

**LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4**



VIEWMODEL AND DEBUGGING

Oleh:

Muhammad Daffa Musyafa NIM. 2310817110007

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

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN I
MODUL 4

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

Nama Praktikan : Muhammad Daffa Musyafa
NIM : 2310817210007

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Muhammad Raka Azwar
NIM. 2210817210012

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

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI	3
DAFTAR GAMBAR.....	4
DAFTAR TABEL	5
SOAL 1.....	6
A. Source Code XML.....	7
B. Output Program	19
C. Pembahasan	21
MainActivity.kt:	21
HeroML.kt.....	22
HeroMLAdapter.kt	23
HomeFragment.kt.....	24
DetailFragment.kt	26
Activity_main.xml.....	30
Item_char_ml.xml	31
fragment_detail.xml.....	32
A. Source Code Compose	33
B. Output Program	43
C. Pembahasan	44
MainActivity.kt:	44
DataHero.kt	47
HeroData.kt	47
HomeScreen.kt	49
DetailScreen.kt	51
HeroViewModel.kt.....	52
HeroViewModelFactory.kt.....	54
Soal 2	55
Tautan Git	55

DAFTAR GAMBAR

Gambar 1 Contoh Penggunaan Debugger	6
Gambar 2. Screenshot Hasil Jawaban Soal 1 XML	19
Gambar 3. Screenshot Hasil Jawaban Soal 1 XML	20
Gambar 4. Timber Button.....	20
Gambar 5 Screenshot Hasil Jawaban Soal 1	43
Gambar 6 Screenshot Hasil Jawaban Soal 1	44

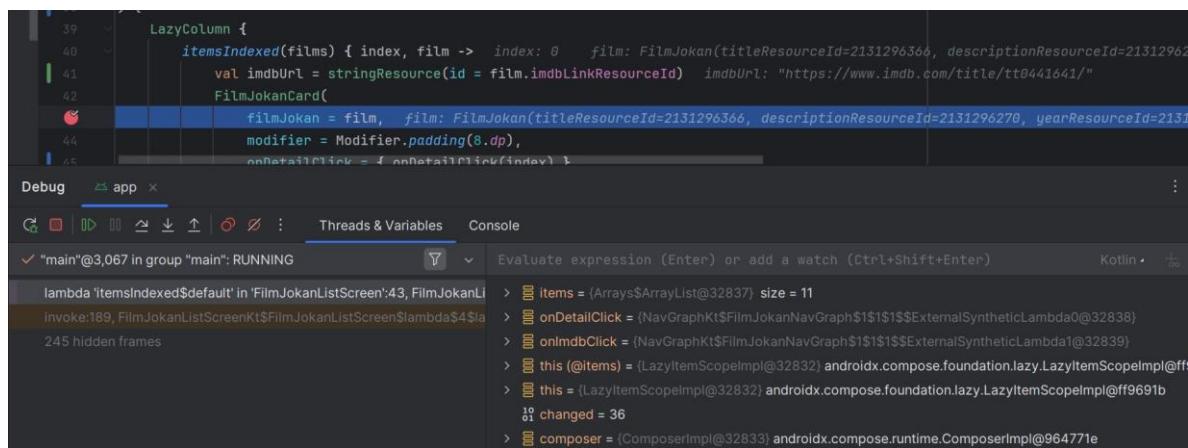
DAFTAR TABEL

Tabel 1 Source Code Jawaban soal 1 XML	7
Tabel 2 Source Code Jawaban soal 1 XML	7
Tabel 3 Source Code Jawaban soal 1 XML	8
Tabel 4 Source Code Jawaban soal 1 XML	9
Tabel 5 Source Code Jawaban soal 1 XML	11
Tabel 6 Source Code Jawaban soal 1 XML	12
Tabel 7 Source Code Jawaban soal 1 XML	13
Tabel 8 Source Code Jawaban soal 1 XML	14
Tabel 9 Source Code Jawaban soal 1 XML	14
Tabel 10 Source Code Jawaban soal 1 XML	15
Tabel 11 Source Code Jawaban soal 1 XML	15
Tabel 12 Source Code Jawaban soal 1 XML	17
Tabel 13 Source Code Jawaban soal 1 XML	17
Tabel 14 Source Code Jawaban soal 1 Compose	34
Tabel 15 Source Code Jawaban soal 1 Compose	35
Tabel 16 Source Code Jawaban soal 1 Compose	35
Tabel 17 Source Code Jawaban soal 1 Compose	37
Tabel 18 Source Code Jawaban soal 1 Compose	40
Tabel 19 Source Code Jawaban soal 1 Compose	41
Tabel 20 Source Code Jawaban soal 1 Compose	42
Tabel 21 Source Code Jawaban soal 1 Compose	42

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 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



The screenshot shows the Android Studio debugger interface. The top part displays a portion of the Kotlin code for a LazyColumn in a FilmJokanListScreen. The bottom part shows the 'Threads & Variables' tab of the debugger, which is currently active. It lists a single thread named 'main' (@3,067) and provides a detailed view of its current state. The 'Variables' section shows several variables with their values and types, such as 'Items' (size = 11), 'onDetailClick' (a synthetic lambda), and 'onImdbClick' (another synthetic lambda). The 'Registers' section shows registers like 'r10 changed = 36' and 'r11 composer = ComposerImpl'. The 'Call Stack' section shows the stack trace with 245 hidden frames, starting from 'lambda \$itemsIndexed\$default' in 'FilmJokanListScreen' at line 43.

Gambar 1 Contoh Penggunaan Debugger

A. Source Code XML

MainActivity.kt

Tabel 1 Source Code Jawaban soal 1 XML

```
1 package com.example.mobilelegendcharacterlistxml
2
3 import HomeFragment
4 import android.os.Bundle
5 import androidx.appcompat.app.AppCompatActivity
6
7
8 class MainActivity : AppCompatActivity() {
9
10    override fun onCreate(savedInstanceState: Bundle?) {
11        super.onCreate(savedInstanceState)
12        setContentView(R.layout.activity_main)
13
14        val fragmentManager = supportFragmentManager
15        val homeFragment = HomeFragment()
16        val fragment =
17        fragmentManager.findFragmentByTag(HomeFragment::class.java.simpleName
18    )
19        if (fragment !is HomeFragment) {
20            fragmentManager
21                .beginTransaction()
22                .add(R.id.frame_container, homeFragment,
23        HomeFragment::class.java.simpleName)
24                .commit()
25        }
26    }
27 }
```

HeroML.kt

Tabel 2 Source Code Jawaban soal 1 XML

```
1 package com.example.mobilelegendcharacterlistxml.model
2
3 import android.os.Parcelable
4 import kotlinx.parcelize.Parcelize
5
6 @Parcelize
7 data class HeroML(
8     val name: String,
9     val image: Int,
10    val url: String,
```

11	val description: String
12): Parcelable

HeroMLAdapter.kt

Tabel 3 Source Code Jawaban soal 1 XML

```

1 package com.example.mobilelegendcharacterlistxml.adapter
2
3 import android.view.LayoutInflater
4 import android.view.ViewGroup
5 import androidx.recyclerview.widget.RecyclerView
6 import
7 com.example.mobilelegendcharacterlistxml.databinding.ItemCharMlBindin
8 g
9 import com.example.mobilelegendcharacterlistxml.model.HeroML
10
11 class HeroMLAdapter(
12     private val onDetailClick: (String) -> Unit,
13     private val onPenjelasanClick: (String, Int, String) -> Unit,
14     private val logClick: (String) -> Unit
15 ) : RecyclerView.Adapter<HeroMLAdapter.ListViewHolder>() {
16
17     private val data = ArrayList<HeroML>()
18
19     inner class ListViewHolder(val binding: ItemCharMlBinding) :
20         RecyclerView.ViewHolder(binding.root) {
21         fun bind(character: HeroML) {
22             binding.tvItemName.text = character.name
23             binding.imgItemMl.setImageResource(character.image)
24             binding.tvIsi.text = character.description
25
26             binding.btnDescription.setOnClickListener {
27                 onDetailClick(character.url)
28             }
29
30             binding.btnPenjelasan.setOnClickListener {
31                 logClick(character.name)
32                 onPenjelasanClick(character.name, character.image,
33 character.description)
34             }
35         }
36     }
37
38     override fun onCreateViewHolder(parent: ViewGroup, viewType:
39     Int): ListViewHolder {
40         val binding =
41             ItemCharMlBinding.inflate(LayoutInflater.from(parent.context),
42             parent, false)
43         return ListViewHolder(binding)

```

```

44 }
45
46     override fun getItemCount(): Int = data.size
47
48     override fun onBindViewHolder(holder: ListViewHolder, position:
49 Int) {
50         holder.bind(data[position])
51     }
52
53     fun submitList(list: List<HeroML>) {
54         data.clear()
55         data.addAll(list)
56         notifyDataSetChanged()
57     }
58 }
```

HomeFragment.kt

Tabel 4 Source Code Jawaban soal 1 XML

```

1 import android.content.Intent
2 import android.net.Uri
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.lifecycle.ViewModelProvider
9 import androidx.recyclerview.widget.LinearLayoutManager
10 import com.example.mobilelegendcharacterlistxml.adapter.HeroMLAdapter
11 import
12 com.example.mobilelegendcharacterlistxml.databinding.FragmentHomeBindi
13 ng
14 import com.example.mobilelegendcharacterlistxml.ui.DetailFragment
15 import
16 com.example.mobilelegendcharacterlistxml.viewmodel.HeroViewModel
17 import
18 com.example.mobilelegendcharacterlistxml.viewmodel.HeroViewModelFactor
19 y
20 import com.example.mobilelegendcharacterlistxml.R
21
22 class HomeFragment : Fragment() {
23
24     private var _binding: FragmentHomeBinding? = null
25     private val binding get() = _binding!!
26
27     private lateinit var viewModel: HeroViewModel
28
29     override fun onCreateView(
30         inflater: LayoutInflater, container: ViewGroup?,
```

```
31     savedInstanceState: Bundle?
32     ): View {
33         _binding = FragmentHomeBinding.inflate(inflater, container,
34         false)
35         return binding.root
36     }
37
38     override fun onViewCreated(view: View, savedInstanceState:
39     Bundle?) {
40         super.onViewCreated(view, savedInstanceState)
41
42         val factory =
43             HeroViewModelFactory(requireActivity().application)
44         viewModel = ViewModelProvider(this,
45             factory)[HeroViewModel::class.java]
46
47         val adapter = HeroMLAdapter(
48             onDetailClick = { url ->
49                 val intent = Intent(Intent.ACTION_VIEW,
50                     Uri.parse(url))
51                 startActivity(intent)
52             },
53             onPenjelasanClick = { name, image, desc ->
54                 val bundle = Bundle().apply {
55                     putString("name", name)
56                     putInt("image", image)
57                     putString("desc", desc)
58                 }
59             }
60
61             val detailFragment = DetailFragment().apply {
62                 arguments = bundle
63             }
64
65             parentFragmentManager.beginTransaction()
66                 .replace(R.id.frame_container, detailFragment)
67                 .addToBackStack(null)
68                 .commit()
69             },
70             logClick = { name ->
71                 viewModel.logHeroClick(name)
72             }
73         )
74
75
76         binding.rvCharacter.layoutManager =
77             LinearLayoutManager(requireContext())
78         binding.rvCharacter.adapter = adapter
79
80         viewModel.heroList.observe(viewLifecycleOwner) { heroList ->
81             adapter.submitList(heroList)
82         }
83     }
```

```

84
85     override fun onDestroyView() {
86         super.onDestroyView()
87         _binding = null
88     }
89 }
```

