LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Adrian Bintang Saputera NIM. 2310817110006

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 : Adrian Bintang Saputera
NIM : 2310817110006

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Muhammad Raka Azwar NIM. 2210817210012	Andreyan Rizky Baskara, S.Kom., M.Kom. NIP. 1993070320190301011

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL

1. Lanjutkan aplikasi Android berbasis XML dan Jetpack Compose yang sudah dibuat pada Modul 3 dengan menambahkan modifikasi sesuai ketentuan berikut:
    a. Buatlah sebuah ViewModel untuk menyimpan dan mengelola data dari list item. Data tidak boleh disimpan langsung di dalam Fragment atau Activity.
    b. Gunakan ViewModelFactory untuk membuat parameter dengan tipe data String di dalam ViewModel
    c. Gunakan StateFlow untuk mengelola event onClick dan data list item dari ViewModel ke Fragment
    d. Install dan gunakan library Timber untuk logging 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.

Application Class dalam pengembangan aplikasi Android merupakan kelas fundamental yang berperan sebagai titik awal global sebelum komponen lain seperti Activity atau Service dijalankan. Kelas ini, yang merupakan turunan dari android.app.Application, berfungsi sebagai wadah untuk inisialisasi komponen-komponen penting yang dibutuhkan seluruh aplikasi, seperti library logging (Timber), database (Room), atau framework dependency injection (Dagger Hilt/Koin). Selain itu, Application Class juga menjadi tempat ideal untuk menyimpan data global yang perlu diakses berbagai bagian aplikasi, misalnya

konfigurasi API atau informasi sesi pengguna. Dengan lifecycle yang mencakup seluruh durasi aplikasi, kelas ini memungkinkan developer memantau status aplikasi secara menyeluruh, seperti saat aplikasi berpindah antara latar belakang dan latar depan.

Step Into, Step Over, dan Step Out:

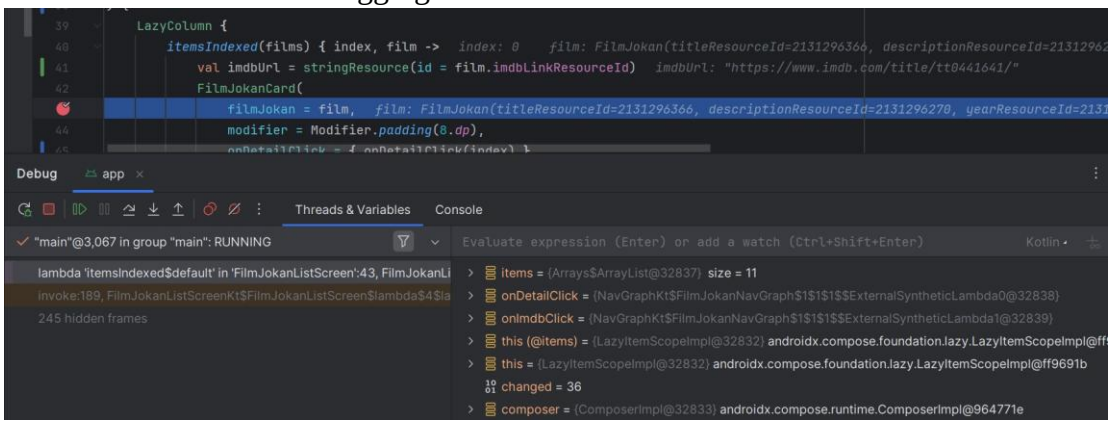
Debugger adalah alat penting dalam pengembangan perangkat lunak yang digunakan untuk menemukan dan memperbaiki kesalahan (bug) dalam kode program. Fungsi utama debugger adalah memungkinkan developer menjalankan program secara terkontrol, memeriksa nilai variabel, melacak alur eksekusi, dan memahami mengapa suatu bagian kode tidak berjalan sesuai harapan. Dengan debugger, programmer dapat menghentikan (pause) program pada titik tertentu yang disebut breakpoint, kemudian menganalisis kondisi program pada saat itu, sehingga memudahkan proses identifikasi kesalahan secara tepat dan efisien.

Cara menggunakan debugger biasanya dimulai dengan menetapkan breakpoint pada baris kode yang ingin diperiksa. Setelah breakpoint diaktifkan, program dijalankan dalam mode debugging. Ketika program mencapai breakpoint, eksekusi akan berhenti sementara, memungkinkan developer untuk melihat nilai variabel, kondisi memori, dan jalur logika yang sedang berjalan. Pada titik ini, debugger menyediakan kontrol eksekusi langkah demi langkah agar pengguna dapat mengeksplorasi jalannya program secara detail.

Fitur-fitur seperti Step Into, Step Over, dan Step Out membantu dalam navigasi selama debugging. Step Into digunakan untuk masuk ke dalam fungsi atau metode yang dipanggil di baris kode saat ini, memungkinkan developer melihat eksekusi kode di dalam fungsi tersebut secara rinci. Step Over menjalankan baris kode saat ini secara keseluruhan tanpa masuk ke fungsi yang dipanggil, cocok digunakan jika fungsi tersebut sudah dipastikan benar dan tidak perlu diperiksa lagi. Sementara itu, Step Out digunakan untuk keluar dari fungsi saat ini dan kembali ke kode

pemanggil, berguna jika developer ingin mengakhiri penelusuran di dalam fungsi dan kembali ke konteks yang lebih luas. Ketiga fitur ini sangat membantu untuk memahami jalur eksekusi dan perilaku program secara mendalam, sehingga mempermudah proses debugging.

Berikut adalah contoh debugging dalam Android Studio.



Gambar 1. Contoh Penggunaan Debugger

A. Source Code

XML:

MainActivity

Table 1. Source Code MainActivity XML

```
1  package com.example.listxml
2
3  import android.os.Bundle
4  import androidx.appcompat.app.AppCompatActivity
5  import com.example.listxml.ui.fragment.HomeFragment
6  import timber.log.Timber
7
8  class MainActivity : AppCompatActivity() {
9      override fun onCreate(savedInstanceState:
10 Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13
14         Timber.plant(Timber.DebugTree())
15         Timber.i("MainActivity created")
16
17         if (savedInstanceState == null) {
18
```

```
supportFragmentManager.beginTransaction()
                .replace(R.id.frame_container,
HomeFragment())
                .commit()
        }
    }
}
```

HomeFragment.kt

Table 2. Source Code HomeFragment.kt XML

```
package com.example.listxml.ui.fragment

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.activityViewModels
import
androidx.recyclerview.widget.LinearLayoutManager
import com.example.listxml.data.model.Agent
import
com.example.listxml.databinding.FragmentHomeBindin
g
import com.example.listxml.ui.adapter.AgentAdapter
import
com.example.listxml.ui.adapter.OnAgentClickListene
r
import
com.example.listxml.viewmodel.AgentViewModel
import
com.example.listxml.viewmodel.AgentViewModelFactor
y
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.lifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch

