TextView`. Jika tersedia, maka gambar sampul buku (`cover`) dimuat ke `ImageView` menggunakan library `Glide`. Berikutnya, `onCreateViewHolder()` dengan fungsi ini dipanggil oleh `RecyclerView` membantu dalam membuat `ViewHolder` baru karena di sini `layout item_list.xml` di-inflate melalui `ViewBinding (ItemListAdapterBinding)`, dan sebuah instance `BookViewHolder` dikembalikan menjadikan tahap ini sebagai pembuatan tampilan tiap item list. Juga, ada `getItemCount()` yang mana fungsi ini mengembalikan jumlah item dalam daftar books. Nah, `RecyclerView` menggunakan nilai ini agar bisa mengetahui seberapa banyak data perlu ditampilkan di layar kepada user.

Selanjutnya, ada `onBindViewHolder()` menjadi fungsi yang dipanggil agar bisa mengisi data menuju `ViewHolder` sudah dibuat. Maksudnya fungsi `bind()` dipanggil dengan data buku di posisi tertentu untuk tampilan dapat menampilkan isi yang sesuai. Serta, ada `updateBooks()` merupakan fungsi sering digunakan untuk memperbarui seluruh daftar buku yang ingin ditampilkan. Dimana daftar yang lama dihapus (`clear()`), nantinya diganti dengan data baru (`addAll(newBooks)`), dan `notifyDataSetChanged()` dipanggil agar `RecyclerView` tahu bahwa seluruh datanya harus diperbarui.

Kesimpulan dari file `MyAdapter` ini tentang bagaimana cara implementasi adapter yang efisien dan modular dalam menampilkan daftar buku dengan memisahkan logika tampilan dan interaksi ke dalam `ViewHolder` serta menggunakan `ViewBinding` dan `Glide` agar kode menjadi lebih bersih, mudah dibaca, dan fleksibel untuk dikembangkan kedepannya.

17. BookViewModel

Pada file `BookViewModel` merupakan bagian dari arsitektur MVVM yang bertugas menjembatani UI (fragment/activity) dengan domain layer (use case dan repository) mewarisi `AndroidViewModel` untuk mengakses `Application context`, yang digunakan saat membuat repository. Dimana `ViewModel` ini untuk memuat data buku dari domain dan menyediakannya ke UI melalui `LiveData`. Nah, disini juga melakukan pembuatan dependency secara manual karena alih-alih kita menggunakan `Dependency Injection` (seperti `Hilt`), maka kode ini secara manual membuat instance `BookRepositoryImpl` menggunakan `applicationContext`, dan kemudian membuat instance `GetBooksUseCase`.

Kemudian, perlu melakukan deklarasi dan penggunaan `LiveData` karena `ViewModel` ini memiliki tiga `LiveData` utama seperti, `_books` untuk menyimpan daftar data buku nantinya ditampilkan di UI bertipe `List<BookUi>`, `_isLoading` menunjukkan apakah data sedang dimuat dengan biasanya untuk menampilkan atau menghilangkan `ProgressBar`, dan `_errorMessage` sebagai penyimpanan pesan kesalahan apabila terjadi error saat mengambil data. Dimana semua variabel `LiveData` disediakan ke UI dalam bentuk `val` agar hanya bisa dibaca dari luar dan tidak bisa dimodifikasi langsung dalam menjaga prinsip enkapsulasi.

Selanjutnya, ada fungsi `fetchBooks()` untuk memulai proses pengambilan data buku dengan cara `isLoading` di-set `true` sebagai pemberi sinyal bahwa proses sedang berjalan, lanjutannya `viewModelScope.launch` digunakan untuk menjalankan coroutine agar proses jaringan berlangsung secara asinkron tanpa memblok UI thread. Nah, di dalam coroutine `getBooksUseCase()` dipanggil untuk mengambil data dari domain layer yang hasilnya ditangani oleh fungsi `onSuccess` dan `onFailure` dalam kondisi jika berhasil, maka

data dari domain layer diubah (mapped) menuju `BookUi` menggunakan `toUiModel()` yang dikirim ke UI melalui `_books` serta jika gagal, maka error message ditampilkan melalui `_errorMessage`. Setelah selesai, `isLoading` dikembalikan ke `false` agar progress indicator bisa disembunyikan.