DetailFragment.kt

Tabel 5 Source Code Jawaban soal 1 XML

```

1 package com.example.mobilelegendcharacterlistxml.ui
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
9 com.example.mobilelegendcharacterlistxml.databinding.FragmentDetailBin
10 ding
11
12 class DetailFragment : Fragment() {
13
14     private var _binding: FragmentDetailBinding? = null
15     private val binding get() = _binding!!
16
17     override fun onCreateView(
18         inflater: LayoutInflater, container: ViewGroup?,
19         savedInstanceState: Bundle?
20     ): View {
21         _binding = FragmentDetailBinding.inflate(inflater, container,
22 false)
23
24         val name = arguments?.getString("name")
25         val photo = arguments?.getInt("image")
26         val description = arguments?.getString("desc")
27
28         binding.tvItemName.text = name
29         photo?.let {
30             binding.imgItemM1.setImageResource(it)
31         }
32         binding.tvIsi.text = description
33
34         return binding.root
35     }
36
37     override fun onDestroyView() {
```

```

38     super.onDestroy()
39     _binding = null
40   }
41 }
```

HeroData.kt

Tabel 6 Source Code Jawaban soal 1 XML

```

1 package com.example.mobilelegendcharacterlistxml.data
2
3 import android.content.Context
4 import com.example.mobilelegendcharacterlistxml.R
5 import com.example.mobilelegendcharacterlistxml.model.HeroML
6
7 object HeroData {
8
9     fun getHeroList(context: Context): List<HeroML> {
10         val names =
11             context.resources.getStringArray(R.array.data_name)
12         val descriptions =
13             context.resources.getStringArray(R.array.data_desc)
14         val links =
15             context.resources.getStringArray(R.array.data_link)
16
17         // Karena drawable tidak bisa diakses lewat string-array
18         langsung, kita hardcode
19         val images = arrayOf(
20             R.drawable.hero1,
21             R.drawable.hero2,
22             R.drawable.hero3,
23             R.drawable.hero4,
24             R.drawable.hero5,
25             R.drawable.hero6
26         )
27
28         val list = ArrayList<HeroML>()
29         for (i in names.indices) {
30             val hero = HeroML(
31                 name = names[i],
32                 image = images[i],
33                 url = links[i],
34                 description = descriptions[i]
35             )
36             list.add(hero)
37         }
38         return list
39 }
```

40	}
----	---

HeroViewModel.kt

Tabel 7 Source Code Jawaban soal 1 XML

```
1 package com.example.mobilelegendcharacterlistxml.viewmodel
2
3 import android.app.Application
4 import androidx.lifecycle.AndroidViewModel
5 import androidx.lifecycle.LiveData
6 import androidx.lifecycle.MutableLiveData
7 import com.example.mobilelegendcharacterlistxml.data.HeroData
8 import com.example.mobilelegendcharacterlistxml.model.HeroML
9
10 class HeroViewModel(application: Application) : 
11     AndroidViewModel(application) {
12
13     private val _heroList = MutableLiveData<List<HeroML>>()
14     val heroList: LiveData<List<HeroML>> = _heroList
15
16     init {
17         loadHeroData()
18     }
19
20     private fun loadHeroData() {
21         _heroList.value = HeroData.getHeroList(getApplicationContext())
22     }
23
24     fun logHeroClick(name: String) {
25         // Timber.d("Klik hero: $name")
26     }
27 }
```

HeroViewModelFactory.kt

Tabel 8 Source Code Jawaban soal 1 XML

```
1 package com.example.mobilelegendcharacterlistxml.viewmodel
2
3 import android.app.Application
4 import androidx.lifecycle.ViewModel
5 import androidx.lifecycle.ViewModelProvider
6
7 class HeroViewModelFactory(
8     private val application: Application
9 ) : ViewModelProvider.NewInstanceFactory() {
10
11     override fun <T : ViewModel> create(modelClass: Class<T>): T {
12         if (modelClass.isAssignableFrom(HeroViewModel::class.java)) {
13             return HeroViewModel(application) as T
14         }
15         throw IllegalArgumentException("Unknown ViewModel class")
16     }
17 }
```

HeroApp.kt

Tabel 9 Source Code Jawaban soal 1 XML

```
1 package com.example.mobilelegendcharacterlistxml
2
3
4 import android.app.Application
5 import timber.log.Timber
6
7
8
9 class HeroApp : Application() {
10     override fun onCreate() {
11         super.onCreate()
12         Timber.plant(Timber.DebugTree())
13     }
14 }
```

Dalam file layout:

activity_main.xml

Dalam XML ada beberapa file tambahan agar sama tampilannya dengan di gambar.

Tabel 10 Source Code Jawaban soal 1 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     android:layout_width="match_parent"
6     android:layout_height="match_parent">
7
8     <!-- Fragment container -->
9     <FrameLayout
10        android:id="@+id/frame_container"
11        android:layout_width="0dp"
12        android:layout_height="0dp"
13        app:layout_constraintTop_toTopOf="parent"
14        app:layout_constraintBottom_toBottomOf="parent"
15        app:layout_constraintStart_toStartOf="parent"
16        app:layout_constraintEnd_toEndOf="parent"/>
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

item_char_ml.xml:

Tabel 11 Source Code Jawaban soal 1 XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.cardview.widget.CardView
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:card_view="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:id="@+id/card_view"
7     android:layout_width="match_parent"
8     android:layout_height="wrap_content"
9     android:layout_gravity="center"
10    android:layout_marginStart="8dp"
11    android:layout_marginTop="4dp"
12    android:layout_marginEnd="8dp"
13    android:layout_marginBottom="4dp"
14    card_view:cardCornerRadius="4dp">
15
16
17     <androidx.constraintlayout.widget.ConstraintLayout
18         android:layout_width="match_parent"
19         android:layout_height="266dp"
20         android:padding="8dp">
21
22         <ImageView
23             android:id="@+id/img_item_ml"
```

```
24         android:layout_width="80dp"
25         android:layout_height="110dp"
26         android:scaleType="centerCrop"
27
28     card_view:layout_constraintBottom_toTopOf="@+id/btn_description"
29             card_view:layout_constraintStart_toStartOf="parent"
30             card_view:layout_constraintTop_toTopOf="parent" />
31
32
33     <TextView
34         android:id="@+id/tv_item_name"
35         android:layout_width="0dp"
36         android:layout_height="wrap_content"
37         android:layout_marginTop="8dp"
38         android:textSize="20sp"
39         android:textStyle="bold"
40         card_view:layout_constraintBottom_toTopOf="@+id/tv_isi"
41         card_view:layout_constraintEnd_toEndOf="parent"
42
43     card_view:layout_constraintStart_toEndOf="@+id/img_item_ml"
44             card_view:layout_constraintTop_toTopOf="parent"
45             card_view:layout_constraintVertical_bias="0.0"
46             tools:text="Nama Hero" />
47
48     <Button
49         android:id="@+id/btn_description"
50         android:layout_width="wrap_content"
51         android:layout_height="wrap_content"
52         android:layout_marginTop="164dp"
53         android:layout_marginStart="20dp"
54         android:text="Detail"
55         card_view:layout_constraintStart_toStartOf="parent"
56
57     card_view:layout_constraintTop_toBottomOf="@+id/tv_item_name" />
58
59     <Button
60         android:id="@+id/btn_penjelasan"
61         android:layout_width="wrap_content"
62         android:layout_height="wrap_content"
63         android:layout_marginStart="16dp"
64         android:layout_marginTop="164dp"
65         android:text="Role"
66
67     card_view:layout_constraintStart_toEndOf="@+id/btn_description"
68
69     card_view:layout_constraintTop_toBottomOf="@+id/tv_item_name" />
70
71     <TextView
72         android:id="@+id/tv_isi"
73         android:layout_width="0dp"
74         android:layout_height="wrap_content"
75         android:layout_marginTop="48dp"
```

```

76         android:layout_marginStart="10dp"
77         android:textSize="16sp"
78         android:textStyle="bold"
79         card_view:layout_constraintEnd_toEndOf="parent"
80         card_view:layout_constraintHorizontal_bias="1.0"
81
82     card_view:layout_constraintStart_toEndOf="@+id/img_item_ml"
83         card_view:layout_constraintTop_toTopOf="parent"
84
85     tools:text="jsndjnjkjnsjnsjidnjksniaunsijnjniskjnaksjnkjnkajsnkjd
86 hisnkjniaanskdnsihaksnidjhaksnidsjaksndisnaksjndisjnaksnjaklnjianskan
87 iusndsj" />
88
89     </androidx.constraintlayout.widget.ConstraintLayout>
90     </androidx.cardview.widget.CardView>

```

fragment_home.xml

Tabel 12 Source Code Jawaban soal 1 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:tools="http://schemas.android.com/tools"
5      xmlns:app="http://schemas.android.com/apk/res-auto"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".HomeFragment">
9
10     <!-- TODO: Update blank fragment layout -->
11     <androidx.recyclerview.widget.RecyclerView
12         android:layout_width="0dp"
13         android:layout_height="0dp"
14         android:id="@+id/rv_character"
15         android:layout_margin="15dp"
16         app:layout_constraintBottom_toBottomOf="parent"
17         app:layout_constraintEnd_toEndOf="parent"
18         app:layout_constraintStart_toStartOf="parent"
19         app:layout_constraintTop_toTopOf="parent" />
20
21 </androidx.constraintlayout.widget.ConstraintLayout>

```

fragment_detail.xml

Tabel 13 Source Code Jawaban soal 1 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:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"

```