```

```
class HomeFragment : Fragment() {

    private var _binding: FragmentHomeBinding? =
null
    private val binding get() = _binding!!

    private val viewModel: AgentViewModel by
activityViewModels {
        AgentViewModelFactory("Agent List")
    }

    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?) {
        val list = getListAgent()
        viewModel.setAgentList(list)


viewModel.agentList.collectIn(viewLifecycleOwner)
{ agentList ->
            val adapter = AgentAdapter(agentList,
object : OnAgentClickListener {
                override fun
onDetailClicked(agent: Agent) {
                    viewModel.selectAgent(agent)
                    val fragment =
DetailFragment()

parentFragmentManager.beginTransaction()

.replace(com.example.listxml.R.id.frame_container,
fragment)
                        .addToBackStack(null)
                        .commit()
                }
```

```
                override fun onLinkClicked(agent:
Agent) {
                    viewModel.onLinkClicked(agent)

startActivity(Intent(Intent.ACTION_VIEW,
Uri.parse(agent.detail)))
                }
            })
            binding.rvAgent.layoutManager =
LinearLayoutManager(context)
            binding.rvAgent.adapter = adapter
        }
    }

    private fun getListAgent(): List<Agent> {
        val dataName =
resources.getStringArray(com.example.listxml.R.arr
ay.data_agentName)
        val dataDesc =
resources.getStringArray(com.example.listxml.R.arr
ay.data_agentDesc)
        val dataRole =
resources.getStringArray(com.example.listxml.R.arr
ay.data_agentRole)
        val dataImage =
resources.obtainTypedArray(com.example.listxml.R.a
rray.data_agentImage)
        val dataDetail =
resources.getStringArray(com.example.listxml.R.arr
ay.data_agentLink)
        val list = mutableListOf<Agent>()
        for (i in dataName.indices) {
            val agent = Agent(
                dataName[i], dataRole[i],
dataDesc[i],
                dataImage.getResourceId(i, -1),
dataDetail[i]
            )
            list.add(agent)
        }
        dataImage.recycle()
        return list
    }
```

```
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}

fun <T> StateFlow<T>.collectIn(owner:
LifecycleOwner, collector: suspend (T) -> Unit) {
    owner.lifecycleScope.launch {

owner.lifecycle.repeatOnLifecycle(Lifecycle.State.
STARTED) {
            this@collectIn.collectLatest {
                collector(it)
            }
        }
    }
}
```

DetailFragment

Table 3. Source Code DetailFragment XML

```
1  package com.example.listxml.ui.fragment
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.fragment.app.activityViewModels
9  import
10 com.example.listxml.databinding.FragmentDetailBindi
11 ng
12 import com.example.listxml.viewmodel.AgentViewModel
13 import
14 com.example.listxml.viewmodel.AgentViewModelFactory
15 import timber.log.Timber
16
17 class DetailFragment : Fragment() {
18
19     private var _binding: FragmentDetailBinding? =
20 null
21     private val binding get() = _binding!!
22
       private val viewModel: AgentViewModel by
```

```
activityViewModels {
        AgentViewModelFactory("Agent Detail")
    }

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

    override fun onViewCreated(view: View,
savedInstanceState: Bundle?) {
        viewModel.selectedAgent.value?.let { agent
->
            Timber.i("Navigating to detail:
$agent")
            binding.detailName.text = agent.name
            binding.detailRole.text = agent.role
            binding.detailDesc.text = agent.desc

binding.detailImage.setImageResource(agent.image)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

AgentAdapter.kt

Table 4. Source Code AgentAdapter.kt XML

```
package com.example.listxml.ui.adapter

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.example.listxml.data.model.Agent
import
```

```
com.example.listxml.databinding.ItemAgentBinding

interface OnAgentClickListener {
    fun onDetailClicked(agent: Agent)
    fun onLinkClicked(agent: Agent)
}

class AgentAdapter(
    private val agentList: List<Agent>,
    private val listener: OnAgentClickListener
) :
RecyclerView.Adapter<AgentAdapter.AgentViewHolder>(
) {

    inner class AgentViewHolder(val binding:
ItemAgentBinding) :
        RecyclerView.ViewHolder(binding.root)

    override fun onCreateViewHolder(parent:
ViewGroup, viewType: Int): AgentViewHolder {
        val binding =
ItemAgentBinding.inflate(LayoutInflater.from(parent
.context), parent, false)
        return AgentViewHolder(binding)
    }

    override fun onBindViewHolder(holder:
AgentViewHolder, position: Int) {
        val agent = agentList[position]
        holder.binding.apply {
            agentName.text = agent.name
            agentRole.text = agent.role
            agentDesc.text = agent.desc

agentImage.setImageResource(agent.image)
            btnDetail.setOnClickListener {
listener.onDetailClicked(agent) }
            btnLink.setOnClickListener {
listener.onLinkClicked(agent) }
        }
    }

    override fun getItemCount(): Int =
agentList.size
}
```

Agent

Table 5. Source Code Agent XML

```
1  package com.example.listxml.data.model
2
3  import android.os.Parcelable
4  import kotlinx.parcelize.Parcelize
5
6  @Parcelize
7  data class Agent(
8      val name: String,
9      val role: String,
10     val desc: String,
11     val image: Int,
12     val detail: String = "",
13 ) : Parcelable
```

AgentViewModel

Table 6. Source Code AgentViewModel XML

```
1  package com.example.listxml.viewmodel
2
3  import androidx.lifecycle.ViewModel
4  import com.example.listxml.data.model.Agent
5  import kotlinx.coroutines.flow.MutableStateFlow
6  import kotlinx.coroutines.flow.StateFlow
7  import timber.log.Timber
8
9  class AgentViewModel(title: String) : ViewModel() {
10
11     private val _agentList =
12 MutableStateFlow<List<Agent>>(emptyList())
13     val agentList: StateFlow<List<Agent>> =
14 _agentList
15
16     private val _selectedAgent =
17 MutableStateFlow<Agent?>(null)
18     val selectedAgent: StateFlow<Agent?> =
19 _selectedAgent
20
21     init {
22         Timber.i("ViewModel created with title:
   $title")
```

```
    }

    fun setAgentList(list: List<Agent>) {
        Timber.i("Setting agent list with
${list.size} items")
        _agentList.value = list
    }

    fun selectAgent(agent: Agent) {
        Timber.i("Detail clicked: ${agent.name}")
        _selectedAgent.value = agent
    }

    fun onLinkClicked(agent: Agent) {
        Timber.i("Link clicked for: ${agent.name} -
${agent.detail}")
    }
}
```

AgentViewModelFactory

Table 7. Source Code AgentViewModelFactory XML

```
package com.example.listxml.viewmodel

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

@Suppress ("UNCHECKED_CAST")
class AgentViewModelFactory(private val title:
String) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass:
Class<T>): T {
        if
(modelClass.isAssignableFrom(AgentViewModel::class.
java)) {
            return AgentViewModel(title) as T
        }
        throw IllegalArgumentException("Unknown
ViewModel class")
    }
}
```

activity_main.xml

Table 8. Source Code activity_main.xml XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout
   xmlns:android="http://schemas.android.com/apk/res/and
   roid"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      tools:context=".MainActivity"
7      android:id="@+id/frame_container"
8      android:fitsSystemWindows="true">
9  </FrameLayout>
```

fragment_detail.xml

Table 9. Source Code fragment_detail.xml XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3  xmlns:android="http://schemas.android.com/apk/res/a
4  ndroid"
5      xmlns:app="http://schemas.android.com/apk/res-
6  auto"
7      xmlns:tools="http://schemas.android.com/tools"
8      android:layout_width="match_parent"
9      android:layout_height="match_parent"
10     android:padding="8dp"
11     tools:context=".ui.fragment.DetailFragment">
12
13     <android.widget.ScrollView
14         android:layout_width="match_parent"
15         android:layout_height="match_parent">
16
17
18 <androidx.constraintlayout.widget.ConstraintLayout
19             android:layout_width="match_parent"
20             android:layout_height="wrap_content"
21             android:padding="8dp">
22
23             <ImageView
24                 android:id="@+id/detail_image"
25                 android:layout_width="0dp"
26
   android:layout_height="match_parent"
```