Berikutnya, logging dan debugging karena di kode saya lebih tepatnya di file ini ada melakukan pemanggilan `Log.d()` dan `Log.e()` agar mencatat keberhasilan atau kegagalan pemanggilan data. Hal ini berguna untuk proses debugging selama pengembangan aplikasi dan memberikan visibilitas tentang apa saja yang terjadi ketika fungsi `fetchBooks()` dijalankan.

Kesimpulan dari file `BookViewModel` menjadi pusat logika pada tampilan yang berperan dalam mengatur pengambilan dan penyediaan data buku menuju UI. Dengan adanya bantuan `LiveData` dan `coroutine (viewModelScope)` membuat `ViewModel` ini bisa menjaga UI tetap responsif, menangani loading atau error state, serta memisahkan logika tampilan dari logika data. Nah, pendekatan inilah yang membuat aplikasi lebih mudah diuji dan dirawat.

18. BookViewModelFactory

Pada file `BookViewModelFactory` berguna untuk membuat instance dari `BookViewModel`. Dimana `ViewModel` ini memiliki konstruktor dengan parameter (`Application`), sehingga tidak bisa langsung dibuat oleh sistem melalui `ViewModelProvider()` secara default. Oleh karena itu, kita perlu membuat `Factory` khusus disini untuk menangani pembuatan `BookViewModel` yang memerlukan argumen tambahan.

Kemudian, di file ini ada class yang mengimplementasikan interface `ViewModelProvider.Factory` yang mengharuskan kita untuk menulis override fungsi `create()` karena fungsi ini nantinya

dipanggil saat sistem ingin membuat instance `ViewModel` dan kita bisa mengatur cara pembuatannya sendiri.

Lalu, ada implementasi fungsi `create()` untuk menerima parameter `modelClass`, yaitu class `ViewModel` yang ingin dibuat. Di dalam fungsi akan dilakukan pengecekan apakah `modelClass` cocok dengan `BookViewModel`. Jika cocok, maka dibuat dan dikembalikan instance `BookViewModel` dengan menyertakan `application` sebagai parameter konstruktor. Perlu diperhatikan karena casting dilakukan secara manual, maka saya disini menggunakan anotasi `@SuppressWarnings("UNCHECKED_CAST")` agar bisa menghindari peringatan kompilator. Tetapi ada kondisi jika tipe `ViewModel` yang diminta bukan `BookViewModel`, maka nanti dilemparkan `IllegalArgumentException` sebagai bentuk validasi.

Selanjutnya, Perlunya `Factory` karena sangat penting dalam arsitektur MVVM Android saat `ViewModel` mempunyai dependensi konstruktor seperti `Application` atau objek lain (misalnya `repository`). Tanpa adanya `factory` ini, kita akan mendapatkan error saat mencoba membuat instance `BookViewModel` menggunakan `ViewModelProvider()` secara langsung.

Kesimpulan dari file `BookViewModelFactory` menjadi solusi di Android dalam hal menangani pembuatan `ViewModel` dengan konstruktor berparameter. Dengan menerapkan prinsip ini dapat menggunakan `ViewModel` yang lebih fleksibel dan sesuai dengan kebutuhan aplikasi saya buat terutama saat memerlukan konteks aplikasi atau dependensi lainnya ketika proses inisialisasi. Adanya `Factory` ini menjaga arsitektur aplikasi tetap modular dan testable.

19. Detail_fragment.xml

Pada file `detail_fragment.xml` ada struktur utama yaitu `ConstraintLayout` digunakan sebagai root layout dalam file ini karena memberikan fleksibilitas ketika saya ingin mengatur tata letak tampilan UI secara responsif. Dimana layout ini mengatur posisi elemen-elemen utama seperti `Toolbar` dan `NestedScrollView` sesuai constraint terhadap parent atau elemen lain, sehingga memungkinkan desainnya konsisten di berbagai ukuran layar. Lalu, ada elemen `Toolbar` berguna menjadi elemen paling atas sebagai action bar khusus untuk fragment detail ini, termasuk `back button` dan judul halaman. Dimana `toolbar` ini disesuaikan tampilannya dengan warna latar belakang dan tema `AppCompat` gelap, serta teks judul berwarna putih. Hal ini diterapkan dalam project ini untuk meningkatkan konsistensi navigasi dan pengalaman user saat berpindah antar halaman.

Kemudian, ada `NestedScrollView` untuk konten `Scrollable` yang mana bagian `NestedScrollView` akan membungkus seluruh konten di bawah `Toolbar` dan memberi kemungkinan kepada user bisa menggulir atau `scroll` isi halaman jika terlalu panjang. Hal ini sangat penting agar deskripsi buku yang panjang tetap dapat dibaca seluruhnya tanpa mengganggu struktur visual antarmukanya. Nah, `ScrollView` ini diberi constraint agar menempel ke bagian bawah `Toolbar` dan mengikuti ukuran parent di sisi samping dan bawah.