```
6     android:layout_height="match_parent"
7     xmlns:app="http://schemas.android.com/apk/res-auto"
8     tools:context=".DetailFragment">
9
10    <ImageView
11        android:id="@+id/img_item_ml"
12        android:layout_width="100dp"
13        android:layout_height="150dp"
14        android:layout_marginTop="84dp"
15        android:scaleType="centerCrop"
16        app:layout_constraintEnd_toEndOf="parent"
17        app:layout_constraintStart_toStartOf="parent"
18        app:layout_constraintTop_toTopOf="parent"
19        tools:src="@tools:sample/avatars" />
20
21    <TextView
22        android:id="@+id/tv_item_name"
23        android:layout_width="wrap_content"
24        android:layout_height="wrap_content"
25        android:layout_marginTop="20dp"
26        app:layout_constraintEnd_toEndOf="parent"
27        app:layout_constraintStart_toStartOf="parent"
28        android:textSize="30dp"
29        app:layout_constraintTop_toBottomOf="@+id/img_item_ml"
30        tools:text="Nama Chara" />
31
32    <TextView
33        android:id="@+id/tv_isi"
34        android:layout_width="0dp"
35        android:layout_height="wrap_content"
36        android:layout_marginTop="32dp"
37        android:text="Deskripsi"
38        android:textSize="16sp"
39        app:layout_constraintEnd_toEndOf="parent"
40        app:layout_constraintHorizontal_bias="1.0"
41        app:layout_constraintStart_toStartOf="parent"
42        app:layout_constraintTop_toBottomOf="@+id/tv_item_name" />
43
44    </androidx.constraintlayout.widget.ConstraintLayout>
```

B. Output Program

The screenshot shows a mobile application interface with three hero profiles displayed vertically. Each profile card includes a hero icon, the hero's name, a brief description of their role and skills, and two buttons at the bottom: 'Detail' and 'Role'. The top card is for 'Franco', the middle for 'Kalea', and the bottom for 'Lylia'. The background of the app has a purple header bar with various icons and the time '01.28'.

Franco

Franco adalah hero dengan role tank yang ada di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu Iron Hook, Skill 2 yaitu Fury Shock, dan Skill 3 yaitu Brutal Massacre, dan memiliki pasif yaitu Annihilation.

Detail

Role

Kalea

Kalea adalah hero dengan role tank/Support yang ada di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu Wavebreaker, Skill 2 yaitu Tidal Strike, dan Skill 3 yaitu Tsunami Slam, dan memiliki pasif yaitu Surge of Life.

Detail

Role

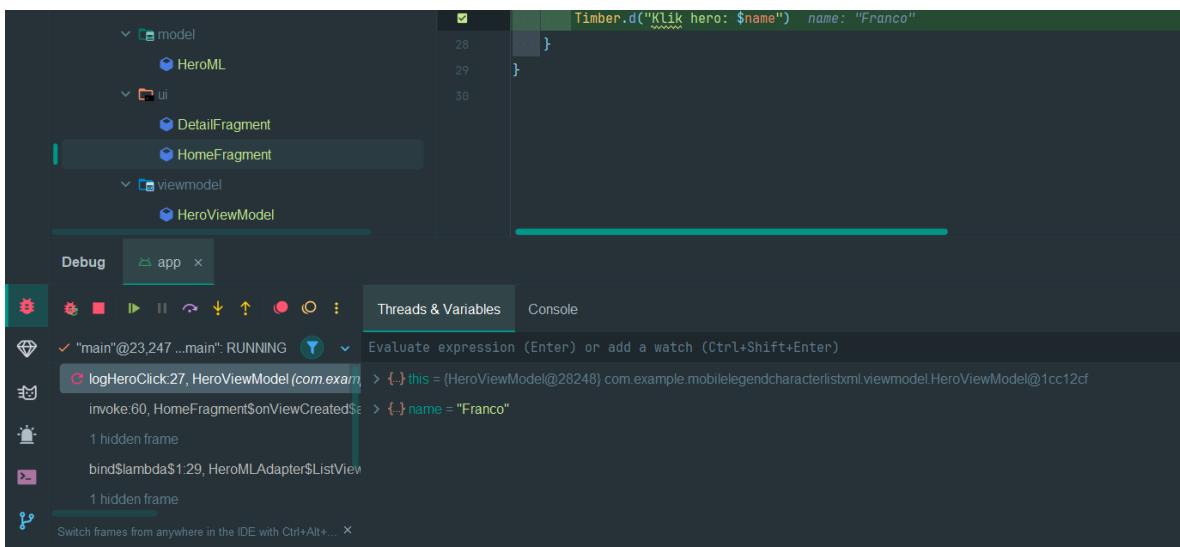
Lylia

Lylia adalah hero dengan role Mage yang ada di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu Magic Shockwave, Skill 2 yaitu Shadow Energy, dan Skill 3 yaitu Black Shoes, dan memiliki pasif yaitu Gloom.

Gambar 2. Screenshot Hasil Jawaban Soal 1 XML



Gambar 3. Screenshot Hasil Jawaban Soal 1 XML



Gambar 4. Timber Button

C. Pembahasan

MainActivity.kt:

Pada baris 1, package com.example.mobilelegendcharacterlistxml mendefinisikan nama package dari file Kotlin ini. Package ini merupakan struktur modular dalam proyek Android yang mengelompokkan file berdasarkan fungsinya.

Pada baris 3, import HomeFragment digunakan untuk mengimpor kelas HomeFragment ke dalam file ini agar bisa digunakan sebagai tampilan awal pada saat aplikasi dijalankan.

Pada baris 4, import android.os.Bundle mengimpor class Bundle yang digunakan untuk menyimpan dan meneruskan data antar komponen, misalnya antar activity atau antar fragment.

Pada baris 5, import androidx.appcompat.app.AppCompatActivity mengimpor class AppCompatActivity, yaitu superclass dari activity yang menyediakan dukungan kompatibilitas untuk fitur-fitur Android modern pada perangkat lama.

Pada baris 8, class MainActivity : AppCompatActivity() mendefinisikan sebuah class bernama MainActivity yang merupakan turunan dari AppCompatActivity, dan bertindak sebagai entry point utama saat aplikasi pertama kali dijalankan.

Pada baris 10, override fun onCreate(savedInstanceState: Bundle?) adalah method yang dipanggil saat activity pertama kali dibuat. Method ini digunakan untuk menginisialisasi UI dan logika pertama kali.

Pada baris 11, super.onCreate(savedInstanceState) memanggil implementasi onCreate milik superclass (AppCompatActivity) untuk menjalankan logika dasar dari lifecycle activity.

Pada baris 12, setContentView(R.layout.activity_main) digunakan untuk menetapkan layout XML activity_main.xml sebagai tampilan utama dari activity ini.

Pada baris 14, val fragmentManager = supportFragmentManager mendeklarasikan sebuah variabel fragmentManager yang digunakan untuk mengelola fragment dalam activity, seperti menambahkan, mengganti, atau menghapus fragment.

Pada baris 15, val homeFragment = HomeFragment() membuat sebuah instance baru dari class HomeFragment yang nantinya akan digunakan untuk ditampilkan sebagai tampilan utama aplikasi.

Pada baris 16, `val fragment = fragmentManager.findFragmentByTag(HomeFragment::class.java.simpleName)` digunakan untuk mencari fragment yang telah ditambahkan sebelumnya berdasarkan tag-nya, untuk menghindari duplikasi fragment saat konfigurasi berubah (seperti rotasi layar).

Pada baris 17, `if (fragment !is HomeFragment)` adalah kondisi untuk memeriksa apakah fragment yang sedang aktif bukan instance dari `HomeFragment`.

Pada baris 18–20, jika kondisi terpenuhi, maka akan dilakukan transaksi fragment menggunakan `fragmentManager.beginTransaction()`, lalu `add()` untuk menambahkan `homeFragment` ke dalam `frame_container`, dan `commit()` untuk menjalankan transaksi tersebut.

HeroML.kt

Pada baris 1, `package com.example.mobilelegendcharacterlistxml.model` menyatakan bahwa file ini berada dalam package `model`, yang berfungsi untuk menyimpan class-class data (data model) dalam struktur proyek aplikasi. Pada baris 3, `import android.os.Parcelable` digunakan untuk mengimpor antarmuka `Parcelable`, yang memungkinkan objek dikirim antar komponen Android seperti `Intent` atau `Fragment`. Ini penting ketika kita ingin mengirim objek secara efisien melalui `Bundle`.

Pada baris 4, `import kotlinx.parcelize.Parcelize` digunakan untuk mengimpor anotasi `@Parcelize` dari Kotlin Android Extensions. Anotasi ini secara otomatis menghasilkan implementasi dari interface `Parcelable`, sehingga kita tidak perlu menuliskan kode `writeToParcel()` dan `describeContents()` secara manual.

Pada baris 6, anotasi `@Parcelize` digunakan di atas deklarasi data class untuk mengaktifkan fitur parcelisasi otomatis. Ini diperlukan agar objek `HeroML` dapat dengan mudah dikirim antar komponen Android, misalnya dari `HomeFragment` ke `DetailFragment`.

Pada baris 7–11, dideklarasikan sebuah data class bernama `HeroML` yang merepresentasikan data karakter Mobile Legends. Class ini memiliki empat properti:

- Pada baris 8, `val name: String` menyimpan nama dari karakter.
- Pada baris 9, `val image: Int` menyimpan ID resource gambar (misalnya `R.drawable.hero1`) dari karakter.
- Pada baris 10, `val url: String` menyimpan URL yang berisi link ke halaman detail karakter di internet.

- Pada baris 11, `val description: String` menyimpan deskripsi singkat mengenai karakter.

Class ini mengimplementasikan interface `Parcelable` melalui tanda titik dua : `Parcelable` pada akhir deklarasi class, sehingga objek `HeroML` dapat dikirim melalui `Intent`, `Bundle`, atau `arguments` dalam `Fragment`.

HeroMLAdapter.kt

Pada baris 1, `package com.example.mobilelegendcharacterlistxml.adapter` menyatakan bahwa file ini berada di dalam package `adapter`, yang biasanya digunakan untuk menyimpan class-class adapter yang berfungsi menghubungkan data dengan tampilan `RecyclerView`. Pada baris 3, `import android.view.LayoutInflater` digunakan untuk mengkonversi layout XML ke dalam objek `View`. Layout ini nantinya akan digunakan sebagai item pada `RecyclerView`. Pada baris 4, `import android.view.ViewGroup` mengimpor class `ViewGroup` yang menjadi induk dari item layout yang akan ditampilkan.

Pada baris 5, `import androidx.recyclerview.widget.RecyclerView` mengimpor class `RecyclerView` dan komponen terkaitnya untuk menampilkan daftar item yang dapat discroll. Pada baris 6, `import com.example.mobilelegendcharacterlistxml.databinding.ItemCharMLBinding` mengimpor class `binding` yang otomatis dibuat dari file layout `item_char_ml.xml`. Binding ini digunakan untuk mengakses elemen-elemen di layout item tanpa perlu menggunakan `findViewById`. Pada baris 7, `import com.example.mobilelegendcharacterlistxml.model.HeroML` mengimpor model data `HeroML` yang akan ditampilkan di `RecyclerView`.

Pada baris 9, dideklarasikan class `HeroMLAdapter` yang merupakan turunan dari `RecyclerView.Adapter`. Class ini menerima tiga parameter lambda:

- Pada baris 10, `onDetailClick: (String) -> Unit` akan dipanggil saat tombol detail ditekan, dan menerima URL sebagai argumen.
- Pada baris 11, `onPenjelasanClick: (String, Int, String) -> Unit` akan dipanggil saat tombol penjelasan ditekan, dan menerima nama, gambar, dan deskripsi sebagai parameter.
- Pada baris 12, `logClick: (String) -> Unit` digunakan untuk mencatat log saat tombol penjelasan ditekan.

Pada baris 14, `data` dideklarasikan sebagai `ArrayList<HeroML>` untuk menyimpan daftar hero yang akan ditampilkan. Data ini akan diisi melalui method `submitList()`.

Pada baris 16–29, class `ListViewHolder` dideklarasikan sebagai inner class dari adapter. Class ini menerima parameter `binding` bertipe `ItemCharMLBinding` dan

bertanggung jawab untuk mengikat data ke tampilan setiap item RecyclerView. Pada baris 17, konstruktor dipanggil dengan binding.root sebagai root View.

Pada baris 18–29, method bind() digunakan untuk mengatur data ke dalam View pada satu item:

- Pada baris 19, binding.tvItemName.text = character.name menampilkan nama karakter.
- Pada baris 20, binding.imgItem1.setImageResource(character.image) menampilkan gambar karakter berdasarkan resource ID.
- Pada baris 21, binding.tvIsi.text = character.description menampilkan deskripsi singkat karakter.

Pada baris 23, listener untuk tombol btnDescription diatur. Saat tombol ditekan, lambda onDetailClick dipanggil dengan parameter URL dari karakter. Pada baris 26, listener untuk tombol btnPenjelasan diatur. Pertama, lambda logClick dipanggil untuk mencatat interaksi pengguna berdasarkan nama karakter. Lalu, lambda onPenjelasanClick dipanggil dengan parameter nama, gambar, dan deskripsi karakter.

Pada baris 31–34, method onCreateViewHolder() digunakan untuk membuat instance dari ViewHolder. Pada baris 32, layout ItemCharMLBinding di-inflate dari XML menggunakan LayoutInflater. Pada baris 33, objek ViewHolder baru dikembalikan.

Pada baris 36, method getItemCount() mengembalikan jumlah item dalam daftar data. Fungsi ini digunakan oleh RecyclerView untuk menentukan berapa banyak item yang akan ditampilkan.

Pada baris 38, method onBindViewHolder() bertanggung jawab untuk memanggil fungsi bind() pada ViewHolder dan memberikan data yang sesuai berdasarkan posisi.

Pada baris 41–44, method submitList() digunakan untuk meng-update data di dalam adapter. Pada baris 42, data lama dihapus menggunakan data.clear(). Pada baris 43, data baru ditambahkan menggunakan data.addAll(list). Pada baris 44, notifyDataSetChanged() dipanggil untuk memberitahu RecyclerView agar memperbarui seluruh tampilan data.

HomeFragment.kt

Pada baris 1, import android.content.Intent digunakan untuk mengimpor class Intent, yang berfungsi untuk melakukan navigasi atau membuka aplikasi lain seperti browser. Pada baris 2, import android.net.Uri digunakan untuk mengimpor class

`Uri`, yang dibutuhkan saat membuka link atau URL dari data karakter. Pada baris 3, `import android.os.Bundle` mengimpor class `Bundle`, yang digunakan untuk mengirim data antar Fragment. Pada baris 4, `import android.view.LayoutInflater` digunakan untuk mengonversi layout XML menjadi objek `View`. Pada baris 5, `import android.view.View` mengimpor class `View`, sebagai elemen dasar dari tampilan Android. Pada baris 6, `import android.view.ViewGroup` digunakan untuk menyatakan parent layout dari fragment yang di-inflate.

Pada baris 7, `import androidx.fragment.app.Fragment` mengimpor class `Fragment` dari `AndroidX`, yang digunakan sebagai dasar pembuatan `HomeFragment`. Pada baris 8, `import androidx.lifecycle.ViewModelProvider` digunakan untuk mendapatkan instance dari `ViewModel`, yaitu `HeroViewModel`, dengan `lifecycle-aware`. Pada baris 9, `import androidx.recyclerview.widget.LinearLayoutManager` digunakan untuk menampilkan daftar data secara vertikal di dalam `RecyclerView`. Pada baris 10, `import com.example.mobilelegendcharacterlistxml.adapter.HeroMLAdapter` mengimpor class adapter yang bertanggung jawab menghubungkan data `HeroML` dengan tampilan `RecyclerView`. Pada baris 11, `import com.example.mobilelegendcharacterlistxml.databinding.FragmentHomeBinding` mengimpor class binding otomatis dari layout `fragment_home.xml` untuk memudahkan akses ke elemen UI tanpa `findViewById`.

Pada baris 12, `import com.example.mobilelegendcharacterlistxml.ui.DetailFragment` mengimpor class `Fragment` yang digunakan untuk menampilkan detail karakter Mobile Legends. Pada baris 13, `import com.example.mobilelegendcharacterlistxml.viewmodel.HeroViewModel` mengimpor `ViewModel` yang menyimpan dan mengelola data list karakter. Pada baris 14, `import com.example.mobilelegendcharacterlistxml.viewmodel.HeroViewModelFactory` mengimpor factory yang digunakan untuk membuat instance `HeroViewModel` dengan parameter `Application`. Pada baris 15, `import com.example.mobilelegendcharacterlistxml.R` mengimpor class `R` yang mewakili semua resource dalam proyek seperti layout, drawable, dan ID view.

Pada baris 17, dideklarasikan class `HomeFragment` yang merupakan turunan dari `Fragment`. Pada baris 19, properti `_binding` bertipe nullable `FragmentHomeBinding` dideklarasikan untuk menghubungkan layout XML dengan kode Kotlin. Pada baris 20, `binding` didefinisikan sebagai non-nullable getter untuk menghindari kesalahan null pointer saat mengakses elemen UI.

Pada baris 22–26, method `onCreateView()` diimplementasikan untuk meng-inflate layout `fragment_home.xml` dan mengembalikan root view yang dihasilkan dari binding. Proses ini akan membuat tampilan Fragment berdasarkan layout XML.

Pada baris 28–56, method `onViewCreated()` digunakan untuk menginisialisasi data dan tampilan setelah Fragment dan view-nya dibuat. Pada baris 30, `HeroViewModelFactory` diinisialisasi dengan `requireActivity().application` sebagai parameter, lalu digunakan untuk mendapatkan instance dari `HeroViewModel` dengan `ViewModelProvider`.

Pada baris 32–47, objek `HeroMLAdapter` diinisialisasi dengan tiga parameter lambda:

- Pada baris 33–35, lambda `onDetailClick` membuat Intent baru dengan `ACTION_VIEW` dan URI dari URL karakter, lalu membuka browser menggunakan `startActivity`.
- Pada baris 36–44, lambda `onPenjelasanClick` digunakan untuk memindahkan data karakter ke `DetailFragment`. Data disimpan dalam `Bundle`, lalu dimasukkan ke `arguments`. Fragment baru diatur dengan transaksi `replace`, kemudian ditambahkan ke back stack agar bisa kembali.
- Pada baris 45, lambda `logClick` memanggil fungsi `logHeroClick(name)` dari `ViewModel` untuk mencatat nama karakter yang ditekan melalui log.

Pada baris 50–51, `RecyclerView` diatur menggunakan `LinearLayoutManager` untuk menampilkan daftar karakter secara vertikal. Adapter yang telah dibuat juga dipasangkan ke `RecyclerView` melalui `binding.rvCharacter.adapter`.

Pada baris 53, `viewModel.heroList` diobservasi menggunakan `observe()` dengan lifecycle milik Fragment. Saat data dalam `LiveData` berubah, fungsi lambda akan dipanggil dan memanggil `adapter.submitList(heroList)` untuk memperbarui isi `RecyclerView`.

Pada baris 55–57, method `onDestroyView()` dipanggil saat view Fragment dihancurkan. Di sini, `_binding` diset `null` untuk menghindari memory leak karena referensi UI tidak lagi diperlukan.

DetailFragment.kt

Pada baris 1, `package com.example.mobilelegendcharacterlistxml.ui` menyatakan bahwa file ini berada di dalam package `ui`, yang biasanya digunakan untuk menyimpan class-class yang berhubungan dengan tampilan (user interface) dalam aplikasi. Pada baris 3, `import android.os.Bundle` digunakan untuk mengimpor class `Bundle` yang berfungsi untuk menerima dan mengirim data antar Fragment atau Activity. Pada baris 4, `import android.view.LayoutInflater` digunakan untuk mengkonversi file XML layout menjadi objek `View`. Pada baris 5, `import`

`android.view.View` mengimpor class `View` sebagai dasar dari semua komponen UI Android. Pada baris 6, `import android.view.ViewGroup` digunakan sebagai parent container dari layout yang akan di-inflate.