```
                android:scaleType="centerCrop"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="parent"

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

            <TextView
                android:id="@+id/detail_name"
                android:layout_width="wrap_content"

android:layout_height="wrap_content"
                android:layout_marginTop="16dp"
                android:textSize="20sp"
                android:textStyle="bold"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@id/detail_ima
ge"
                tools:text="Brimstone" />

            <TextView
                android:id="@+id/detail_role"
                android:layout_width="wrap_content"

android:layout_height="wrap_content"
                android:textColor="#666"
                android:textSize="16sp"
                android:textStyle="italic"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@id/detail_nam
e"

app:layout_constraintVertical_bias="0.5"
                tools:text="Controller" />
```

```
            <TextView
                android:id="@+id/detail_desc"
                android:layout_width="0dp"

android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:textSize="14sp"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintHorizontal_bias="0.0"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@id/detail_rol
e"
                tools:text="Joining from the
U.S.A..." />


</androidx.constraintlayout.widget.ConstraintLayout
>

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

fragment_home.xml

Table 10. Source Code fragment_home.xml XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout
3  xmlns:android="http://schemas.android.com/apk/res/an
4  droid"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".ui.fragment.HomeFragment">
9
10     <androidx.recyclerview.widget.RecyclerView
11         android:id="@+id/rvAgent"
12         android:layout_width="match_parent"
```

```
        android:layout_height="match_parent" />
</FrameLayout>
```

item_agent.xml

Table 11. Source Code item_agnet.xml XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
android:layout_width="match_parent"
    android:id="@+id/card_view"
    android:layout_height="wrap_content"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_margin="8dp"
    app:cardCornerRadius="22dp"
    app:cardElevation="4dp"
    android:layout_gravity="center"
    xmlns:app="http://schemas.android.com/apk/res-
auto"

xmlns:android="http://schemas.android.com/apk/res/an
droid">


<androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="8dp">

        <ImageView
            android:id="@+id/agent_image"
            android:layout_width="150dp"
            android:layout_height="200dp"
            android:scaleType="centerCrop"

app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintHorizontal_bias="0.0"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.0"
            tools:src="@tools:sample/avatars" />
```

```
39
40          <TextView
41              android:id="@+id/agent_name"
42              android:layout_width="120dp"
43              android:layout_height="wrap_content"
44              android:layout_marginStart="12dp"
45              android:textSize="13sp"
46              android:textStyle="bold"
47
48  app:layout_constraintEnd_toEndOf="parent"
49
50  app:layout_constraintHorizontal_bias="0.0"
51
52  app:layout_constraintStart_toEndOf="@+id/agent_image
53  "
54
55  app:layout_constraintTop_toTopOf="parent"
56
57  app:layout_constraintVertical_bias="0.082"
58              tools:text="judul film" />
59
60          <TextView
61              android:id="@+id/agent_role"
62              android:layout_width="wrap_content"
63              android:layout_height="wrap_content"
64              android:textSize="13sp"
65              android:textStyle="bold"
66
67  app:layout_constraintBottom_toTopOf="@+id/agent_desc
68  "
69
70  app:layout_constraintEnd_toEndOf="parent"
71
72  app:layout_constraintHorizontal_bias="0.75"
73
74  app:layout_constraintStart_toEndOf="@+id/agent_name"
75
76  app:layout_constraintTop_toTopOf="@+id/agent_name"
77              app:layout_constraintVertical_bias="0.0"
78              tools:ignore="MissingConstraints"
79              tools:text="Controller" />
80
81          <TextView
82              android:id="@+id/agent_desc"
              android:layout_marginStart="10dp"
```

```
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:textSize="10sp"

app:layout_constraintBottom_toTopOf="@+id/btn_link"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintHorizontal_bias="0.266"

app:layout_constraintStart_toEndOf="@+id/agent_image
"

app:layout_constraintTop_toBottomOf="@+id/agent_name
"
            app:layout_constraintVertical_bias="0.2"
            tools:text="hab" />

        <Button
            android:id="@+id/btn_link"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_link"
            android:textSize="10sp"

    app:layout_constraintBottom_toBottomOf="parent"

    app:layout_constraintEnd_toStartOf="@+id/btn_detail"

    app:layout_constraintHorizontal_bias="0.609"

    app:layout_constraintStart_toEndOf="@+id/agent_image
    "

    app:layout_constraintTop_toTopOf="parent"

    app:layout_constraintVertical_bias="0.947" />

        <Button
            android:id="@+id/btn_detail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

    app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent"

    app:layout_constraintHorizontal_bias="1.0"

    app:layout_constraintStart_toEndOf="@+id/agent_image
    "

    app:layout_constraintTop_toTopOf="parent"

    app:layout_constraintVertical_bias="0.947"
                android:textSize="10sp"
                android:text="@string/btn_detail" />


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

strings.xml

Table 12. Source Code Jawaban Soal 1 XML

```
1  <resources>
2      <string name="app_name">XML</string>
3      <string name="btn_detail">Deskripsi</string>
4      <string name="btn_link">Detail</string>
5
6      <string-array name="data_agentName">
7          <item>Brimstone</item>
8          <item>Phoenix</item>
9          <item>Chamber</item>
10         <item>Jett</item>
11         <item>Kay/o</item>
12         <item>Viper</item>
13     </string-array>
14
15     <string-array name="data_agentRole">
16         <item>Controller</item>
17         <item>Duelist</item>
18         <item>Sentinel</item>
19         <item>Duelist</item>
20         <item>Initiator</item>
21         <item>Controller</item>
22     </string-array>
23
24     <string-array name="data_agentDesc">
           <item>Joining from the U.S.A., Brimstone`s
```

```
orbital arsenal ensures his squad always has the
advantage. His ability to deliver utility precisely
and safely make him the unmatched boots-on-the-
ground commander.</item>
        <item>Hailing from the U.K., Phoenix`s star
power shines through in his fighting style, igniting
the battlefield with flash and flare. Whether he`s
got backup or not, he`ll rush into a fight on his
own terms.</item>
        <item>Well-dressed and well-armed, French
weapons designer Chamber expels aggressors with
deadly precision. He leverages his custom arsenal to
hold the line and pick off enemies from afar, with a
contingency built for every plan.</item>
        <item>Representing her home country of South
Korea, Jett`s agile and evasive fighting style lets
her take risks no one else can. She runs circles
around every skirmish, cutting enemies before they
even know what hit them.</item>
        <item>KAY/O is a machine of war built for a
single purpose: neutralizing radiants. His power to
Suppress enemy abilities dismantles his opponents`
capacity to fight back, securing him and his allies
the ultimate edge.</item>
        <item>The American Chemist, Viper deploys an
array of poisonous chemical devices to control the
battlefield and choke the enemy`s vision. If the
toxins don`t kill her prey, her mindgames surely
will.</item>
    </string-array>