Selanjutnya, ada `LinearLayout` sebagai kontainer vertikal karena di dalam `NestedScrollView` terdapat `LinearLayout` dengan orientasi vertikal. Dengan adanya layout ini untuk menempatkan `ImageView` (gambar cover buku) dan `LinearLayout` lain berisi teks-teks deskriptif (seperti judul dan deskripsi buku). Dimana struktur vertikal ini berguna untuk menjaga urutan elemen agar tampil dari atas ke bawah secara rapi. Berikutnya, `ImageView` untuk sampul buku digunakan untuk

menampilkan gambar sampul buku dengan tinggi tetap 600dp melibatkan atribut `scaleType="centerCrop"` agar gambar menyesuaikan lebar tampilan lebih baik di layar. Juga, jangan lupa peran `tools:src` yang hanya digunakan saat preview di Android Studio karena atribut ini memberi gambaran kepada saya sebagai developer tentang tampilan visual tanpa perlu memengaruhi runtime aplikasi ini.

Setelah itu, ada konten berupa teks berisi judul dan deskripsi terletak di bagian bawah terdapat `LinearLayout` berisi dua `TextView`. Dimana `TextView` pertama (`detailTitle`) digunakan untuk menampilkan judul buku dengan gaya `Headline5` agar menonjol. Terus peran dari `TextView` kedua (`detailDescription`) digunakan untuk deskripsi buku karena diberi `lineSpacingMultiplier` agar teks lebih nyaman dibaca. `Padding` keseluruhan diberikan di container `LinearLayout` ini agar isi teks tidak menempel ke pinggir layar dan memberikan tampilan yang bagus membantu dalam membaca apa saja yang ada di aplikasi itu.

Kesimpulan file `detail_fragment.xml` dibuat untuk menampilkan tampilan detail dari sebuah buku dengan penekanan di bagian struktur yang fleksibel, scrollable, dan ramah pengguna. Dimana layout ini menyusun konten secara vertikal dengan gambar di atas dan teks di bawah, dipadukan dengan `Toolbar` untuk navigasi, dan menjadikannya cocok bagi aplikasi berbasis Android dengan arsitektur `fragment`.

20. activity_main.xml

Pada file `activity_main` ada layout ini menggunakan `ConstraintLayout` sebagai root agar memberikan fleksibilitas dalam menyusun elemen-elemen UI dengan sistem `constraint` atau batasan antar elemen. Dimana layout ini mengatur ukuran lebar dan tinggi untuk mengisi seluruh layar perangkat (`match_parent`). Terdapat atribut `tools:context` menuju bagian `MainActivity`, sehingga layout ini

nanti dipergunakan oleh activity tersebut dan `android:background` diatur ke tema default perangkat agar tampilan latar konsisten dengan sistem tersebut.

Kemudian, di dalam `ConstraintLayout` terdapat sebuah `FrameLayout` dengan ID `fragmentContainer`. Dimana elemen ini berguna sebagai tempat untuk menampilkan fragment, misalnya di sini `HomeFragment` atau `DetailFragment` nantinya ditambahkan dari kode `MainActivity`. Dengan ukuran `FrameLayout` dibuat `0dp` untuk lebar dan tinggi, namun dikendalikan oleh `constraint` agar menempel ke keempat sisi parent (atas, bawah, kiri, kanan). Hal ini membuat fragment yang dimuat dapat mengisi seluruh area tampilan tersedia.

Selanjutnya, ada `ProgressBar` digunakan sebagai indikator loading ketika data sedang diambil dari API atau sumber lain. Letaknya diatur di tengah layar `constraint` ke semua sisi (`top`, `bottom`, `start`, dan `end`) dari parent layout. Default-nya bagian `ProgressBar` disembunyikan (`android:visibility="gone"`) agar tidak selalu terlihat, tetapi bisa diatur tampil ketika aplikasi sedang dalam proses latar seperti pemanggilan data. Terdapat atribut `tools:visibility="visible"` hanya berguna untuk preview di Android Studio agar saya tetap bisa melihat posisi `ProgressBar` saat mendesain UI.