Pada baris 7, `import androidx.fragment.app.Fragment` mengimpor class `Fragment` dari AndroidX yang menjadi dasar dari class `DetailFragment`. Fragment ini digunakan untuk menampilkan halaman detail dari karakter Mobile Legends. Pada baris 8, `import com.example.mobilelegendcharacterlistxml.databinding.FragmentDetailBinding` mengimpor class `binding` yang secara otomatis dibuat dari layout `fragment_detail.xml`, yang digunakan untuk mengakses elemen UI tanpa harus menggunakan `findViewById`.

Pada baris 10, dideklarasikan class `DetailFragment` yang merupakan turunan dari `Fragment`. Class ini berfungsi untuk menampilkan tampilan detail dari salah satu karakter yang dipilih pada halaman utama.

Pada baris 12, variabel `_binding` bertipe nullable `FragmentDetailBinding` dideklarasikan untuk menyimpan instance binding dari layout. Pada baris 13, variabel `binding` dideklarasikan sebagai non-nullable dan digunakan untuk mengakses elemen layout secara aman selama fragment masih aktif.

Pada baris 15–24, method `onCreateView()` diimplementasikan untuk meng-inflate layout `fragment_detail.xml`, menampilkan data, dan mengembalikan root view-nya. Pada baris 16, `binding` diinisialisasi dengan `FragmentDetailBinding.inflate()` untuk mengakses elemen UI.

Pada baris 18–20, data dikirim dari Fragment sebelumnya diambil menggunakan `arguments?.getString()` dan `arguments?.getInt()`. Tiga data yang diambil adalah `name`, `image`, dan `desc`, yang mewakili nama karakter, resource ID gambar, dan deskripsi karakter.

Pada baris 22, `binding.tvItemName.text = name` digunakan untuk menampilkan nama karakter ke dalam `TextView`. Pada baris 23–25, jika `photo` tidak null, maka gambar karakter ditampilkan melalui `setImageResource()`. Pada baris 26, deskripsi karakter ditampilkan ke dalam `TextView` `tvIsi`.

Pada baris 28, `return binding.root` mengembalikan root view dari layout untuk ditampilkan sebagai isi dari Fragment.

Pada baris 30–32, method `onDestroyView()` dipanggil ketika view Fragment dihancurkan. Di dalam method ini, `_binding` diset null untuk menghindari memory leak, karena binding tidak lagi diperlukan setelah tampilan dihancurkan.

HeroData.kt:

Pada baris 1, package com.example.mobilelegendcharacterlistxml.data menyatakan bahwa file ini berada dalam package data, yang biasanya digunakan untuk menyimpan class atau object yang berkaitan dengan pengolahan atau penyediaan data dalam aplikasi. Pada baris 3, import android.content.Context mengimpor class Context dari Android, yang diperlukan untuk mengakses resource aplikasi seperti string-array. Pada baris 4, import com.example.mobilelegendcharacterlistxml.R mengimpor class R yang merepresentasikan semua resource dalam proyek, termasuk drawable dan string-array. Pada baris 5, import com.example.mobilelegendcharacterlistxml.model.HeroML mengimpor data class HeroML yang akan digunakan untuk membuat daftar objek karakter.

Pada baris 7, dideklarasikan object HeroData yang merupakan singleton object di Kotlin. Object ini berfungsi sebagai penyedia data (data provider) untuk daftar karakter Mobile Legends yang akan ditampilkan pada aplikasi.

Pada baris 9, fungsi getHeroList(context: Context): List<HeroML> dideklarasikan dengan parameter context bertipe Context, dan akan mengembalikan list dari objek HeroML. Fungsi ini digunakan untuk mengambil data karakter dari resource dan menyusunnya menjadi objek HeroML.

Pada baris 10, val names = context.resources.getStringArray(R.array.data_name) digunakan untuk mengambil array string data_name dari resource strings.xml, yang berisi daftar nama karakter. Pada baris 11, val descriptions = context.resources.getStringArray(R.array.data_desc) mengambil array string yang berisi deskripsi dari setiap karakter. Pada baris 12, val links = context.resources.getStringArray(R.array.data_link) mengambil array string yang berisi URL link karakter.

Pada baris 14–21, karena ID drawable tidak dapat disimpan di dalam strings.xml atau array.xml, maka pada baris 15, array images didefinisikan secara manual (hardcode) menggunakan ID resource drawable. Setiap elemen dalam array ini mewakili gambar dari masing-masing karakter.

Pada baris 23, val list = ArrayList<HeroML>() mendeklarasikan list kosong bertipe HeroML yang nantinya akan diisi dengan data dari semua karakter. Pada baris 24, dilakukan perulangan for (i in names.indices) yang berarti perulangan sebanyak jumlah elemen di dalam array names.

Pada baris 25–28, setiap iterasi membuat objek HeroML menggunakan data dari array names, images, links, dan descriptions berdasarkan indeks yang sama. Pada baris 29, objek hero ditambahkan ke dalam list menggunakan list.add(hero).

Pada baris 30, fungsi `getHeroList` mengembalikan seluruh list yang telah berisi objek `HeroML`.

HeroViewModel.kt:

Pada baris 1, `package com.example.mobilelegendcharacterlistxml.viewmodel` menyatakan bahwa file ini berada di dalam package `viewmodel`, yang berfungsi untuk menyimpan kelas-kelas `ViewModel` dalam arsitektur MVVM (Model-View-ViewModel). Pada baris 3, `import android.app.Application` mengimpor class `Application` dari Android, yang digunakan untuk mendapatkan konteks aplikasi dalam `ViewModel`. Pada baris 4, `import androidx.lifecycle.AndroidViewModel` mengimpor class `AndroidViewModel`, yaitu subclass dari `ViewModel` yang memiliki akses langsung ke `Application`. Ini diperlukan untuk mengambil resource seperti `Context`.

Pada baris 5, `import androidx.lifecycle.LiveData` digunakan untuk mengimpor class `LiveData`, yaitu komponen lifecycle-aware yang dapat diamati oleh UI. Pada baris 6, `import androidx.lifecycle.MutableLiveData` mengimpor versi mutable dari `LiveData`, yang digunakan untuk menyimpan dan memperbarui data. Pada baris 7, `import com.example.mobilelegendcharacterlistxml.data.HeroData` mengimpor class `HeroData`, yang berisi fungsi penyedia data karakter. Pada baris 8, `import com.example.mobilelegendcharacterlistxml.model.HeroML` mengimpor data class `HeroML`, yang merepresentasikan data karakter Mobile Legends. Pada baris 9, `import timber.log.Timber` mengimpor library Timber yang digunakan untuk melakukan logging secara efisien.

Pada baris 11, dideklarasikan class `HeroViewModel` yang merupakan subclass dari `AndroidViewModel`. Class ini digunakan untuk mengelola dan menyimpan data karakter, serta bertahan saat rotasi layar. Konstruktor menerima parameter `application: Application`, yang diteruskan ke superclass `AndroidViewModel(application)`.

Pada baris 13, variabel `_heroList` bertipe `MutableLiveData<List<HeroML>>` digunakan untuk menyimpan list data karakter dan memungkinkan perubahan data. Pada baris 14, variabel `heroList` bertipe `LiveData<List<HeroML>>` hanya memberikan akses baca ke UI untuk mencegah perubahan langsung dari luar `ViewModel`.

Pada baris 16, blok `init` dipanggil secara otomatis saat `HeroViewModel` pertama kali dibuat. Pada baris 17, method `loadHeroData()` dipanggil untuk memuat data karakter dari `HeroData`. Pada baris 18, `Timber.d("HeroViewModel initialized")` digunakan untuk mencatat bahwa `ViewModel` telah diinisialisasi, sebagai bagian dari proses debugging menggunakan Timber.

Pada baris 20–22, fungsi `loadHeroData()` digunakan untuk mengambil data karakter dari `HeroData` menggunakan `getHeroList(getApplicationContext())`, karena `getHeroList` membutuhkan Context. Nilai yang didapat kemudian disimpan ke `_heroList.value`. Pada baris 22, `Timber.d("Data loaded: ${_heroList.value}")` digunakan untuk mencatat data yang berhasil dimuat ke log.

Pada baris 24–26, fungsi `logHeroClick(name: String)` digunakan untuk mencatat nama karakter yang diklik ke dalam log menggunakan `Timber.d()`. Fungsi ini akan dipanggil dari Fragment saat tombol ditekan, sebagai bagian dari pencatatan aktivitas pengguna.

HeroViewModelFactory.kt:

Pada baris 1, `package com.example.mobilelegendcharacterlistxml.viewmodel` menyatakan bahwa file ini berada di dalam package `viewmodel`, yang digunakan untuk menyimpan class-class `ViewModel` dan komponennya dalam arsitektur MVVM. Pada baris 3, `import android.app.Application` mengimpor class `Application` dari Android yang digunakan untuk mendapatkan konteks global aplikasi. Pada baris 4, `import androidx.lifecycle.ViewModel` mengimpor class `ViewModel` sebagai superclass dasar untuk semua `ViewModel` yang digunakan dalam arsitektur lifecycle-aware. Pada baris 5, `import androidx.lifecycle.ViewModelProvider` mengimpor `ViewModelProvider`, yang digunakan untuk mengelola pembuatan dan penyimpanan instance `ViewModel`.

Pada baris 7, dideklarasikan class `HeroViewModelFactory` yang merupakan turunan dari `ViewModelProvider.NewInstanceFactory`. Class ini bertugas sebagai factory atau pembuat instance dari `HeroViewModel`, khususnya saat `HeroViewModel` membutuhkan parameter tambahan seperti `Application`.

Pada baris 8, variabel `application` bertipe `Application` dideklarasikan sebagai properti privat dari class, dan nilainya akan diberikan melalui konstruktor. Properti ini nantinya akan digunakan untuk menginisialisasi `HeroViewModel`.

Pada baris 10–14, method override `fun <T : ViewModel> create(modelClass: Class<T>): T` digunakan untuk membuat dan mengembalikan instance dari `ViewModel`. Pada baris 11, dilakukan pengecekan apakah `modelClass` merupakan turunan dari `HeroViewModel` menggunakan `isAssignableFrom()`. Jika iya, maka pada baris 12, akan dikembalikan objek `HeroViewModel(application)` dan dikonversi menjadi tipe `T`. Jika `modelClass` bukan `HeroViewModel`, maka pada baris 14 akan dilemparkan `IllegalArgumentException` dengan pesan "Unknown ViewModel class" untuk menunjukkan bahwa kelas yang diminta tidak dikenali oleh factory ini.

Activity_main.xml

Pada line 1–7, layout menggunakan `FrameLayout` dengan atribut `layout_width` dan `layout_height` diset ke `match_parent`, sehingga memenuhi seluruh layar

perangkat. Elemen ini memiliki `id` yaitu `@+id/frame_container`, yang berfungsi sebagai referensi dalam `MainActivity.kt` untuk menambahkan fragment menggunakan metode `fragmentManager.beginTransaction().add(...)`.

Atribut `tools:context=".MainActivity"` hanya digunakan saat design preview di Android Studio untuk memberi tahu bahwa layout ini digunakan oleh kelas `MainActivity`. Di dalam `FrameLayout` ini tidak ada elemen UI lain secara langsung karena kontennya akan diganti secara dinamis oleh fragment yang dimasukkan ke dalam `frame_container` tersebut saat runtime.

Item_char_ml.xml

File `item_char_ml.xml` merupakan layout yang digunakan sebagai tampilan item tunggal dalam `RecyclerView` di aplikasi ini. Layout ini digunakan di dalam `HeroMLAdapter.kt`, tepatnya di `ViewHolder` bernama `ListViewHolder`, untuk menampilkan setiap karakter Mobile Legends satu per satu dalam daftar.

Pada line 1–15, layout dibungkus oleh komponen `CardView` dari `androidx.cardview.widget.CardView`, yang memberikan efek bayangan dan sudut membulat pada item. Atribut `cardCornerRadius` diset ke `4dp` agar sudutnya agak melengkung, dan terdapat margin di setiap sisi item agar tidak terlalu mepet satu sama lain ketika ditampilkan dalam daftar. `CardView` ini akan menjadi elemen visual utama dari setiap item hero.

Pada line 17–94, di dalam `CardView` terdapat `ConstraintLayout` yang digunakan untuk menyusun elemen-elemen UI secara fleksibel dengan constraint antar komponen. `ConstraintLayout` memiliki tinggi tetap `266dp` dan padding `8dp` agar isi tidak terlalu rapat ke tepi.

Di dalamnya, pertama ada `ImageView` dengan id `img_item_ml` (baris 19–26) yang digunakan untuk menampilkan gambar hero. Gambar ini berukuran `80dp x 110dp` dan disetel dengan `scaleType="centerCrop"` agar gambar mengisi seluruh area dengan proporsional. Letaknya dikaitkan (`constraint`) di bagian atas layout dan sejajar secara vertikal dengan elemen lainnya.

Lalu ada `TextView` dengan id `tv_item_name` (baris 28–40) yang menampilkan nama hero. Lebarnya dibuat `0dp` karena menggunakan constraint start dan end, dengan ukuran teks `20sp` dan gaya `bold`. Letaknya berada di atas elemen `tv_isi`, dan disesuaikan agar bersebelahan dengan `ImageView` sebelumnya.

Baris 42–54 Berikutnya terdapat dua buah `Button`, yaitu `btn_description` dan `btn_penjelasan`. Kedua tombol ini berada di bawah teks nama dan digunakan sebagai aksi untuk menampilkan link deskripsi hero (tombol `Detail`) dan penjelasan detail dalam

fragment baru (tombol Role). Keduanya disejajarkan secara horizontal dengan sedikit jarak di antara keduanya.

Baris 56–68 Terakhir, ada `TextView` dengan id `tv_isi` yang berfungsi menampilkan deskripsi singkat dari hero tersebut. Komponen ini juga berada di samping `ImageView`, sejajar secara horizontal dan berada di atas tombol-tombol. Deskripsinya diset bold dengan ukuran 16sp agar mudah terbaca. Properti `tools:text` digunakan sebagai contoh isi deskripsi saat preview di Android Studio.

fragment_detail.xml

Pada line 1, dideklarasikan bahwa file ini merupakan file XML dengan encoding UTF-8.

Pada line 2–5, digunakan `ConstraintLayout` sebagai root layout. Layout ini memungkinkan setiap elemen UI diposisikan relatif terhadap elemen lain dan/atau parentnya. Layout memiliki lebar dan tinggi `match_parent`, sehingga memenuhi seluruh layar. Tiga namespace juga dideklarasikan dalam elemen root: `xmlns:android` digunakan untuk atribut standar Android seperti `layout_width`, `id`, dan sebagainya; `xmlns:tools` digunakan untuk kebutuhan preview di Android Studio, seperti memberikan data dummy melalui `tools:text`; dan `xmlns:app` digunakan untuk atribut-atribut khusus dari `ConstraintLayout`, seperti `app:layout_constraintTop_toBottomOf`.

Pada line 7–14, didefinisikan sebuah `ImageView` dengan ID `img_item_ml`. Komponen ini digunakan untuk menampilkan gambar karakter Mobile Legends. Ukurannya diset sebesar 100dp x 150dp dan `scaleType` diatur `centerCrop`, yang membuat gambar memenuhi seluruh area tampilan tanpa mengubah aspek rasio. Gambar ini diposisikan di tengah horizontal dengan margin atas sebesar 84dp dari parent layout.

Pada line 16–23, terdapat sebuah `TextView` dengan ID `tv_item_name` untuk menampilkan nama karakter. Ukuran teks dibuat cukup besar yaitu 30dp agar nama karakter lebih menonjol di layar. Elemen ini diletakkan tepat di bawah `ImageView` dengan margin atas 20dp, dan disejajarkan secara horizontal di tengah parent layout.

Pada line 25–32, dideklarasikan `TextView` kedua dengan ID `tv_isi`, yang digunakan untuk menampilkan deskripsi karakter Mobile Legends secara lebih lengkap. Lebarnya menggunakan 0dp untuk mengikuti aturan `ConstraintLayout` (akan dihitung berdasarkan batas kiri dan kanan), dengan tinggi menyesuaikan isi. Posisi elemen ini berada di bawah nama karakter dengan margin atas 32dp. Ukuran teks disesuaikan agar tetap nyaman dibaca, yaitu 16sp, dan penempatan horizontalnya tetap berada di tengah lebar layout.

fragment_char.xml

Pada line 1, dideklarasikan bahwa file ini merupakan file XML dengan encoding UTF-8. Ini adalah deklarasi standar untuk file XML agar sistem dapat memahami format karakter yang digunakan.

Pada line 2–7, digunakan ConstraintLayout sebagai root layout. ConstraintLayout merupakan jenis layout fleksibel yang memungkinkan setiap elemen UI diposisikan secara relatif terhadap elemen lain maupun terhadap parent-nya. Layout ini memiliki lebar dan tinggi match_parent, yang berarti akan memenuhi seluruh ukuran layar. Tiga namespace juga didefinisikan di sini:

Tiga namespace didefinisikan dalam elemen root ConstraintLayout. `xmlns:android` digunakan untuk atribut umum Android seperti `layout_width`, `id`, dan lainnya. `xmlns:tools` digunakan oleh Android Studio untuk keperluan preview layout, seperti menampilkan data dummy saat proses desain. Sementara itu, `xmlns:app` dipakai untuk atribut khusus yang berasal dari library AndroidX, termasuk atribut-atribut dari ConstraintLayout seperti `app:layout_constraintTop_toTopOf`.

Pada line 9–17, dideklarasikan sebuah komponen RecyclerView dengan ID `rv_character`. Komponen ini digunakan untuk menampilkan daftar karakter Mobile Legends dalam bentuk list atau grid yang bisa scroll. Lebar dan tinggi diatur 0dp, yang berarti mengikuti aturan constraint dari ConstraintLayout. RecyclerView ini diberi margin sebesar 15dp di keempat sisi agar tidak menempel langsung ke tepi layar. Untuk posisi atributnya `app:layout_constraintTop_toTopOf="parent"` dan `app:layout_constraintBottom_toBottomOf="parent"` digunakan untuk menempatkan komponen dari bagian atas hingga bawah, sehingga memenuhi tinggi parent-nya secara vertikal. Sedangkan `app:layout_constraintStart_toStartOf="parent"` dan `app:layout_constraintEnd_toEndOf="parent"` membuat RecyclerView memanjang secara horizontal dari kiri ke kanan, mengikuti lebar parent. Dengan keempat constraint ini, RecyclerView akan ditampilkan memenuhi seluruh layar, dengan margin di sekelilingnya yang telah ditentukan melalui atribut `android:layout_margin`.

A. Source Code Compose

MainActivity.kt

Tabel 14 Source Code Jawaban soal 1 Compose