    <string-array name="data_agentImage">
        <item>@drawable/brimstone</item>
        <item>@drawable/phoenix</item>
        <item>@drawable/chamber</item>
        <item>@drawable/jett</item>
        <item>@drawable/kayo</item>
        <item>@drawable/viper</item>
    </string-array>

    <string-array name="data_agentLink">
        <item>https://playvalorant.com/en-
us/agents/brimstone/</item>
        <item>https://playvalorant.com/en-
us/agents/phoenix/</item>
```

```
        <item>https://playvalorant.com/en-
us/agents/chamber/</item>
        <item>https://playvalorant.com/en-
us/agents/jett/</item>
        <item>https://playvalorant.com/en-
us/agents/kay-o/</item>
        <item>https://playvalorant.com/en-
us/agents/viper/</item>
    </string-array>

    <string name="hello_blank_fragment">Hello blank
fragment</string>
</resources>
```

Compose:

MainActivity.kt

Table 13. Source Code MainActivity Compose

```
1  package com.example.listcompose
2
3  import android.os.Bundle
4  import androidx.activity.ComponentActivity
5  import androidx.activity.compose.setContent
6  import androidx.activity.enableEdgeToEdge
7  import androidx.activity.viewModels
8  import androidx.navigation.compose.*
9  import
10 com.example.listcompose.ui.screen.AgentListScreen
11 import
12 com.example.listcompose.ui.screen.DeskripsiScreen
13 import
14 com.example.listcompose.viewmodel.AgentViewModel
15 import
16 com.example.listcompose.viewmodel.AgentViewModelFac
17 tory
18 import
19 com.example.listcompose.ui.theme.ComposeTheme
20 import timber.log.Timber
21
22 class MainActivity : ComponentActivity() {
23     private val viewModel: AgentViewModel by
24 viewModels {
```

```
        AgentViewModelFactory("Default Param")
    }

    override fun onCreate(savedInstanceState:
Bundle?) {
        super.onCreate(savedInstanceState)
        Timber.plant(Timber.DebugTree())
        enableEdgeToEdge()
        setContent {
            ComposeTheme {
                val navController =
rememberNavController()
                NavHost(navController =
navController, startDestination = "agentList") {
                    composable("agentList") {

AgentListScreen(navController = navController,
viewModel = viewModel)
                    }
                    composable("deskripsi") {
                        DeskripsiScreen(viewModel =
viewModel)
                    }
                }
            }
        }
    }
}
```

DataAgent

Table 14. Source Code DataAgent Compose

```
package com.example.listcompose.data.model

data class DataAgent(
    val name: String,
    val image: Int,
    val url: String,
    val description: String
)
```

AgentList.kt

Table 15. Source Code AgentList.kt Compose

```
1  package com.example.listcompose.data.dummy
2
3  import com.example.listcompose.R
4  import com.example.listcompose.data.model.DataAgent
5
6  val AgentList = listOf(
7      DataAgent(
8          "Jett", R.drawable.jett,
9  "https://playvalorant.com/en-us/agents/jett/",
10         "Role: Duelist\nRepresenting her home
11 country of South Korea, Jett's agile and evasive
12 fighting style lets her take risks no one else can.
13 She runs circles around every skirmish, cutting
14 enemies before they even know what hit them."
15     ),
16     DataAgent(
17         "Brimstone", R.drawable.brimstone,
18 "https://playvalorant.com/en-us/agents/brimstone/",
19         "Role: Controller\nJoining from the U.S.A.,
20 Brimstone's orbital arsenal ensures his squad always
21 has the advantage. His ability to deliver utility
22 precisely and safely make him the unmatched boots-
23 on-the-ground commander."
24     ),
25     DataAgent(
26         "Phoenix", R.drawable.phoenix,
27 "https://playvalorant.com/en-us/agents/phoenix/",
28         "Role: Duelist\nHailing from the U.K.,
29 Phoenix's star power shines through in his fighting
30 style, igniting the battlefield with flash and
31 flare. Whether he's got backup or not, he'll rush
32 into a fight on his own terms."
       ),
       DataAgent(
           "Chamber", R.drawable.chamber,
   "https://playvalorant.com/en-us/agents/chamber/",
           "Role: Sentinel\nWell-dressed and well-
   armed, French weapons designer Chamber expels
   aggressors with deadly precision. He leverages his
   custom arsenal to hold the line and pick off enemies
   from afar, with a contingency built for every plan."
       ),
       DataAgent(
           "Viper", R.drawable.viper,
   "https://playvalorant.com/en-us/agents/viper/",
```

```
        "Role: Controller\nThe American Chemist,
Viper deploys an array of poisonous chemical devices
to control the battlefield and choke the enemy's
vision. If the toxins don't kill her prey, her
mindgames surely will."
    ),
    DataAgent(
        "KAY/O", R.drawable.kayo,
"https://playvalorant.com/en-us/agents/kay-o/",
        "Role: Innitiatot\nKAY/O is a machine of war
built for a single purpose: neutralizing radiants.
His power to Suppress enemy abilities dismantles his
opponents' capacity to fight back, securing him and
his allies the ultimate edge."
    ),
)
```

AgentItem.kt

Table 16. Source Code AgentItem.kt Compose

```
1  package com.example.listcompose.ui.component
2
3  import android.content.Intent
4  import android.net.Uri
5  import androidx.compose.foundation.Image
6  import androidx.compose.foundation.layout.*
7  import
8  androidx.compose.foundation.shape.RoundedCornerShape
9  import androidx.compose.material3.*
10 import androidx.compose.runtime.Composable
11 import androidx.compose.ui.Modifier
12 import androidx.compose.ui.platform.LocalContext
13 import androidx.compose.ui.res.painterResource
14 import androidx.compose.ui.text.font.FontWeight
15 import androidx.compose.ui.text.style.TextOverflow
16 import androidx.compose.ui.unit.dp
17 import androidx.compose.ui.unit.sp
18 import androidx.navigation.NavHostController
19 import com.example.listcompose.data.model.DataAgent
20 import
21 com.example.listcompose.viewmodel.AgentViewModel
22 import timber.log.Timber
23
24 @Composable
```

```
fun AgentItem(
    agent: DataAgent,
    navController: NavHostController,
    viewModel: AgentViewModel
) {
    val context = LocalContext.current

    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        shape = RoundedCornerShape(16.dp),
        elevation = CardDefaults.cardElevation(4.dp)
    ) {
        Row(modifier = Modifier.padding(16.dp)) {
            Image(
                painter = painterResource(id =
agent.image),
                contentDescription = null,
                modifier = Modifier
                    .size(100.dp)
            )
            Spacer(modifier = Modifier.width(16.dp))
            Column(modifier = Modifier.weight(1f)) {
                Text(agent.name, fontWeight =
FontWeight.Bold, fontSize = 20.sp)
                Spacer(modifier =
Modifier.height(4.dp))
                Text(
                    agent.description,
                    fontSize = 14.sp,
                    maxLines = 3,
                    overflow = TextOverflow.Ellipsis
                )
                Spacer(modifier =
Modifier.height(8.dp))
                Row {
                    Button(onClick = {

viewModel.onDetailClick(agent)

context.startActivity(Intent(Intent.ACTION_VIEW,
Uri.parse(agent.url)))
                    }) { Text("Detail", fontSize =
12.sp) }
```