Kesimpulan file `activity_main.xml` merupakan layout utama dari `MainActivity` yang menyiapkan dua elemen inti seperti, `FrameLayout` sebagai kontainer dinamis untuk fragment, dan `ProgressBar` sebagai penanda proses loading. Dengan adanya struktur simpel namun fleksibel ini, maka `MainActivity` dapat berfungsi sebagai kerangka utama aplikasi secara dinamis menampilkan berbagai tampilan (fragment) dan memberi feedback visual saat data sedang melakukan proses.

21. home_fragment.xml

Pada file `home_fragment.xml` menggunakan `ConstraintLayout` sebagai root layout. Hal ini memungkinkan penempatan elemen UI lebih fleksibel menggunakan aturan constraint antar elemen. Dimana layout diatur agar mengisi seluruh layar (`match_parent` untuk lebar dan tinggi), dan atribut `tools:context` menunjukkan bahwa layout ini digunakan oleh `HomeFragment`. Dengan adanya pendekatan ini, UI dari fragment bisa tampil dengan layout konsisten dan responsif.

Kemudian, ada `AppBarLayout` dan `Toolbar` terdiri atas header fragment karena bagian atas layout berisi `AppBarLayout` dari material design berfungsi sebagai tempat untuk elemen toolbar yang di dalamnya terdapat `Toolbar` dengan ID `homeToolbar` menjadi header untuk fragment ini. Dimana `Toolbar` menampilkan nama aplikasi (melalui `app:title="@string/app_name"`) dengan latar belakang mengikuti tema utama aplikasi (`?attr/colorPrimary`) dan teks berwarna putih agar kontras. `Toolbar` ini juga menggunakan tema `Dark.ActionBar` untuk memastikan tampilan sesuai aplikasi.

Selanjutnya, `RecyclerView` berisi daftar buku dengan ini bagian inti dari layout adalah `RecyclerView` dengan ID `rv_character` berguna untuk menampilkan daftar item buku atau karakter. Dimana `RecyclerView` ditempatkan tepat di bawah `AppBarLayout` dan memenuhi sisa ruang layar sampai ke bawah (`layout_constraintTop_toBottomOf, Bottom_toBottomOf, dll`). Adanya `RecyclerView` ini menggunakan `LinearLayoutManager` (vertikal secara default) agar bisa menyusun item satu per satu ke bawah dan padding 4dp untuk jarak dari tepi layar. Melibatkan atribut `tools:listitem="@layout/item_list"` hanya digunakan saat melakukan pratinjau tampilan list di Android Studio menggunakan file layout `item_list.xml`.

Kesimpulan file `home_fragment.xml` untuk menyusun interface `HomeFragment` yang berfungsi sebagai halaman utama aplikasi. Dengan komponen `Toolbar` berguna bagi navigasi ataupun header, sedangkan `RecyclerView` di sini menampilkan daftar data seperti buku atau karakter yang diambil dari API. Struktur layout ini mendukung pemakaian `fragment` yang ringan, responsif, dan modular di aplikasi saya buat.

22. `item_list.xml`

Pada file `item_list.xml` menggunakan `CardView` untuk memberikan efek visual seperti kartu dengan sudut melengkung dan bayangan (`elevation`), sehingga tampilan daftar terlihat rapi. Dimana properti seperti `cardCornerRadius="16dp"` agar bisa memberikan efek melengkung, sementara `cardElevation="4dp"` menciptakan bayangan agar terlihat sedikit "terangkat" dari latar belakang. Melibatkan margin luar `8dp` juga memberikan ruang antar kartu dalam daftar `RecyclerView`. Lalu, di dalam `CardView` terdapat `ConstraintLayout` sebagai kontainer utama agar bisa menyusun semua komponen UI secara efisien. Dengan padding sebesar `16dp` agar tampilan dalam kartu mempunyai ruang yang baik dan terlihat proporsional. `ConstraintLayout` berguna dalam mengatur posisi relatif antar komponen (gambar, teks, dan tombol) presisi tinggi.