```
1 package com.example.mobilelegendcharacterlist
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.compose.foundation.layout.fillMaxSize
8 import androidx.lifecycle.viewmodel.compose.viewModel
9 import androidx.compose.material3.*
10 import androidx.compose.ui.Modifier
11 import
12 com.example.mobilelegendcharacterlist.ui.theme.MobileLegendCharacterLi
13 stTheme
14 import com.example.mobilelegendcharacterlist.viewmodel.HeroViewModel
15 import
16 com.example.mobilelegendcharacterlist.viewmodel.HeroViewModelFactory
17 import androidx.navigation.compose.rememberNavController
18 import androidx.navigation.compose.NavHost
19 import androidx.navigation.compose.composable
20 import androidx.navigation.NavType
21 import androidx.navigation.navArgument
22 import com.example.mobilelegendcharacterlist.ui.screen.DetailScreen
23 import com.example.mobilelegendcharacterlist.ui.screen.HomeScreen
24
25 class MainActivity : ComponentActivity() {
26     override fun onCreate(savedInstanceState: Bundle?) {
27         super.onCreate(savedInstanceState)
28         enableEdgeToEdge()
29
30         setContent {
31             MobileLegendCharacterListTheme {
32                 val navController = rememberNavController()
33
34                 //  Inisialisasi ViewModel pakai Factory
35                 val viewModel: HeroViewModel = viewModel(
36                     factory = HeroViewModelFactory("List Hero ML")
37                 )
38
39                 Surface(
40                     modifier = Modifier.fillMaxSize(),
41                     color = MaterialTheme.colorScheme.background
42                 ) {
43                     NavHost(
44                         navController = navController,
45                         startDestination = "heroList"
46                 ) {
```

