LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Firda Khoirunisa **NIM. 2310817220025**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE
MODUL 4

Laporan Praktikum Pemrograman Mobile Modul 4: ViewModel and Debugging ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile. Laporan Prakitkum ini dikerjakan oleh:

Nama Praktikan : Firda Khoirunisa
NIM : 2310817220025

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

DAFTAR GAMBAR

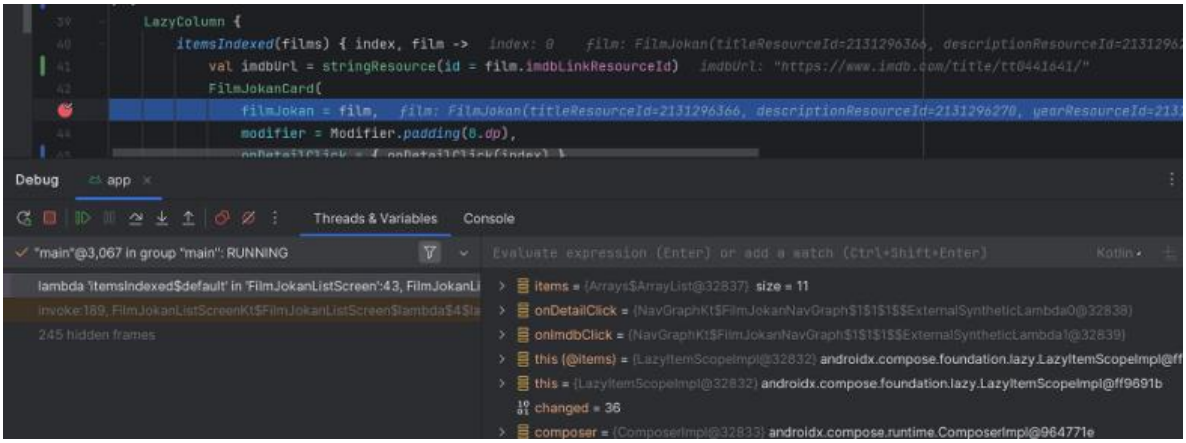
DAFTAR TABEL

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
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 1. Contoh Penggunaan Debugger

A. Source Code

1. MainActivity.kt

```
1  package com.example.modul4
2
3  import android.os.Bundle
4  import androidx.appcompat.app.AppCompatActivity
5  import com.example.modul4.databinding.ActivityMainBinding
6
7  class MainActivity : AppCompatActivity() {
8
9      private lateinit var binding: ActivityMainBinding
10
11     override fun onCreate(savedInstanceState: Bundle?) {
12         super.onCreate(savedInstanceState)
13         binding = ActivityMainBinding.inflate(layoutInflater)
14         setContentView(binding.root)
15
16         if (savedInstanceState == null) {
17             supportFragmentManager.beginTransaction()
18                 .replace(R.id.fragment_container,
19 HomeFragment())
20                 .commit()
21         }
22     }
23 }
```

Tabel 1. Source Code Jawaban Soal 1

2. Drama.kt

```
1  package com.example.modul4
2
3  import android.os.Parcelable
4  import kotlinx.parcelize.Parcelize
5
6  @Parcelize
7  data class Drama(
8      val title: String,
9      val yearGenre: String,
10     val rating: String,
11     val episodes: String,
12     val imageUrl: String,
13     val link: String
14 ) : Parcelable
```

Tabel 2. Source Code Jawaban Soal 1

3. ListDramaAdapter.kt

```
1  package com.example.modul4
2
3  import android.content.Intent
4  import android.net.Uri
```

```
5  import android.view.LayoutInflater
6  import android.view.ViewGroup
7  import androidx.recyclerview.widget.RecyclerView
8  import com.bumptech.glide.Glide
9  import com.example.modul4.databinding.ItemDramaBinding
10
11 class ListDramaAdapter(
12     private val listDrama: List<Drama>,
13     private val onDetailClick: (Drama) -> Unit
14 ) : RecyclerView.Adapter<ListDramaAdapter.MyViewHolder>() {
15
16     inner class MyViewHolder(val binding: ItemDramaBinding) :
17 RecyclerView.ViewHolder(binding.root)
18
19     override fun onCreateViewHolder(parent: ViewGroup, viewType:
20 Int): MyViewHolder {
21         val binding =
22 ItemDramaBinding.inflate(LayoutInflater.from(parent.context),
23 parent, false)
24         return MyViewHolder(binding)
25     }
26
27     override fun getItemCount(): Int = listDrama.size
28
29     override fun onBindViewHolder(holder: MyViewHolder,
30 position: Int) {
31         val drama = listDrama[position]
32         val context = holder.binding.root.context
33
34         holder.binding.drama = drama
35
36         holder.binding.buttonDetail.setOnClickListener {
37             onDetailClick(drama)
38         }
39
40         holder.binding.buttonLink.setOnClickListener {
41             val intent = Intent(Intent.ACTION_VIEW,
42 Uri.parse(drama.link))
43             context.startActivity(intent)
44         }
45     }
46 }
```