Kemudian, ada `ImageView` berfungsi saat ingin menampilkan gambar cover buku dengan ID `imageView` yang memiliki lebarnya diatur `100dp` dan tingginya `150dp` dengan `scaleType="centerCrop"` agar gambar memenuhi area tanpa distorsi. Posisi gambar ditautkan menuju sisi kiri dan atas kontainer menggunakan `constraint`. Terdapat atribut `tools:src` hanya digunakan sebagai placeholder saat melihat preview di Android Studio. Berikutnya, ada `TextView` dengan ID `textViewName` berguna untuk menampilkan judul utama buku. Dimana letaknya berada di sebelah kanan `ImageView` dengan margin kiri `16dp` sebagai pemisah visual. Adanya teks ini dibatasi hingga dua baris

(`maxLines="2"`) dan nanti dipotong jika terlalu panjang (`ellipsize="end"`), serta menggunakan teks tebal (`textStyle="bold"`) dan ukuran `Subtitle1` dari Material Design. Selanjutnya, ada dua `TextView` yaitu, `textViewAuthor` dan `textViewYear` berguna untuk menampilkan nama penulis dan tahun terbit buku secara berurutan di bawah judul. Dimana keduanya terletak tepat di bawah elemen sebelumnya menggunakan `constraint Top_toBottomOf`, sehingga konten teks tertata secara vertikal di sebelah gambar. Juga, `textViewAuthor` dibatasi hanya satu baris dan `textViewYear` menggunakan `Caption` agar lebih kecil dan bersifat informatif.

Setelah itu, ada bagian paling bawah dari layout yaitu, `LinearLayout` horizontal berisi dua tombol seperti `buttonInfo` dan `buttonDetail`. Dimana `LinearLayout` berada di sisi kanan bawah `card` dan menyesuaikan tombol ke kanan (`gravity="end"`). Tombol `buttonInfo` menggunakan `TextButton` agar tampil ringan, sementara `buttonDetail` memakai tombol standar dari *Material Components*. Nah, kedua tombol ini memungkinkan user nantinya bisa melihat `info` atau `detail` dari buku yang dipilih.

Kesimpulan file `item_list.xml` untuk mendefinisikan tampilan satu item (satu buku) dalam daftar `RecyclerView`. Dimana layout ini memadukan gambar cover, informasi dasar (judul, penulis, tahun), serta tombol menggunakan `CardView`. Struktur ini memastikan setiap item terlihat jelas, informatif, dan mudah untuk digunakan user.

23. nav_graph.xml

Pada file `nav_graph.xml` menggunakan elemen `<navigation>` sebagai root yang mendefinisikan struktur navigasi antar fragmen di dalam aplikasi. Dimana atribut `app:startDestination="@id/HomeFragment"` menentukan bahwa fragment pertama yang ditampilkan saat aplikasi dimulai itu

HomeFragment. Dengan elemen ini bagian dari fitur *Jetpack Navigation Component* memudahkan manajemen navigasi antar tampilan dalam arsitektur berbasis fragment.

Kemudian, di sini Fragment pertama yang dideklarasikan itu HomeFragment dengan ID `@+id/HomeFragment` dan nama class `com.example.myapi_test.presentation.ui.HomeFragment` untuk tampilan utama aplikasi. Di dalamnya terdapat `<action>` dengan ID `action_HomeFragment_to_DetailFragment` yang mendefinisikan transisi dari HomeFragment ke DetailFragment saat pengguna memilih salah satu item, misalnya dari daftar buku. Adanya elemen `<action>` ini sangat penting agar bisa memungkinkan ketika melakukan perpindahan antar tampilan sistem navigasi.

Selanjutnya, ada deklarasi DetailFragment dengan ID `@+id/detailFragment` dan nama class lengkap `com.example.myapi_test.presentation.ui.DetailFragment` menjadi tujuan dari aksi navigasi yang sebelumnya didefinisikan. Di dalam fragment ini terdapat tiga elemen `<argument>` yang masing-masing mendeskripsikan data yang harus diteruskan saat melakukan navigasi, yaitu, `imageResId` tipe datanya integer untuk menyampaikan resource ID gambar (misalnya sampul buku), nama bertipe string sebagai nama buku atau judul, dan `deskripsi` bertipe string untuk deskripsi lengkap buku. Nah, dari semua `argument` ini memungkinkan DetailFragment menampilkan konten sesuai data diteruskan dari HomeFragment.

Kesimpulan file `nav_graph.xml` ini menjadi peta navigasi yang berguna dalam mengatur perpindahan antar tampilan fragment di dalam aplikasi. Juga, dari file ini menunjukkan bahwa pengguna memulai dari HomeFragment dan bisa berpindah menuju DetailFragment sambil membawa data penting seperti gambar, nama, dan deskripsi.

Menggunakan *Jetpack Navigation Component* sangat membantu navigasi ini menjadi terstruktur dan mudah dikelola di satu tempat

TAUTAN GIT

https://github.com/alysaarmelia/AlysaArmelia_2310817120009_Pemrograman_Mobile.git