```

47         composable("heroList") {
48             HomeScreen(viewModel = viewModel,
49             navController = navController)
50         }
51         composable(
52             route =
53             "penjelasan/{description}/{image}",
54             arguments = listOf(
55                 navArgument("description") { type =
56                 NavType.StringType },
57                 navArgument("image") { type =
58                 NavType.IntType }
59             )
60         ) { backStackEntry ->
61             val description =
62             backStackEntry.arguments?.getString("description") ?: ""
63             val image =
64             backStackEntry.arguments?.getInt("image") ?: 0
65             DetailScreen(description = description,
66             image = image)
67         }
68     }
69 }
70 }
71 }
72 }
73 }

```

DataHero.kt

Tabel 15 Source Code Jawaban soal 1 Compose

```

1 package com.example.mobilelegendcharacterlist.model
2
3 data class DataHero(
4     val name: String,
5     val image: Int,
6     val url: String,
7     val description : String
8 )

```

HeroData.kt

Tabel 16 Source Code Jawaban soal 1 Compose

```

1 package com.example.mobilelegendcharacterlist.data
2
3 import com.example.mobilelegendcharacterlist.R
4 import com.example.mobilelegendcharacterlist.model.DataHero

```

```

5
6     object HeroData{
7         val heroList = listOf(
8             DataHero(name ="Franco",
9                 R.drawable.hero1,
10                url =
11                "https://www.mobilelegends.com/hero/detail?channelid=2678746&heroid=10
12                ",
13                description = "Role: Tank\n" +
14                    "Franco adalah hero dengan role tank yang ada di
15 Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Iron Hook',
16 Skill 2 yaitu 'Fury Shock', dan Skill 3 yaitu 'Brutal Massacre', dan
17 memiliki pasif yaitu 'Annihilation'."),
18             DataHero(name ="Kalea",
19                 R.drawable.hero2,
20                url =
21                "https://www.mobilelegends.com/hero/detail?channelid=2863075&heroid=12
22                ",
23                description = "Role: Tank/Support\n" +
24                    "Kalea adalah hero dengan role tank/Support yang
25 ada di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu
26 'Wavebreaker', Skill 2 yaitu 'Tidal Strike', dan Skill 3 yaitu
27 'Tsunami Slam', dan memiliki pasif yaitu 'Surge of Life'.),
28             DataHero(name ="Lylia",
29                 R.drawable.hero3,
30                url =
31                "https://www.mobilelegends.com/hero/detail?channelid=2678822&heroid=86
32                ",
33                description = "Role: Mage\n" +
34                    "Lylia adalah hero dengan role Mage yang ada di
35 Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Magic
36 Shockwave', Skill 2 yaitu 'Shadow Energy', dan Skill 3 yaitu 'Black
37 Shoes', dan memiliki pasif yaitu 'Angry Gloom'.),
38             DataHero(name ="Hanzo",
39                 R.drawable.hero4,
40                url =
41                "https://www.mobilelegends.com/hero/detail?channelid=2678805&heroid=69
42                ",
43                description = "Role: Assasin\n" +
44                    "Hanzo adalah hero dengan role Assasin yang ada di
45 Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Ninjutsu: Demon
46 Feast', Skill 2 yaitu 'Ninjutsu: Dark Mist', dan Skill 3 yaitu
47 'Kinjutsu: Pinnacle Ninja', dan memiliki pasif yaitu 'Ame no
48 Habakiri'.),
49             DataHero(name ="Lancelot",
50                 R.drawable.hero5,
51                url =
52                "https://www.mobilelegends.com/hero/detail?channelid=2678783&heroid=47
53                ",
54                description = "Role: Assasin\n" +
55                    "Lancelot adalah hero dengan role Assasin yang ada
di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Puncture',
```

```

56 Skill 2 yaitu 'Thorned Rose', dan Skill 3 yaitu 'Phantom Execution',
57 dan memiliki pasif yaitu 'Soul Cutter'."),
58     DataHero(name ="Lukas",
59             R.drawable.hero6,
60             url =
61 "https://www.mobilelegends.com/hero/detail?channelid=2819992&heroid=12
62 7",
63         description = "Role: Fighter\n" +
64                 "Lukas adalah hero dengan role tank yang ada di
65 Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Flash Combo',
66 Skill 2 yaitu 'Flash Step', dan Skill 3 yaitu 'Unleash the Beast', dan
67 memiliki pasif yaitu 'Hero's Resolve'.")
68
69     )
70 }

```

HomeScreen.kt

Tabel 17 Source Code Jawaban soal 1 Compose