```
                    Spacer(modifier =
Modifier.width(8.dp))
                    Button(onClick = {

viewModel.onDeskripsiClick(agent)

navController.navigate("deskripsi")
                    }) { Text("Deskripsi", fontSize
= 12.sp) }
                }
            }
        }
    }
}
```

AgentListScreen.kt

Table 17. Source Code AgentListScreen.kt Compose

```
1  package com.example.listcompose.ui.screen
2
3  import androidx.compose.foundation.layout.*
4  import androidx.compose.foundation.lazy.LazyColumn
5  import androidx.compose.runtime.Composable
6  import androidx.compose.ui.Modifier
7  import androidx.compose.ui.unit.dp
8  import androidx.navigation.NavHostController
9  import
10 com.example.listcompose.ui.component.AgentItem
11 import
12 com.example.listcompose.viewmodel.AgentViewModel
13
14 @Composable
15 fun AgentListScreen(navController:
16 NavHostController, viewModel: AgentViewModel) {
17     LazyColumn(
18         modifier = Modifier
19             .fillMaxSize()
20             .padding(start = 8.dp, end = 8.dp)
21
22 .windowInsetsPadding(WindowInsets.statusBars)
23
24     ) {
25         items(viewModel.agentList.size) { index ->
26             val agent = viewModel.agentList[index]
```

```
            AgentItem(agent = agent, navController
= navController, viewModel = viewModel)
        }
    }
}
```

DeskripsiScreen.kt

Table 18. Source Code DeskripsiScreen.kt Compose

```
1  package com.example.listcompose.ui.screen
2
3  import androidx.compose.foundation.Image
4  import androidx.compose.foundation.layout.*
5  import
6  androidx.compose.foundation.rememberScrollState
7  import androidx.compose.foundation.verticalScroll
8  import androidx.compose.material3.Text
9  import androidx.compose.runtime.*
10 import androidx.compose.ui.Alignment
11 import androidx.compose.ui.Modifier
12 import androidx.compose.ui.res.painterResource
13 import androidx.compose.ui.unit.dp
14 import androidx.compose.ui.unit.sp
15 import
16 com.example.listcompose.viewmodel.AgentViewModel
17 import androidx.compose.runtime.collectAsState
18 import androidx.compose.runtime.getValue
19 import androidx.compose.ui.text.style.TextAlign
20
21 @Composable
22 fun DeskripsiScreen(viewModel: AgentViewModel) {
23     val selectedAgent by
24 viewModel.selectedAgent.collectAsState()
25
26     selectedAgent?.let { agent ->
27         Column(
28             modifier = Modifier
29                 .fillMaxSize()
30                 .padding(8.dp)
31
32 .windowInsetsPadding(WindowInsets.statusBars)
33
34 .verticalScroll(rememberScrollState()),
35             horizontalAlignment =
```

```
Alignment.CenterHorizontally
        ) {
            Image(
                painter = painterResource(id =
agent.image),
                contentDescription = null,
                modifier = Modifier
                    .fillMaxWidth()
                    .height(400.dp)
            )
            Spacer(modifier =
Modifier.height(16.dp))
            Text(
                text = agent.description,
                fontSize = 14.sp,
                modifier = Modifier.fillMaxWidth(),
                textAlign = TextAlign.Start
            )
        }
    }
}
```

AgentViewModel

Table 19. Source Code AgentViewModel Compose

```
package com.example.listcompose.viewmodel

import androidx.lifecycle.ViewModel
import com.example.listcompose.data.dummy.AgentList
import com.example.listcompose.data.model.DataAgent
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import timber.log.Timber

class AgentViewModel(param: String) : ViewModel() {

    val agentList = AgentList

    private val _selectedAgent =
MutableStateFlow<DataAgent?>(null)
    val selectedAgent: StateFlow<DataAgent?> =
_selectedAgent

    init {
```

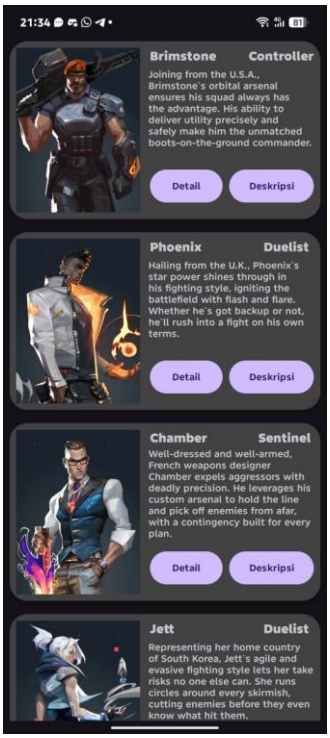
```
        Timber.d("ViewModel created with param:
$param")
        AgentList.forEach {
            Timber.d("Loaded agent: ${it.name}")
        }
    }

    fun selectAgent(agent: DataAgent) {
        Timber.d("Agent selected: ${agent.name}")
        _selectedAgent.value = agent
    }

    fun onDetailClick(agent: DataAgent) {
        Timber.d("Detail button clicked:
${agent.name}")
    }

    fun onDeskripsiClick(agent: DataAgent) {
        Timber.d("Deskripsi button clicked:
${agent.name}")
        selectAgent(agent)
    }

}
```

AgentViewModelFactory

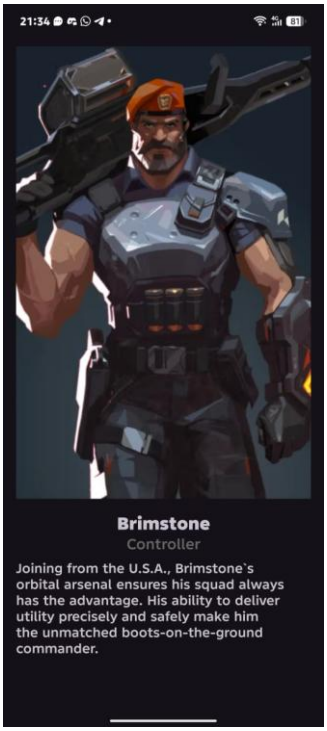
Table 20. Source Code AgentViewModelFactory Compose