Tabel 3. Source Code Jawaban Soal 1

4. HomeFragment.kt

```
1  package com.example.modul4
2
3  import android.os.Bundle
4  import android.util.Log
5  import android.view.LayoutInflater
6  import android.view.View
```

```
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.modul4.databinding.FragmentHomeBinding
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch

class HomeFragment : Fragment() {

    private lateinit var binding: FragmentHomeBinding
    private val viewModel: DramaViewModel by viewModels {
        DramaViewModelFactory()
    }
    private lateinit var dramaAdapter: ListDramaAdapter

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?
    ): View {
        binding = FragmentHomeBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        dramaAdapter = ListDramaAdapter(emptyList()) { drama ->
            Log.d("HomeFragment", "Detail clicked:
${drama.title}")
            val fragment = DetailFragment().apply {
                arguments = Bundle().apply {
                    putParcelable("drama", drama)
                }
            }
            parentFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, fragment)
                .addToBackStack(null)
                .commit()
        }

        binding.recyclerView.apply {
            layoutManager = LinearLayoutManager(context)
            adapter = dramaAdapter
        }

        viewLifecycleOwner.lifecycleScope.launch {
            viewModel.dramas.collectLatest { list ->
                Log.d("HomeFragment", "List updated:
${list.size} items")
                dramaAdapter = ListDramaAdapter(list) { drama -
```

```
59  >
60                      Log.d("HomeFragment",    "Detail    clicked:
61  ${drama.title}")
62                      val  fragment  =  DetailFragment().apply  {
63                          arguments     =     Bundle().apply     {
64                              putParcelable("drama",       drama)
65                          }
66                      }
67                      parentFragmentManager.beginTransaction()
68                          .replace(R.id.fragment_container,
69  fragment)
70                          .addToBackStack(null)
71                          .commit()
72                  }
73                  binding.recyclerView.adapter   =   dramaAdapter
74              }
75          }
76      }
77  }
```

Tabel 4. Source Code Jawaban Soal 1

5. DetailFragment.kt

```
1   package                                          com.example.modul4
2
3   import                                       android.content.Intent
4   import                                              android.net.Uri
5   import                                            android.os.Bundle
6   import                                   android.view.LayoutInflater
7   import                                            android.view.View
8   import                                       android.view.ViewGroup
9   import                                androidx.fragment.app.Fragment
10  import     com.example.modul4.databinding.FragmentDetailBinding
11
12  class        DetailFragment        :        Fragment()        {
13      private   lateinit   var   binding:   FragmentDetailBinding
14
15      override                  fun                  onCreateView(
16          inflater:   LayoutInflater,   container:   ViewGroup?,
17          savedInstanceState:                                Bundle?
18      ):                            View?                              {
19          binding    =    FragmentDetailBinding.inflate(inflater,
20  container,                                                  false)
21
22          val  drama  =  arguments?.getParcelable<Drama>("drama")
23          drama?.let          {          selectedDrama          ->
24              binding.drama             =             selectedDrama
25
26              binding.openLinkButton.setOnClickListener          {
27                  val    intent    =    Intent(Intent.ACTION_VIEW,
28  Uri.parse(selectedDrama.link))
29                  startActivity(intent)
```

```
            }

            binding.backButton.setOnClickListener {
requireActivity().supportFragmentManager.popBackStack()
            }
        }

        return binding.root
    }
}
```