```

1 package com.example.mobilelegendcharacterlist.ui.screen
2
3 import android.content.Intent
4 import android.net.Uri
5 import androidx.compose.foundation.Image
6 import androidx.compose.foundation.layout.Arrangement
7 import androidx.compose.foundation.layout.Column
8 import androidx.compose.foundation.layout.PaddingValues
9 import androidx.compose.foundation.layout.Row
10 import androidx.compose.foundation.layout.Spacer
11 import androidx.compose.foundation.layout.fillMaxWidth
12 import androidx.compose.foundation.layout.height
13 import androidx.compose.foundation.layout.padding
14 import androidx.compose.foundation.layout.size
15 import androidx.compose.foundation.layout.width
16 import androidx.compose.foundation.layout.wrapContentSize
17 import androidx.compose.foundation.lazy.LazyColumn
18 import androidx.compose.foundation.shape.RoundedCornerShape
19 import androidx.compose.material3.Button
20 import androidx.compose.material3.Card
21 import androidx.compose.material3.CardDefaults
22 import androidx.compose.material3.Text
23 import androidx.compose.runtime.Composable
24 import androidx.compose.runtime.collectAsState
25 import androidx.compose.ui.Alignment
26 import androidx.compose.ui.Modifier
27 import androidx.compose.ui.platform.LocalContext
28 import androidx.compose.ui.res.painterResource
29 import androidx.compose.ui.text.font.FontWeight

```

```
29 import androidx.compose.ui.text.style.TextOverflow
30 import androidx.compose.ui.unit.dp
31 import androidx.compose.ui.unit.sp
32 import androidx.navigation.NavController
33 import com.example.mobilelegendcharacterlist.viewmodel.HeroViewModel
34 import timber.log.Timber
35
36 @Composable
37 fun HomeScreen(viewModel: HeroViewModel, navController: NavController)
38 {
39     val heroList = viewModel.heroList.collectAsState().value
40     LazyColumn(
41         modifier = Modifier
42             .fillMaxWidth()
43             .padding(16.dp)
44     ) {
45         items(heroList.size) { index ->
46             val hero = heroList[index]
47             HeroCard(hero.name, hero.image, hero.url,
48 hero.description, navController, viewModel)
49         }
50     }
51 }
52
53
54 @Composable
55 fun HeroCard(
56     name: String,
57     image: Int,
58     url: String,
59     description: String,
60     navController: NavController,
61     viewModel: HeroViewModel
62 ) {
63     val context = LocalContext.current
64
65     Card(
66         modifier = Modifier
67             .fillMaxWidth()
68             .padding(8.dp),
69         shape = RoundedCornerShape(12.dp),
70         elevation = CardDefaults.cardElevation(defaultElevation =
71 4.dp)
72     ) {
73         Row(
74             modifier = Modifier
75                 .padding(16.dp)
76                 .fillMaxWidth(),
77             verticalAlignment = Alignment.CenterVertically
78         ) {
79             Image(

```

```
80         contentDescription = name,
81         modifier = Modifier
82             .size(width = 100.dp, height = 120.dp)
83     )
84
85     Spacer(modifier = Modifier.width(12.dp))
86
87     Column(
88         modifier = Modifier.weight(1f)
89     ) {
90         Text(text = name, fontWeight = FontWeight.Bold,
91         fontSize = 18.sp)
92         Text(
93             text = description,
94             fontSize = 12.sp,
95             maxLines = 3,
96             overflow = TextOverflow.Ellipsis
97         )
98
99         Spacer(modifier = Modifier.height(8.dp))
100
101        Row(
102            modifier = Modifier
103                .fillMaxWidth()
104                .padding(top = 8.dp),
105                horizontalArrangement = Arrangement.Center
106        ) {
107            Button(
108                onClick = {
109                    viewModel.logHeroClick(name)
110                    val intent = Intent(Intent.ACTION_VIEW,
111                     Uri.parse(url))
112                    context.startActivity(intent)
113
114                },
115                shape = RoundedCornerShape(50),
116                contentPadding = PaddingValues(horizontal =
117                    24.dp, vertical = 8.dp),
118                modifier = Modifier
119                    .wrapContentWidth()
120            ) {
121                Text("Detail", fontSize = 14.sp)
122            }
123
124         Spacer(modifier = Modifier.width(12.dp))
125
126         Button(
127             onClick = {
128                 viewModel.logHeroClick(name)
129                 val encodedDesc = Uri.encode(description)
130
131                 navController.navigate("penjelasan/${encodedDesc}/$image")
```

```

132         },
133         shape = RoundedCornerShape(50),
134         contentPadding = PaddingValues(horizontal =
135             16.dp, vertical = 8.dp),
136         modifier = Modifier
137             .wrapContentWidth()
138     ) {
139         Text("Penjelasan", fontSize = 12.sp)
140     }
141 }
142 }
143 }
144 }
145 }
```

DetailScreen.kt

Tabel 18 Source Code Jawaban soal 1 Compose

```

1 package com.example.mobilelegendcharacterlist.ui.screen
2
3 import androidx.compose.foundation.Image
4 import androidx.compose.foundation.layout.Column
5 import androidx.compose.foundation.layout.Spacer
6 import androidx.compose.foundation.layout.fillMaxSize
7 import androidx.compose.foundation.layout.fillMaxWidth
8 import androidx.compose.foundation.layout.height
9 import androidx.compose.foundation.layout.padding
10 import androidx.compose.material3.Text
11 import androidx.compose.runtime.Composable
12 import androidx.compose.ui.Alignment
13 import androidx.compose.ui.Modifier
14 import androidx.compose.ui.res.painterResource
15 import androidx.compose.ui.unit.dp
16 import androidx.compose.ui.unit.sp
17
18 @Composable
19 fun DetailScreen(description: String, image: Int) {
20     Column(
21         modifier = Modifier
22             .fillMaxSize()
23             .padding(16.dp),
24         horizontalAlignment = Alignment.CenterHorizontally
25     ) {
26         Image(
27             painter = painterResource(id = image),
28             contentDescription = null,
29             modifier = Modifier
30                 .fillMaxWidth()
31                 .height(200.dp)
```

```

31    )
32
33    Spacer(modifier = Modifier.height(16.dp))
34
35    Text(
36        text = description,
37        fontSize = 16.sp
38    )
39
40 }

```

HeroViewModel.kt

Tabel 19 Source Code Jawaban soal 1 Compose

```

1 package com.example.mobilelegendcharacterlist.viewmodel
2
3 import androidx.lifecycle.ViewModel
4 import com.example.mobilelegendcharacterlist.model.DataHero
5 import com.example.mobilelegendcharacterlist.data.HeroData
6 import kotlinx.coroutines.flow.MutableStateFlow
7 import kotlinx.coroutines.flow.StateFlow
8 import kotlinx.coroutines.flow.asStateFlow
9 import timber.log.Timber
10
11 class HeroViewModel : ViewModel() {
12
13     private val _heroList =
14         MutableStateFlow<List<DataHero>>(emptyList())
15     val heroList: StateFlow<List<DataHero>> = _heroList.asStateFlow()
16
17     init {
18         loadHeroes()
19     }
20
21     private fun loadHeroes() {
22         Timber.d("Memuat data hero...")
23         _heroList.value = HeroData.heroList
24     }
25
26     fun logHeroClick(name: String) {
27         Timber.d("Tombol 'Detail' diklik untuk hero: $name")
28     }
29 }

```

HeroViewModelFactory.kt

Tabel 20 Source Code Jawaban soal 1 Compose

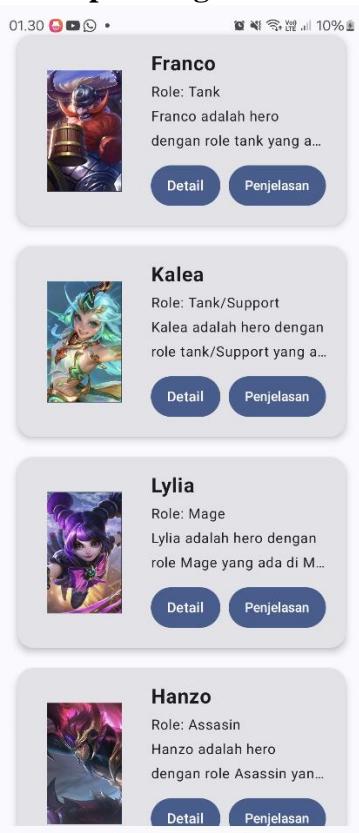
```
1 package com.example.mobilelegendcharacterlist.viewmodel
2
3 import androidx.lifecycle.ViewModel
4 import androidx.lifecycle.ViewModelProvider
5
6 class HeroViewModelFactory(
7     private val title: String
8 ) : ViewModelProvider.Factory {
9
10    override fun <T : ViewModel> create(modelClass: Class<T>): T {
11        if (modelClass.isAssignableFrom(HeroViewModel::class.java)) {
12            return HeroViewModel() as T // ganti jika constructor
13            butuh parameter
14            }
15        throw IllegalArgumentException("Unknown ViewModel class")
16    }
17 }
```

HeroApp.kt

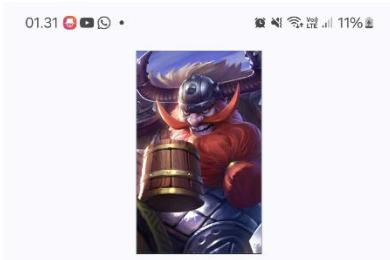
Tabel 21 Source Code Jawaban soal 1 Compose

```
1 package com.example.mobilelegendcharacterlist
2
3 import android.app.Application
4 import timber.log.Timber
5
6 class HeroApp: Application() {
7     override fun onCreate() {
8         super.onCreate()
9         Timber.plant(Timber.DebugTree())
10    }
11 }
```

B. Output Program



Gambar 5 Screenshot Hasil Jawaban Soal 1



Role: Tank

Franco adalah hero dengan role tank yang ada di Mobile Legends: Bang Bang. Dia memiliki Skill 1 yaitu 'Iron Hook', Skill 2 yaitu 'Fury Shock', dan Skill 3 yaitu 'Brutal Massacre', dan memiliki pasif yaitu 'Annihilation'.

Gambar 6 Screenshot Hasil Jawaban Soal 1