```
package com.example.listcompose.viewmodel

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

@Suppress("UNCHECKED_CAST")
class AgentViewModelFactory(private val param:
String) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass:
Class<T>): T {
        return AgentViewModel(param) as T
    }
}
```

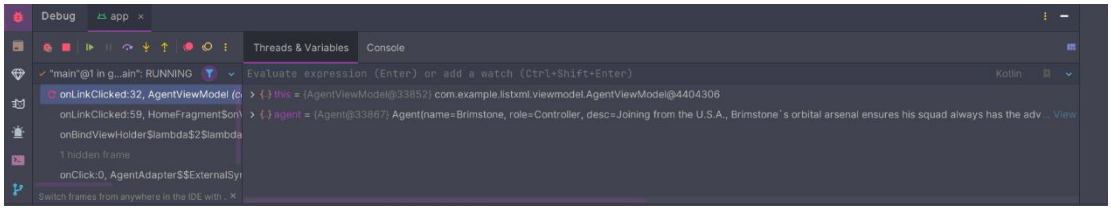
B. Output Program



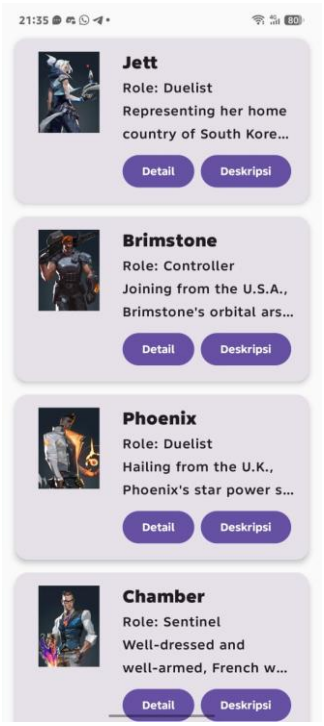
Gambar 2. Tampilan Awal Aplikasi XML



Gambar 3. Tampilan Deskripsi XML



Gambar 4. Tampilan Debug XML



Gambar 5. Tampilan Awal Aplikasi Compose

Gambar 6. Tampilan Detail XML

Gambar 7. Tampilan Debug Compose

C. Pembahasan

XML:

MainActivity:

Kode di atas merupakan implementasi dari kelas MainActivity dalam proyek Android berbasis XML yang menggunakan arsitektur MVVM dan Timber untuk logging. MainActivity mewarisi AppCompatActivity dan berfungsi sebagai titik masuk utama aplikasi. Di dalam metode onCreate(), layout activity_main.xml diatur sebagai tampilan utama dengan memanggil setContentView(). Untuk kebutuhan logging, Timber ditanamkan melalui Timber.plant(Timber.DebugTree()), yang memungkinkan pencatatan log saat pengembangan. Pesan log informasi juga ditambahkan menggunakan Timber.i() untuk menandai bahwa MainActivity telah dibuat. Selain itu,

ketika aktivitas pertama kali dimuat (yaitu savedInstanceState bernilai null), fragment HomeFragment dimuat ke dalam FrameLayout dengan ID frame_container menggunakan supportFragmentManager.

HomeFragment.kt:

Kode di atas mendefinisikan HomeFragment, sebuah fragment dalam aplikasi Android yang bertugas menampilkan daftar agen menggunakan arsitektur MVVM. Fragment ini menggunakan View Binding (FragmentHomeBinding) untuk mengakses elemen-elemen antarmuka pengguna secara langsung dan aman dari kesalahan penulisan ID. ViewModel AgentViewModel diambil dengan delegasi activityViewModels, menggunakan AgentViewModelFactory untuk menyuplai parameter yang dibutuhkan. Pada onViewCreated, daftar agen diambil dari resource aplikasi menggunakan fungsi getListAgent() dan kemudian dikirim ke ViewModel. Data agen diamati melalui StateFlow menggunakan ekstensi collectIn(), yang memastikan koleksi hanya terjadi saat lifecycle berada dalam kondisi STARTED atau lebih. Ketika daftar agen berubah, AgentAdapter akan dibuat dan diatur ke RecyclerView untuk menampilkan data. Adapter ini juga menangani dua aksi klik: membuka detail agen dalam fragment baru (DetailFragment), dan membuka tautan eksternal melalui Intent. Fungsi getListAgent() membaca array data dari resource (nama, deskripsi, peran, gambar, dan tautan), membungkusnya ke dalam objek Agent, lalu mengembalikannya sebagai daftar. Di akhir siklus tampilan (onDestroyView), binding dibersihkan untuk mencegah memory leak.

DetailFragment:

Kode di atas merupakan implementasi dari DetailFragment, yaitu fragment yang digunakan untuk menampilkan detail informasi tentang agen yang dipilih dalam aplikasi Android. Fragment ini menggunakan FragmentDetailBinding untuk mengakses dan menampilkan data ke antarmuka pengguna secara langsung dan aman. ViewModel AgentViewModel diakses menggunakan activityViewModels agar berbagi data dengan fragment lainnya, khususnya HomeFragment, dengan bantuan

AgentViewModelFactory yang menginisialisasi ViewModel dengan parameter "Agent Detail". Pada siklus onCreateView(), binding diinisialisasi dengan layout fragment_detail.xml, lalu pada onViewCreated(), fragment mengambil data agen yang dipilih melalui properti selectedAgent dari ViewModel. Jika data agen tersedia, maka informasi seperti nama, peran, deskripsi, dan gambar ditampilkan ke UI. Proses ini juga disertai logging menggunakan Timber (Timber.i) untuk mencatat bahwa navigasi ke halaman detail telah dilakukan beserta data agen terkait. Akhirnya, pada onDestroyView(), binding dihapus dengan menyetel _binding = null untuk menghindari kebocoran memori.

AgentAdapter:

Kode di atas merupakan implementasi dari kelas AgentAdapter, sebuah adapter untuk RecyclerView yang digunakan untuk menampilkan daftar agen dalam tampilan berbasis XML. Adapter ini menerima dua parameter utama: agentList, yaitu daftar objek Agent yang akan ditampilkan, dan listener, yaitu antarmuka OnAgentClickListener untuk menangani interaksi pengguna pada setiap item. Di dalam AgentAdapter, terdapat inner class AgentViewHolder yang menyimpan objek ItemAgentBinding, digunakan untuk mengikat data dengan tampilan item_agent.xml. Fungsi onCreateViewHolder() akan membuat dan mengembalikan AgentViewHolder baru dengan cara melakukan inflasi layout dari file binding. Kemudian, onBindViewHolder() akan mem-bind data dari setiap agen ke tampilan masing-masing, seperti nama, peran, deskripsi, dan gambar agen. Dua tombol juga disediakan: btnDetail untuk menampilkan detail agen, dan btnLink untuk membuka tautan terkait agen. Kedua aksi ini didelegasikan ke listener agar logika penanganan tetap dipisahkan dari adapter. Fungsi getItemCount() mengembalikan jumlah total item dalam agentList, yang menentukan berapa banyak item yang akan ditampilkan di RecyclerView.

Agent:

Kode di atas mendefinisikan data class Agent dalam paket com.example.listxml.data.model, yang merepresentasikan entitas agen dalam aplikasi.

Kelas ini menggunakan anotasi @Parcelize, yang secara otomatis mengimplementasikan antarmuka Parcelable, memungkinkan objek Agent untuk dikirim antar komponen Android seperti Fragment atau Activity melalui Bundle. Properti dalam kelas ini mencakup:

- name: nama agen,
- role: peran atau spesialisasi agen,
- desc: deskripsi agen,
- image: ID resource gambar agen (bertipe Int),
- detail: tautan atau informasi tambahan agen (bernilai default string kosong jika tidak disediakan).

Penggunaan @Parcelize menyederhanakan serialisasi data dibandingkan implementasi manual Parcelable, dan sangat berguna dalam konteks navigasi fragment atau pengiriman data antar layar.

AgentViewModel:

Kode di atas mendefinisikan AgentViewModel, sebuah kelas ViewModel yang bertanggung jawab mengelola dan menyediakan data agen kepada UI secara reaktif dalam arsitektur MVVM. Kelas ini menerima parameter title untuk tujuan pencatatan (logging) dan mencatat pesan saat ViewModel dibuat menggunakan Timber. ViewModel ini memiliki dua properti StateFlow utama:

- _agentList: MutableStateFlow yang menyimpan daftar agen, dan diakses secara publik melalui agentList sebagai StateFlow immutable.
- _selectedAgent: MutableStateFlow yang menyimpan agen yang sedang dipilih, dan diakses melalui selectedAgent sebagai StateFlow immutable.