Tabel 5. Source Code Jawaban Soal 1

6. DramaViewModel.kt

```
package com.example.modul4

import android.util.Log
import androidx.lifecycle.ViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow

class DramaViewModel : ViewModel() {

    private val _dramas =
MutableStateFlow<List<Drama>>(emptyList())
    val dramas: StateFlow<List<Drama>> = _dramas

    private val _selectedDrama = MutableStateFlow<Drama?>(null)
    val selectedDrama: StateFlow<Drama?> = _selectedDrama

    init {
        _dramas.value = listOf(
            Drama(
                "Mysterious Lotus Casebook", "2023 · Mystery,
Wuxia", "Rating: 8.7", "40 Episodes",
                "https://i.mydramalist.com/kAozjj_4c.jpg?v=1",
                "https://www.iq.com/album/mysterious-lotus-
casebook-2023-hg4cefqzed?lang=en_us"
            ),
            Drama(
                "Love Game In Eastern Fantasy", "2024 · Romance,
Wuxia, Fantasy", "Rating: 8.5", "32 Episodes",
                "https://i.mydramalist.com/VXOLzy_4c.jpg?v=1",

"https://wetv.vip/en/play/227hqhmxvabwggz/d4100tnphtz"
            ),
            Drama(
                "Melody of Golden Age", "2024 · Historical,
Mystery, Romance, Drama", "Rating: 8.1", "40 Episodes",
                "https://i.mydramalist.com/d0Q62d_4c.jpg?v=1",
                "https://www.iq.com/album/melody-of-golden-age-
```

```
2025-vobwm47vvd?lang=en_us"
            ),
            Drama(
                "One And Only", "2021 · Military, Historical,
Romance,    Political",    "Rating:    8.6",    "24    Episodes",
                "https://i.mydramalist.com/BLrkR_4c.jpg?v=1",
                "https://www.iq.com/album/one-and-only-2021-
24mppdujjp9?lang=en_us"
            ),
            Drama(
                "The Prisoner of Beauty", "2025 · Historical,
Romance,    Political",    "Rating:    8.9",    "36    Episodes",
                "https://i.mydramalist.com/mOE1XJ_4c.jpg?v=1",

"https://wetv.vip/en/play/nwo9p9nml6dgm8l/e41010dzbwf-
EP01%3A_The_Prisoner_of_Beauty"
            )
        )

        Log.d("DramaViewModel", "Drama list initialized with
${_dramas.value.size}                                    items")
    }

    fun          selectDrama(drama:          Drama)          {
        _selectedDrama.value              =              drama
        Log.d("DramaViewModel",        "Drama        selected:
${drama.title}")
    }
}
```

Tabel 6. Source Code Jawaban Soal 1

7. DramaViewModelFactory.kt

```
package                                          com.example.modul4

import                              androidx.lifecycle.ViewModel
import                      androidx.lifecycle.ViewModelProvider

class   DramaViewModelFactory   :   ViewModelProvider.Factory   {
    override fun <T : ViewModel> create(modelClass: Class<T>):
T                                                                {
        if
(modelClass.isAssignableFrom(DramaViewModel::class.java))     {
            return        DramaViewModel()        as        T
        }
        throw    IllegalArgumentException("Unknown    ViewModel
class")
    }
}
```

Tabel 7. Source Code Jawaban Soal 1

8. activity_main.xml

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

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"                />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Tabel 8. Source Code Jawaban Soal 1

9. item_drama.xml

```
<?xml              version="1.0"              encoding="utf-8"?>
<layout
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">

    <data>
        <variable
            name="drama"
            type="com.example.modul4.Drama"                 />
    </data>

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        app:cardCornerRadius="12dp"
        app:cardElevation="4dp">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="12dp">

            <ImageView
                android:id="@+id/imageView"
                android:layout_width="100dp"
                android:layout_height="140dp"

android:contentDescription="@string/detail_image"
                android:scaleType="centerCrop"
                app:imageUrl="@{drama.imageUrl}"

app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
```

```
                app:layout_constraintTop_toTopOf="parent"
                tools:src="@tools:sample/avatars"           />

            <TextView
                android:id="@+id/titleText"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginStart="12dp"
                android:text="@{drama.title}"
                android:textSize="16sp"
                android:textStyle="bold"
                app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toEndOf="@id/imageView"

app:layout_constraintTop_toTopOf="@id/imageView"            />

            <TextView
                android:id="@+id/yearText"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="4dp"
                android:text="@{drama.yearGenre}"
                android:textSize="14sp"
                app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="@+id/titleText"

app:layout_constraintTop_toBottomOf="@id/titleText"         />

            <TextView
                android:id="@+id/info1Text"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="2dp"
                android:text="@{drama.rating}"
                app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="@+id/titleText"

app:layout_constraintTop_toBottomOf="@id/yearText"          />

            <TextView
                android:id="@+id/info2Text"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginTop="2dp"
                android:text="@{drama.episodes}"
                app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="@+id/titleText"

```

```
app:layout_constraintTop_toBottomOf="@id/info1Text"          />

            <Button
                android:id="@+id/buttonLink"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:text="Link"

app:layout_constraintStart_toStartOf="@+id/titleText"

app:layout_constraintTop_toBottomOf="@id/info2Text"          />

            <Button
                android:id="@+id/buttonDetail"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="8dp"
                android:text="Detail"

app:layout_constraintStart_toEndOf="@id/buttonLink"

app:layout_constraintTop_toTopOf="@id/buttonLink"            />

        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.cardview.widget.CardView>
</layout>
```

Tabel 9. Source Code Jawaban Soal 1

10. fragment_home.xml

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

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"            />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Tabel 10. Source Code Jawaban Soal 1

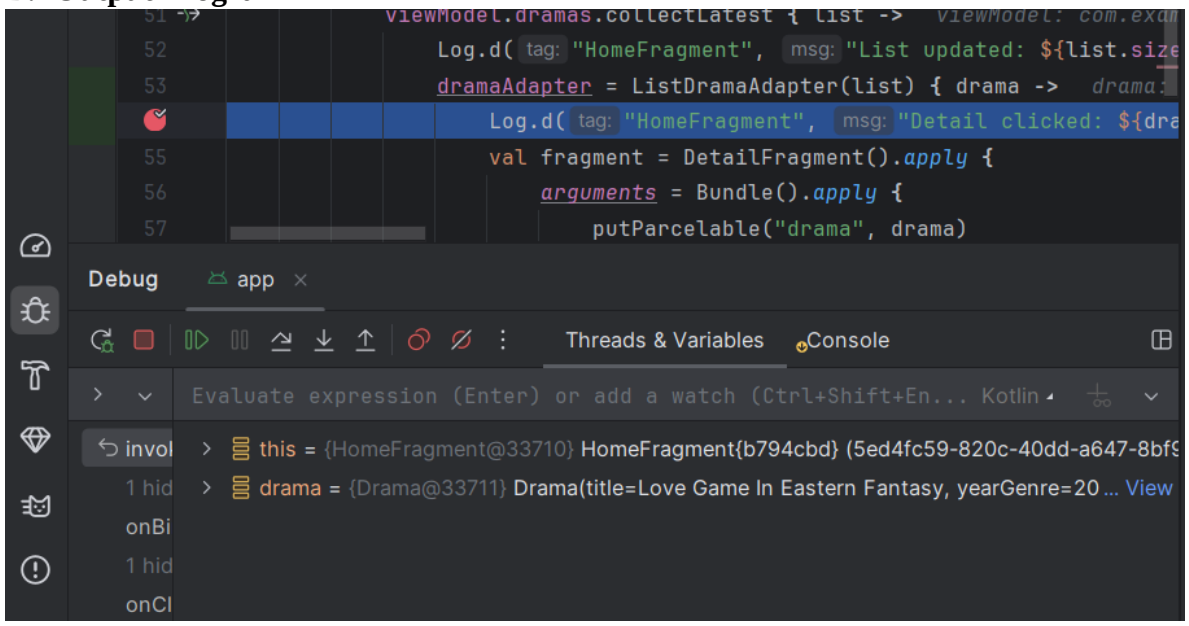
11. fragment_detail.xml

```
<?xml              version="1.0"              encoding="utf-8"?>
<layout
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">

    <data>
        <variable
            name="drama"
            type="com.example.modul4.Drama"                  />
    </data>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".DetailFragment">

        <LinearLayout
            android:orientation="vertical"
            android:padding="16dp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal">

            <ImageView
                android:id="@+id/detail_image"
                android:layout_width="match_parent"
                android:layout_height="500dp"
                android:scaleType="centerCrop"

android:contentDescription="@string/detail_image"
                app:imageUrl="@{drama.imageUrl}"             />

            <TextView
                android:id="@+id/title_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@{drama.title}"
                android:textSize="20sp"
                android:textStyle="bold"
                android:paddingTop="8dp"
                android:layout_gravity="center_horizontal"/>

            <TextView
                android:id="@+id/year_genre_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@{drama.yearGenre}"
                android:textSize="16sp"
                android:paddingTop="4dp"
                android:layout_gravity="center_horizontal"/>