Fungsi setAgentList() digunakan untuk mengatur daftar agen dan mencatat jumlah data yang dimasukkan. Fungsi selectAgent() digunakan saat pengguna memilih agen untuk melihat detailnya, yang juga tercatat melalui Timber. Sedangkan onLinkClicked() menangani aksi ketika tautan pada agen diklik, dan mencatat informasi nama agen beserta detail link-nya. Penggunaan StateFlow memastikan bahwa setiap perubahan

data akan diamati secara reaktif oleh UI yang mengikatnya, tanpa perlu manual menyinkronkan data antar komponen.

AgentViewModelFactory:

Kode di atas merupakan implementasi dari AgentViewModelFactory, sebuah kelas yang mengimplementasikan antarmuka ViewModelProvider.Factory untuk membuat instance AgentViewModel dengan parameter tambahan, yaitu title. Secara default, ViewModelProvider hanya dapat membuat ViewModel yang memiliki konstruktor tanpa parameter. Oleh karena itu, ketika AgentViewModel memerlukan parameter title pada konstruktornya, maka kita membutuhkan ViewModelFactory untuk menyuplai nilai tersebut saat pembuatan. Dalam metode create(), dicek apakah modelClass merupakan turunan dari AgentViewModel menggunakan isAssignableFrom(). Jika ya, maka AgentViewModel(title) dibuat dan dikembalikan setelah dilakukan casting ke tipe T. Casting ini diperingatkan oleh compiler sebagai tidak aman, sehingga anotasi @Suppress("UNCHECKED_CAST") digunakan untuk menghindari warning. Jika tipe modelClass tidak dikenali, maka dilemparkan IllegalArgumentException.

activity_main.xml:

Kode XML ini mendefinisikan sebuah FrameLayout yang berfungsi sebagai wadah utama dalam layout aktivitas (activity_main.xml) pada aplikasi Android. FrameLayout ini didesain agar memenuhi seluruh lebar dan tinggi layar perangkat dengan atribut match_parent pada lebar dan tingginya. Layout ini juga diberikan ID frame_container, yang memungkinkan kode program untuk mengakses dan memanipulasinya, misalnya untuk mengganti fragment secara dinamis melalui FragmentManager. Selain itu, atribut android:fitsSystemWindows="true" digunakan agar tata letak ini dapat menyesuaikan dirinya dengan area tampilan yang tersedia, memperhitungkan elemen-elemen sistem seperti status bar atau navigasi, sehingga konten tidak tertutup oleh komponen sistem tersebut. Penetapan atribut tools:context membantu Android Studio untuk menampilkan preview layout ini dalam konteks MainActivity.

fragment_detail.xml:

Layout XML ini mendesain tampilan detail untuk sebuah fragment (DetailFragment) menggunakan ConstraintLayout sebagai root dengan padding 8dp di seluruh sisi dan ukuran yang memenuhi layar. Di dalamnya terdapat sebuah ScrollView agar konten dapat digulir jika melebihi tinggi layar, yang kemudian membungkus ConstraintLayout kedua dengan tinggi wrap_content untuk menyesuaikan isi. Di dalam ConstraintLayout kedua ini terdapat beberapa komponen UI utama: sebuah ImageView dengan ID detail_image yang menampilkan gambar agen dengan mode centerCrop, yang mengisi lebar penuh dan menyesuaikan tinggi berdasarkan isi layout. Di bawahnya terdapat tiga TextView, yaitu detail_name yang menampilkan nama agen dengan teks tebal dan ukuran font 20sp, detail_role yang menampilkan peran agen dengan gaya teks miring dan warna abu-abu, serta detail_desc yang menampilkan deskripsi agen dengan ukuran teks lebih kecil dan lebar yang memenuhi batas layout. Setiap komponen diatur posisinya secara fleksibel menggunakan constraint untuk menjaga jarak dan perataan antar elemen, memastikan tampilan responsif dan rapi. Penggunaan tools: pada beberapa atribut menyediakan contoh isi untuk preview di Android Studio, seperti gambar avatar dan teks contoh, tanpa mempengaruhi runtime.

fragment_home.xml:

Layout XML ini mendefinisikan sebuah FrameLayout sebagai wadah utama untuk fragment HomeFragment. FrameLayout ini dirancang agar mengisi seluruh lebar dan tinggi layar dengan atribut match_parent. Di dalamnya terdapat sebuah RecyclerView yang juga memenuhi seluruh ruang dari FrameLayout, berfungsi untuk menampilkan daftar item secara efisien dan dapat digulir. RecyclerView ini memiliki ID rvAgent, sehingga dapat diakses dari kode program untuk mengatur adapter dan layout manager yang mengelola tampilan daftar agen.

item_agent.xml:

Layout XML ini mendefinisikan tampilan sebuah item kartu (CardView) yang biasanya digunakan di dalam daftar seperti RecyclerView untuk menampilkan

informasi tentang agen. CardView tersebut memiliki lebar penuh (match_parent), tinggi yang menyesuaikan isi (wrap_content), sudut membulat dengan radius 22dp, dan elevasi 4dp yang memberikan efek bayangan agar tampak mengambang di atas latar belakang. Margin 8dp memberikan jarak antar kartu agar tidak saling berhimpitan. Di dalam CardView terdapat ConstraintLayout dengan padding 8dp yang berfungsi mengatur posisi elemen-elemen secara fleksibel dan responsif. Komponen pertama adalah ImageView berukuran 150dp x 200dp yang menampilkan gambar agen dengan mode centerCrop agar gambar terpotong rapi sesuai ukuran. Di samping gambar, ada tiga TextView yang menampilkan nama agen (agent_name), perannya (agent_role), dan deskripsi singkat (agent_desc) dengan ukuran font kecil dan penataan posisi sesuai constraint yang menghubungkan antar elemen. Di bagian bawah sebelah kanan terdapat dua tombol (Button), yakni btn_link dan btn_detail. Tombol btn_link berfungsi untuk membuka tautan terkait agen, sementara btn_detail digunakan untuk menampilkan detail lebih lanjut dari agen tersebut. Penempatan tombol dan teks diatur sedemikian rupa agar tata letak terlihat rapi dan proporsional dalam satu baris.

strings.xml:

File XML ini berfungsi sebagai sumber data string yang digunakan dalam aplikasi Android untuk menampilkan informasi tentang berbagai agen. Di dalamnya terdapat beberapa string dasar seperti nama aplikasi dan label tombol yang akan muncul pada antarmuka pengguna, yaitu tombol "Deskripsi" dan "Detail". Selain itu, terdapat beberapa array string yang berisi data statis, termasuk nama agen, peran mereka, deskripsi lengkap yang menjelaskan latar belakang dan kemampuan masing-masing agen, serta referensi gambar yang digunakan untuk menampilkan foto agen di aplikasi. Tidak hanya itu, file ini juga menyimpan URL resmi masing-masing agen yang dapat digunakan untuk mengarahkan pengguna ke halaman web terkait.

Compose:

MainActivity:

Kode MainActivity ini merupakan entry point dari aplikasi Android yang dibangun menggunakan Jetpack Compose. Di sini, MainActivity mewarisi dari ComponentActivity dan menggunakan viewModels untuk mendapatkan instance AgentViewModel dengan parameter default melalui AgentViewModelFactory. Pada metode onCreate, aplikasi menyiapkan logging dengan Timber.DebugTree() untuk debugging, lalu mengaktifkan fitur edge-to-edge agar konten tampil memenuhi layar tanpa batasan sistem UI. Selanjutnya, setContent digunakan untuk mengatur UI Compose. Dalam ComposeTheme, dibuat sebuah NavHost yang mengelola navigasi antar layar (screen) menggunakan NavController. Navigasi ini dimulai dari layar dengan rute "agentList", yang menampilkan AgentListScreen dan memberikan akses ke navController serta viewModel yang sama untuk berbagi data. Kemudian ada rute "deskripsi" yang menampilkan DeskripsiScreen juga menggunakan viewModel yang sama.

DataAgent:

Kode DataAgent ini adalah sebuah data class sederhana di Kotlin yang digunakan untuk merepresentasikan data agen dalam aplikasi Compose Anda. Kelas ini memiliki empat properti: name (nama agen, bertipe String), image (referensi gambar agen, bertipe Int, biasanya merujuk ke resource drawable), url (link atau alamat web terkait agen, bertipe String), dan description (deskripsi atau penjelasan tentang agen, bertipe String).

AgentList.kt:

Kode AgentList ini adalah daftar dummy (contoh data) berupa list dari objek DataAgent yang digunakan untuk mengisi data agen dalam aplikasi Jetpack Compose Anda. Setiap item DataAgent berisi nama agen, referensi gambar (drawable resource), URL yang mengarah ke halaman resmi agen, serta deskripsi lengkap termasuk peran (role) dan latar belakangnya.

AgentItem.kt:

Kode AgentItem ini adalah composable function di Jetpack Compose yang merepresentasikan tampilan sebuah item agen dalam daftar. Fungsi ini menerima tiga parameter: data agen (agent), controller navigasi (navController), dan viewModel untuk pengelolaan data serta event.

Di dalam fungsi, LocalContext.current digunakan untuk mendapatkan konteks aplikasi agar bisa melakukan aksi seperti membuka link lewat intent.

UI utama dibungkus dengan Card bercornerradius 16dp dan elevation 4dp untuk efek bayangan, serta padding 8dp. Di dalam Card, sebuah Row digunakan untuk meletakkan gambar agen di sebelah kiri dengan ukuran 100dp persegi. Di samping gambar, ada Column yang berisi nama agen (ditampilkan dengan font tebal dan ukuran 20sp), deskripsi singkat agen yang dibatasi hanya 3 baris dengan teks yang dipotong jika terlalu panjang (ellipsis), dan dua tombol aksi.

Dua tombol tersebut punya fungsi berbeda:

- Tombol **Detail** akan memanggil fungsi onDetailClick di viewModel lalu membuka URL agen di browser menggunakan Intent.ACTION_VIEW.
- Tombol **Deskripsi** akan memanggil onDeskripsiClick di viewModel dan kemudian melakukan navigasi ke layar deskripsi (deskripsi) menggunakan navController.

AgentListScreen.kt:

Kode AgentListScreen ini adalah sebuah composable function di Jetpack Compose yang menampilkan daftar agen menggunakan LazyColumn. Fungsi ini menerima navController untuk navigasi antar layar dan viewModel untuk mengambil data daftar agen. LazyColumn digunakan untuk menampilkan daftar scrollable yang hanya me-render item yang terlihat di layar saja, sehingga efisien untuk performa. Modifier-nya mengatur agar daftar memenuhi seluruh ukuran layar (fillMaxSize()), dengan padding di sisi kiri dan kanan sebesar 8dp, serta memperhitungkan inset status bar agar tampilan tidak tertutup. Setiap item di dalam LazyColumn dibuat dengan mengakses list agentList dari viewModel berdasarkan indeks. Kemudian setiap agent tersebut diteruskan ke AgentItem composable yang bertugas menampilkan konten visual

masing-masing agen. Di sini juga diteruskan navController dan viewModel agar item bisa menangani event seperti klik dan navigasi ke detail.

DeskripsiScreen.kt:

Fungsi DeskripsiScreen adalah sebuah composable di Jetpack Compose yang menampilkan detail deskripsi lengkap dari agen yang dipilih. Di dalam fungsi ini, selectedAgent diambil dari viewModel menggunakan collectAsState() untuk mengamati perubahan data secara reaktif. Jika ada agen yang dipilih (selectedAgent tidak null), maka tampilan akan menampilkan sebuah kolom (Column) yang mengisi seluruh layar dengan padding 8dp dan menyesuaikan dengan status bar menggunakan windowInsetsPadding. Kolom ini juga bisa di-scroll secara vertikal dengan verticalScroll sehingga konten deskripsi yang panjang dapat dibaca dengan mudah. Di dalam kolom tersebut, gambar agen ditampilkan secara penuh dengan lebar layar dan tinggi 400dp di bagian atas, diikuti oleh spasi 16dp, lalu teks deskripsi lengkap agen dengan ukuran font 14sp yang diratakan ke kiri.

AgentViewModel:

Kelas AgentViewModel adalah sebuah ViewModel yang mengelola data dan logika bisnis untuk daftar agen dalam aplikasi. ViewModel ini menerima sebuah parameter string saat dibuat, yang dicatat menggunakan Timber untuk tujuan debugging. ViewModel menyimpan daftar agen dalam properti agentList yang diambil dari data dummy AgentList. Untuk mengelola agen yang sedang dipilih, ViewModel menggunakan MutableStateFlow bernama _selectedAgent yang diinisialisasi dengan nilai null, dan mengeksposnya sebagai StateFlow yang bersifat hanya-baca bernama selectedAgent. Pada inisialisasi, ViewModel juga mencatat semua nama agen yang dimuat menggunakan Timber. Fungsi selectAgent digunakan untuk mengubah agen yang sedang dipilih dan mencatat aksi ini. Selain itu, ada dua fungsi yang menangani interaksi pengguna pada tombol di UI: onDetailClick yang mencatat klik pada tombol detail tanpa mengubah state, dan onDeskripsiClick yang mencatat klik pada tombol

deskripsi sekaligus memanggil fungsi selectAgent untuk memperbarui agen yang dipilih.

ViewModelFactory:

Kelas AgentViewModelFactory adalah sebuah pabrik (factory) untuk membuat instance AgentViewModel dengan cara yang benar dan terkontrol. Karena AgentViewModel membutuhkan parameter (param: String) saat pembuatannya, kita tidak bisa langsung membuatnya tanpa factory ini saat menggunakan ViewModelProvider di Android. AgentViewModelFactory mengimplementasikan interface ViewModelProvider.Factory dan mengoverride metode create, yang bertugas mengembalikan instance ViewModel sesuai kelas yang diminta. Dalam metode create, factory ini mengembalikan objek AgentViewModel dengan mengoper parameter yang dibutuhkan, lalu melakukan cast ke tipe generik T. Anotasi @Suppress("UNCHECKED_CAST") digunakan untuk menghilangkan peringatan cast yang tidak bisa diverifikasi secara statis oleh compiler.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/MadeByBintang/PraktikumMobile