```

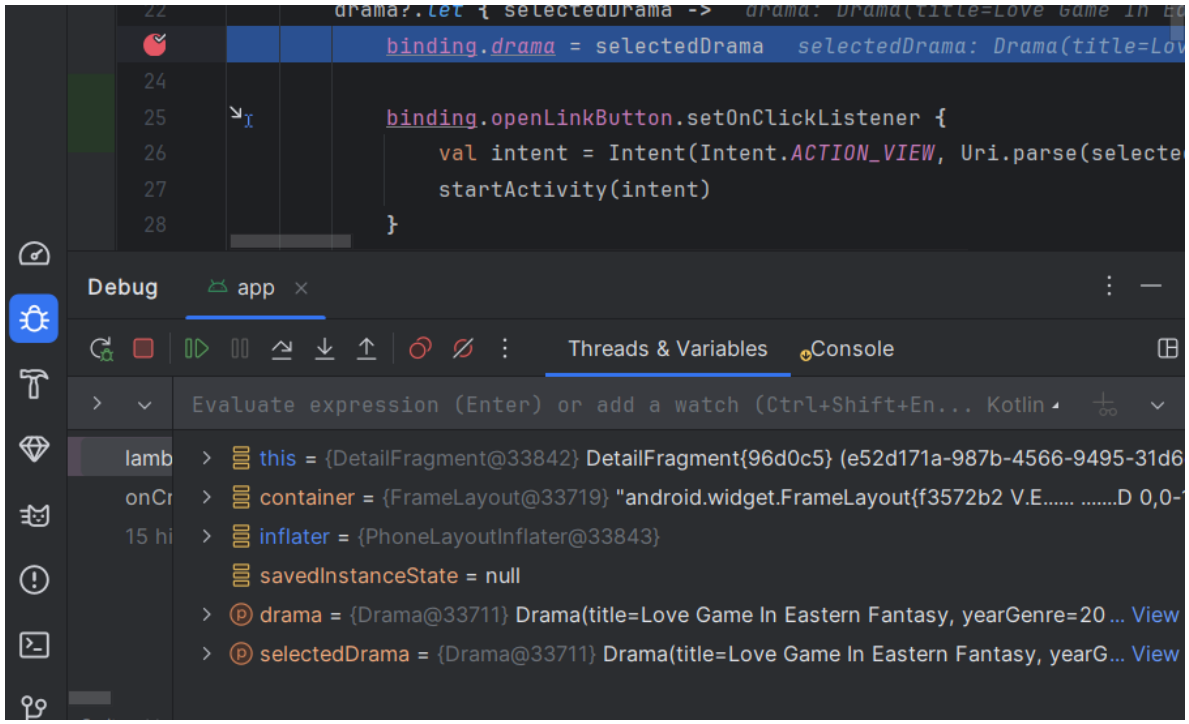
```
53              <TextView
54                  android:id="@+id/rating_text"
55                  android:layout_width="wrap_content"
56                  android:layout_height="wrap_content"
57                  android:text="@{drama.rating}"
58                  android:textSize="16sp"
59                  android:paddingTop="4dp"
60                  android:layout_gravity="center_horizontal"/>
61
62              <TextView
63                  android:id="@+id/episodes_text"
64                  android:layout_width="wrap_content"
65                  android:layout_height="wrap_content"
66                  android:text="@{drama.episodes}"
67                  android:textSize="16sp"
68                  android:paddingTop="4dp"
69                  android:layout_gravity="center_horizontal"/>
70
71              <Button
72                  android:id="@+id/openLinkButton"
73                  android:layout_width="wrap_content"
74                  android:layout_height="wrap_content"
75                  android:text="@string/watch_here"
76                  android:layout_marginTop="16dp"
77                  android:layout_gravity="center_horizontal"   />
78
79              <Button
80                  android:id="@+id/backButton"
81                  android:layout_width="wrap_content"
82                  android:layout_height="wrap_content"
83                  android:text="@string/button_back"
84                  android:layout_marginTop="16dp"
85                  android:layout_gravity="center_horizontal"/>
86          </LinearLayout>
87      </ScrollView>
88  </layout>
```

Tabel 11. Source Code Jawaban Soal 1

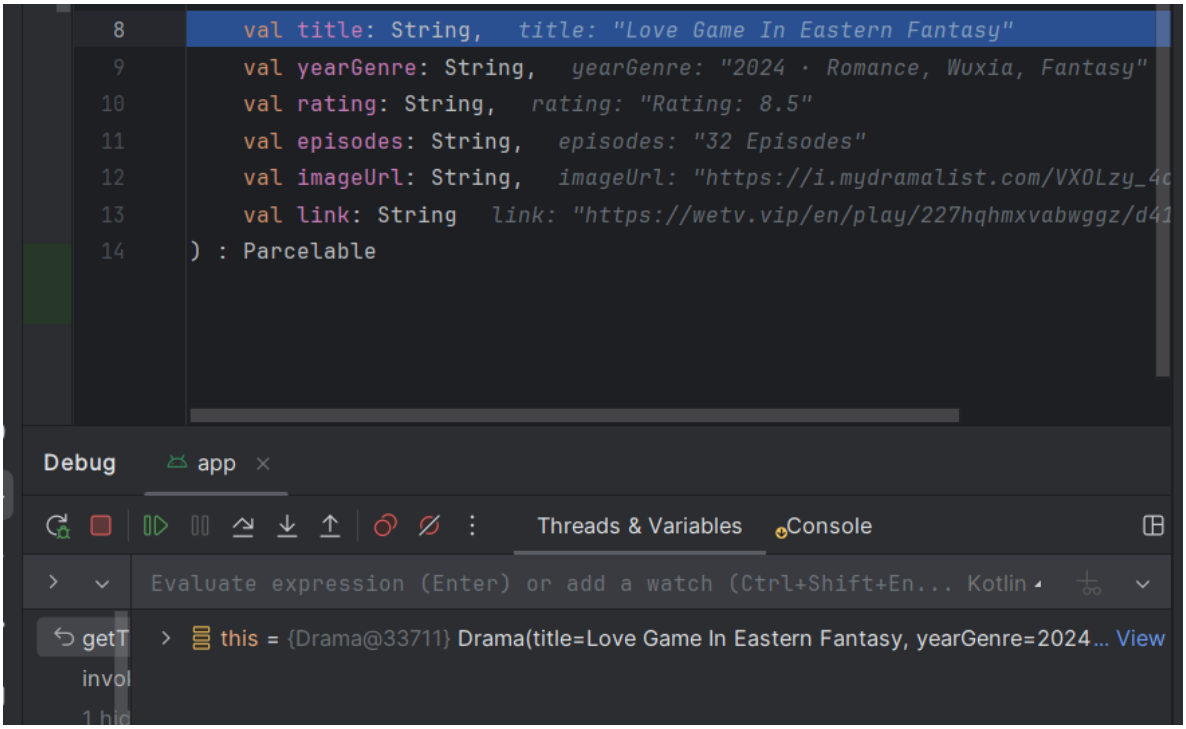
B. Output Program



Gambar 2. Screenshot Hasil Jawaban Soal 1



Gambar 3. Screenshot Hasil Jawaban Soal 1



Gambar 4. Screenshot Hasil Jawaban Soal 1

C. Pembahasan

1. MainActivity.kt:

File ini merupakan titik awal aplikasi Android, tempat di mana layout utama ditentukan dan fragment pertama dimuat. Di dalam metode onCreate(), fungsi setContentView() digunakan untuk menetapkan layout XML utama (activity_main.xml) yang berisi FrameLayout sebagai wadah fragment. Selanjutnya, ada pemeriksaan terhadap savedInstanceState, yang digunakan untuk memastikan bahwa fragment HomeFragment hanya ditambahkan ketika aplikasi pertama kali dijalankan, bukan saat dipulihkan dari keadaan sebelumnya. Jika kondisinya terpenuhi (yakni savedInstanceState == null), maka fragment HomeFragment dimuat ke dalam fragment_container menggunakan supportFragmentManager. Hal ini memastikan

pengguna langsung melihat daftar drama saat aplikasi dibuka, dan fragment tidak dimuat ulang secara tidak perlu saat orientasi layar berubah atau aplikasi dikembalikan dari latar belakang.

2. Drama.kt

File ini berisi sebuah data class bernama Drama, yang digunakan sebagai model data dalam aplikasi. Data class ini menyimpan lima properti utama: title (judul drama), yearGenre (kombinasi tahun rilis dan genre), episodes (jumlah episode), imageUrl (alamat URL gambar poster drama), dan link (URL video atau halaman nonton). Dengan adanya data class ini, setiap objek drama dapat dengan mudah dibuat dan dikelola di dalam daftar. Karena Drama mengimplementasikan Parcelable, objek-objek ini bisa dengan mudah dikirimkan antar fragment melalui Bundle, yang sangat penting dalam navigasi aplikasi.

3. ListDramaAdapter.kt

File ini berperan sebagai jembatan antara data dan tampilan pada RecyclerView. Kelas DramaAdapter mengambil daftar objek Drama serta sebuah fungsi callback bernama onDetailClick yang akan dipanggil saat pengguna menekan tombol "Detail". Di dalamnya terdapat kelas DramaViewHolder, yang mengikat data ke layout item_drama.xml menggunakan data binding. Pada metode onBindViewHolder, adapter akan menetapkan data Drama ke properti binding.drama dan juga menetapkan listener tombol "Detail". Saat tombol ditekan, fungsi onDetailClick akan dijalankan dengan parameter objek Drama terkait, sehingga aktivitas atau fragment dapat merespons aksi pengguna untuk menampilkan detail. Struktur ini memisahkan logika tampilan dan logika interaksi, membuat kode lebih modular dan mudah dirawat.

4. HomeFragment.kt

Fragment ini bertugas menampilkan daftar drama yang tersedia kepada pengguna. Pada saat onCreateView(), fragment ini menggunakan FragmentHomeBinding untuk menghubungkan layout fragment_home.xml dengan kode Kotlin. RecyclerView yang ada di layout kemudian dihubungkan dengan DramaAdapter, dan daftar drama diisi menggunakan fungsi getListDrama() yang berisi data dummy. Fungsi ini menciptakan beberapa objek Drama

secara manual dan menambahkannya ke dalam list. Salah satu bagian penting di sini adalah penanganan tombol "Detail" pada setiap item. Saat tombol tersebut ditekan, aplikasi akan membuat instance DetailFragment, mengirim data drama melalui Bundle, dan mengganti tampilan fragment saat ini dengan DetailFragment. Proses ini dilakukan menggunakan parentFragmentManager dan transaksi fragment, memungkinkan navigasi antar halaman tanpa perlu memulai aktivitas baru.

5. DetailFragment.kt

DetailFragment merupakan bagian dari aplikasi Android yang berfungsi menampilkan informasi detail dari sebuah drama yang dipilih oleh pengguna. Kelas ini menggunakan View Binding melalui objek FragmentDetailBinding untuk menghubungkan layout XML dengan logika di Kotlin secara langsung dan aman dari kesalahan penulisan ID view. Pada metode onCreateView(), layout di-inflate menggunakan FragmentDetailBinding.inflate() dan view utama dikembalikan sebagai root view dari fragment. Data Drama yang dikirim dari fragment sebelumnya diambil dari arguments dengan key "drama", lalu dicek keberadaannya. Jika data tersebut tidak null, maka akan digunakan untuk mengisi tampilan melalui binding (binding.drama = selectedDrama). Tombol openLinkButton disiapkan untuk membuka link dari drama menggunakan Intent eksplisit yang menjalankan browser dengan URL yang berasal dari properti link pada objek drama. Sedangkan backButton diprogram untuk menavigasi kembali ke fragment sebelumnya dengan memanggil popBackStack() pada supportFragmentManager. Seluruh interaksi ini dirancang untuk memberikan pengalaman pengguna yang intuitif dan responsif saat melihat detail sebuah drama.

6. DramaViewModel.kt

DramaViewModel merupakan kelas turunan dari ViewModel yang berfungsi sebagai pengelola data untuk antarmuka pengguna pada aplikasi daftar drama. Di dalamnya terdapat dua properti utama yang dibungkus dengan MutableStateFlow, yaitu _dramas untuk menyimpan daftar drama dan _selectedDrama untuk menyimpan drama yang sedang dipilih pengguna. Kedua properti ini diekspose ke UI menggunakan StateFlow, memungkinkan komponen UI mengamati perubahan data secara reaktif dan efisien sesuai dengan prinsip arsitektur MVVM. Pada blok init, daftar drama diinisialisasi secara statis menggunakan

beberapa objek Drama yang berisi informasi seperti judul, genre, rating, jumlah episode, gambar, dan tautan menonton. Selain itu, terdapat fungsi selectDrama() yang digunakan untuk memperbarui nilai selectedDrama ketika pengguna memilih salah satu item, serta mencatat aktivitas tersebut melalui Log. Dengan memisahkan logika penyimpanan dan pengolahan data dari UI, kelas ini membantu menjaga kebersihan kode dan membuat aplikasi lebih mudah dirawat.

7. DramaViewModelFactory.kt

DramaViewModelFactory adalah kelas yang mengimplementasikan ViewModelProvider.Factory dan digunakan untuk membuat instance DramaViewModel secara eksplisit. Factory ini diperlukan karena ViewModelProvider tidak bisa langsung membuat ViewModel dengan konstruktor kustom tanpa bantuan kelas pembantu. Dalam implementasinya, method create() memeriksa apakah modelClass merupakan turunan dari DramaViewModel, dan jika ya, maka objek baru akan dibuat dan dikembalikan. Jika tidak cocok, factory akan melemparkan pengecualian IllegalArgumentException. Penggunaan factory ini mempermudah pengelolaan dependensi dan mendukung penerapan arsitektur yang bersih dan skalabel, terutama ketika ViewModel di masa depan membutuhkan parameter atau dependensi tambahan seperti repository atau context.

8. activity_main.xml

File ini adalah layout utama untuk aplikasi, yang menggunakan ConstraintLayout sebagai root view. Layout ini hanya memiliki satu elemen, yaitu FrameLayout dengan ID fragment_container, yang akan menjadi wadah untuk menampilkan fragment lainnya. FrameLayout ini mengisi seluruh layar dan akan digunakan untuk menampung fragment yang akan di-embed dalam activity ini. Jadi, layout utama aplikasi hanya berfokus pada tempat untuk memuat fragment, tanpa ada elemen tampilan lainnya.

9. item_drama.xml

File ini mendesain tampilan item dalam daftar drama yang akan ditampilkan di dalam RecyclerView. Pada bagian awal, dideklarasikan sebuah variabel drama yang bertipe com.example.modul3.Drama menggunakan data binding. Di dalam layout, terdapat sebuah

CardView dengan margin, padding, dan elevasi untuk memberikan efek bayangan. Di dalam CardView, terdapat beberapa elemen tampilan: sebuah ImageView untuk menampilkan gambar drama dengan proporsi yang diatur agar terlihat pas, dan dua TextView untuk menampilkan judul dan tahun dari drama. Selain itu, ada sebuah tombol "Detail" yang digunakan untuk menunjukkan detail dari drama ketika diklik. Semua elemen ini diletakkan dengan bantuan ConstraintLayout, yang memastikan elemen-elemen tersebut terorganisir dengan baik di dalam layout.

10. fragment_home.xml

File ini adalah layout untuk fragment yang menampilkan daftar drama dalam bentuk RecyclerView. RecyclerView diatur untuk mengisi seluruh ukuran fragment, dari atas ke bawah dan kiri ke kanan, berkat penggunaan ConstraintLayout dan pengaturan constraint yang tepat. Fragment ini bertugas untuk menampilkan daftar item dalam aplikasi, di mana setiap item tersebut akan menggunakan layout item_drama.xml yang telah didefinisikan sebelumnya. Dengan demikian, RecyclerView akan menjadi tempat di mana pengguna dapat menggulir dan melihat berbagai drama yang tersedia di aplikasi.

11. fragment_detail.xml

File ini digunakan untuk menampilkan detail dari sebuah drama yang dipilih. Layout menggunakan ScrollView untuk memastikan bahwa konten dapat digulir jika ukurannya melebihi layar. Di dalam ScrollView, terdapat LinearLayout dengan orientasi vertikal untuk mengatur elemen-elemen yang ada. Elemen pertama adalah ImageView yang menampilkan gambar drama, yang dihubungkan dengan URL gambar melalui data binding. Berikutnya ada beberapa TextView yang menampilkan informasi tentang drama, seperti judul, tahun, genre, dan jumlah episode, semuanya diatur agar terpusat secara horizontal dan diberi padding agar lebih rapi. Di bagian bawah, terdapat dua tombol: satu untuk membuka tautan ke video drama (dengan teks "Watch Now"), dan satu lagi untuk kembali ke halaman sebelumnya (dengan teks "Back"). Semua elemen ini diatur dengan cara yang membuat tampilan detail drama terlihat terstruktur dan mudah dinavigasi.

Fungsi Debugger

Debugger digunakan untuk mencari dan memperbaiki kesalahan (bug) dalam kode dengan menjalankan aplikasi secara bertahap. Alat ini membantu melihat nilai variabel dan alur eksekusi program secara langsung saat aplikasi berjalan.

Cara Menggunakan Debugger

1. Pasang breakpoint di baris kode yang ingin dianalisis.
2. Jalankan aplikasi dalam mode debug.
3. Saat aplikasi berhenti di breakpoint, kamu bisa melihat isi variabel, stack trace, dan alur program.
4. Gunakan fitur step untuk menelusuri kode lebih dalam.

Fitur Step:

- **Step Into**: Masuk ke dalam fungsi yang dipanggil.
- **Step Over**: Melewati fungsi tanpa masuk ke dalamnya.
- **Step Out**: Keluar dari fungsi dan kembali ke pemanggilnya.

Jawaban Soal 2

Application class dalam arsitektur aplikasi Android adalah kelas dasar yang merepresentasikan seluruh siklus hidup aplikasi. Kelas ini dibuat satu kali saat aplikasi dijalankan dan tetap aktif selama aplikasi berjalan. Fungsi utama Application class adalah sebagai tempat untuk menginisialisasi komponen atau data yang harus tersedia secara global dan bertahan sepanjang aplikasi, seperti konfigurasi library, pengaturan dependency injection, atau objek shared resources. Dengan menggunakan Application class, developer dapat mengelola state dan resource yang bersifat global tanpa perlu mengulang inisialisasi di setiap activity atau fragment. Ini membuat aplikasi lebih efisien dan terstruktur karena sumber daya penting hanya diinisialisasi sekali pada saat aplikasi mulai berjalan. Application class juga dapat dimanfaatkan untuk menangani event global seperti perubahan konfigurasi atau memonitor lifecycle aplikasi secara keseluruhan.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

Pemrograman-Mobile/MODUL4 at main · firdakhrns/Pemrograman-Mobile