```
private fun loadHeroes() {
    Timber.d("Memuat data hero...")
    _heroList.value = HeroData.heroList _heroList: kotlinx.coroutines.flow.StateFlowImpl@8b8a68f
}

fun logHeroClick(name: String) { name: "Franco"
    Timber.d("Tomhol 'Detail' diklik untuk hero: $name") name: "Franco"
}
```

The screenshot shows the Android Studio interface with the code editor open to MainActivity.kt. The code uses Timber for logging. The bottom part of the screen shows the debugger, threads, and variables, indicating a running application named "main".

Gambar 7 Timber Button

C. Pembahasan

MainActivity.kt:

Pada baris 1, package com.example.mobilelegendcharacterlist menyatakan bahwa file ini berada dalam package utama aplikasi yang menggunakan Jetpack Compose. Pada baris 3, import android.os.Bundle digunakan untuk mengimpor class Bundle, yang diperlukan untuk menyimpan data sementara saat aktivitas

dibuat. Pada baris 4, `import androidx.activity.ComponentActivity` mengimpor class dasar dari aktivitas untuk Compose, yaitu `ComponentActivity`. Pada baris 5, `import androidx.activity.compose.setContent` digunakan untuk mengatur konten UI dari activity menggunakan deklaratif Compose. Pada baris 6, `import androidx.activity.enableEdgeToEdge` digunakan untuk memungkinkan UI menampilkan elemen hingga ke tepi layar.

Pada baris 7, `import androidx.compose.foundation.layout.fillMaxSize` mengimpor modifier untuk membuat komponen UI mengisi seluruh ukuran layar. Pada baris 8, `import androidx.lifecycle.viewmodel.compose.viewModel` digunakan untuk mengambil instance ViewModel dalam konteks Compose. Pada baris 9, `import androidx.compose.material3.*` mengimpor semua komponen dari Material 3 seperti `Surface`, `MaterialTheme`, dan `colorScheme`. Pada baris 10, `import androidx.compose.ui.Modifier` digunakan untuk menerapkan properti layout terhadap elemen UI.

Pada baris 11, `import com.example.mobilelegendcharacterlist.ui.theme.MobileLegendCharacterListTheme` mengimpor tema khusus aplikasi yang didefinisikan di file `theme`, untuk menyatukan tampilan warna dan typography aplikasi. Pada baris 12, `import com.example.mobilelegendcharacterlist.viewmodel.HeroViewModel` mengimpor class ViewModel yang berfungsi untuk menyimpan dan mengelola data karakter. Pada baris 13, `import com.example.mobilelegendcharacterlist.viewmodel.HeroViewModelFactory` mengimpor factory untuk membuat ViewModel yang membutuhkan parameter di konstruktor.

Pada baris 14, `import androidx.navigation.compose.rememberNavController` mengimpor fungsi untuk membuat dan mengingat instance NavController untuk navigasi antar layar Compose. Pada baris 15, `import androidx.navigation.compose.NavHost` digunakan untuk mendefinisikan struktur navigasi aplikasi. Pada baris 16, `import androidx.navigation.compose.composable` digunakan untuk mendefinisikan rute (route) individual untuk setiap composable screen. Pada baris 17, `import androidx.navigation.NavType` digunakan untuk menentukan tipe argumen yang digunakan dalam navigasi. Pada baris 18, `import androidx.navigation.navArgument` digunakan untuk mendeklarasikan argumen navigasi seperti `description` dan `image`.

Pada baris 19–20, `import com.example.mobilelegendcharacterlist.ui.screen.DetailScreen`

dan HomeScreen mengimpor composable yang menampilkan masing-masing tampilan detail karakter dan daftar karakter.

Pada baris 22, dideklarasikan class MainActivity yang merupakan subclass dari ComponentActivity, class ini digunakan sebagai entry point dari aplikasi berbasis Jetpack Compose. Pada baris 23–48, method onCreate() di-override untuk mengatur konten utama UI Compose saat aplikasi pertama kali dijalankan.

Pada baris 24, super.onCreate(savedInstanceState) memanggil implementasi dari superclass ComponentActivity. Pada baris 25, enableEdgeToEdge() memungkinkan konten tampil hingga ke tepi layar untuk pengalaman pengguna yang lebih imersif.

Pada baris 27, setContent digunakan untuk mendeklarasikan UI dengan Jetpack Compose. Semua komponen UI didefinisikan di dalam blok lambda setContent.

Pada baris 28, MobileLegendCharacterListTheme digunakan untuk membungkus seluruh tampilan dengan tema Material 3 yang telah ditentukan. Ini memastikan konsistensi gaya UI di seluruh aplikasi.

Pada baris 29, val navController = rememberNavController() digunakan untuk membuat dan menyimpan instance NavController, yang diperlukan untuk navigasi antar composable.

Pada baris 32, ViewModel HeroViewModel diinisialisasi dengan menggunakan viewModel() dan mengirimkan HeroViewModelFactory("List Hero ML") sebagai factory. Ini memungkinkan kita untuk menggunakan parameter konstruktor di ViewModel dalam konteks Compose.

Pada baris 34–36, Surface digunakan sebagai wadah UI utama, dan modifier fillMaxSize() memastikan komponen mengisi seluruh layar. Warna latar belakang diatur menggunakan MaterialTheme.colorscheme.background.

Pada baris 37–47, NavHost digunakan untuk mendefinisikan rute-rute navigasi dalam aplikasi. navController digunakan sebagai pengendali navigasi, dan startDestination ditetapkan ke "heroList", artinya tampilan pertama yang muncul adalah daftar hero.

Pada baris 38–40, composable("heroList") mendefinisikan rute ke halaman utama HomeScreen, yang menerima viewModel dan navController sebagai parameter. Ini adalah tampilan daftar karakter.

Pada baris 41–46, didefinisikan rute kedua penjelasan/{description}/{image} dengan dua argumen yaitu description dan image. Baris 43 menyatakan bahwa description bertipe String, dan baris 44 menyatakan bahwa image bertipe Int.

Pada baris 45, argumen dari `backStackEntry` diambil menggunakan `getString()` dan `getInt()`. Pada baris 46, `DetailScreen` dipanggil dengan `description` dan `image` sebagai parameter.

DataHero.kt

Pada baris 1, `package com.example.mobilelegendcharacterlist.model` menyatakan bahwa file ini berada di dalam package `model`, yang biasanya digunakan untuk menyimpan class-class data (data model) dalam arsitektur aplikasi Android. Package ini berfungsi sebagai tempat penyimpanan representasi data yang akan digunakan oleh UI maupun `ViewModel`.

Pada baris 3, `data class DataHero` dideklarasikan sebagai data class, yaitu class khusus di Kotlin yang secara otomatis menyediakan fungsi-fungsi seperti `toString()`, `equals()`, `hashCode()`, dan `copy()`. Class ini digunakan untuk merepresentasikan satu entitas data karakter Mobile Legends.

Pada baris 4–7, terdapat empat properti utama dari `DataHero`:

- Pada baris 4, `val name: String` menyimpan nama dari karakter Mobile Legends dalam bentuk string.
- Pada baris 5, `val image: Int` menyimpan ID resource dari gambar karakter, yang mengacu pada `drawable` (contoh: `R.drawable.hero1`).
- Pada baris 6, `val url: String` menyimpan link atau tautan eksternal yang mengarah ke informasi lebih lanjut tentang karakter.
- Pada baris 7, `val description: String` menyimpan deskripsi atau informasi singkat tentang karakter yang bersangkutan.

Keempat properti ini akan digunakan dalam aplikasi sebagai sumber data utama untuk menampilkan informasi hero di halaman daftar maupun detail.

HeroData.kt

Pada baris 1, `package com.example.mobilelegendcharacterlist.data` menyatakan bahwa file ini berada di dalam package `data`, yang biasanya digunakan untuk menyimpan class atau object yang bertanggung jawab menyediakan dan mengelola data aplikasi. Pada baris 3, `import com.example.mobilelegendcharacterlist.R` digunakan untuk mengimpor class `R` yang merepresentasikan semua resource aplikasi, termasuk gambar `drawable`. Pada baris 4, `import com.example.mobilelegendcharacterlist.model.DataHero` digunakan untuk mengimpor data class `DataHero` yang mewakili struktur data hero Mobile Legends.

Pada baris 6, dideklarasikan object `HeroData`, yang merupakan singleton object Kotlin. Object ini digunakan sebagai penyedia data statis untuk daftar hero. Karena menggunakan object, maka `HeroData` akan selalu memiliki satu instance di seluruh aplikasi.

Pada baris 7, dideklarasikan properti `heroList` bertipe `List<DataHero>`. Ini adalah daftar hero yang telah diisi secara manual (hardcoded) di dalam kode.

Pada baris 8–30, terdapat enam item `DataHero`, masing-masing merepresentasikan satu karakter Mobile Legends. Setiap item berisi empat parameter:

- `name`: nama hero dalam bentuk `String`.
- `image`: ID resource dari gambar hero (misalnya `R.drawable.hero1`).
- `url`: link eksternal ke halaman resmi hero di situs Mobile Legends.
- `description`: penjelasan lengkap mengenai role dan skill hero.

Pada baris 9, `DataHero(name = "Franco", ...)` digunakan untuk menambahkan hero bernama Franco, seorang tank dengan skill seperti "Iron Hook" dan "Brutal Massacre". Pada baris 13, `DataHero(name = "Kalea", ...)` mendeskripsikan Kalea sebagai hero tank/support dengan skill seperti "Wavebreaker" dan "Tsunami Slam".

Pada baris 17, `DataHero(name = "Lylia", ...)` merupakan hero mage dengan skill seperti "Magic Shockwave" dan "Black Shoes". Pada baris 21, `DataHero(name = "Hanzo", ...)` dijelaskan sebagai hero assassin dengan skill bertema ninja.

Pada baris 25, `DataHero(name = "Lancelot", ...)` juga merupakan assassin dengan skill seperti "Puncture" dan "Phantom Execution". Pada baris 29, `DataHero(name = "Lukas", ...)` adalah hero bertipe fighter dengan skill seperti "Flash Combo" dan "Unleash the Beast".

Semua data hero tersebut disimpan dalam list `heroList` dan dapat digunakan langsung di dalam aplikasi untuk menampilkan daftar karakter ke dalam UI.

HeroData.kt

Pada baris 1, `package com.example.mobilelegendcharacterlist.data` menyatakan bahwa file ini berada di dalam package `data`, yang biasanya digunakan untuk menyimpan class atau object yang bertanggung jawab menyediakan dan mengelola data aplikasi. Pada baris 3, `import com.example.mobilelegendcharacterlist.R` digunakan untuk mengimpor class `R` yang merepresentasikan semua resource aplikasi, termasuk gambar drawable. Pada baris 4, `import com.example.mobilelegendcharacterlist.model.DataHero` digunakan untuk mengimpor data class `DataHero` yang mewakili struktur data hero Mobile Legends.

Pada baris 6, dideklarasikan object `HeroData`, yang merupakan singleton object Kotlin. Object ini digunakan sebagai penyedia data statis untuk daftar hero. Karena menggunakan object, maka `HeroData` akan selalu memiliki satu instance di seluruh aplikasi.

Pada baris 7, dideklarasikan properti `heroList` bertipe `List<DataHero>`. Ini adalah daftar hero yang telah diisi secara manual (hardcoded) di dalam kode.

Pada baris 8–30, terdapat enam item `DataHero`, masing-masing merepresentasikan satu karakter Mobile Legends. Setiap item berisi empat parameter:

- `name`: nama hero dalam bentuk `String`.
- `image`: ID resource dari gambar hero (misalnya `R.drawable.hero1`).
- `url`: link eksternal ke halaman resmi hero di situs Mobile Legends.
- `description`: penjelasan lengkap mengenai role dan skill hero.

Pada baris 9, `DataHero(name = "Franco", ...)` digunakan untuk menambahkan hero bernama Franco, seorang tank dengan skill seperti "Iron Hook" dan "Brutal Massacre". Pada baris 13, `DataHero(name = "Kalea", ...)` mendeskripsikan Kalea sebagai hero tank/support dengan skill seperti "Wavebreaker" dan "Tsunami Slam".

Pada baris 17, `DataHero(name = "Lylia", ...)` merupakan hero mage dengan skill seperti "Magic Shockwave" dan "Black Shoes". Pada baris 21, `DataHero(name = "Hanzo", ...)` dijelaskan sebagai hero assassin dengan skill bertema ninja.

Pada baris 25, `DataHero(name = "Lancelot", ...)` juga merupakan assassin dengan skill seperti "Puncture" dan "Phantom Execution". Pada baris 29, `DataHero(name = "Lukas", ...)` adalah hero bertipe fighter dengan skill seperti "Flash Combo" dan "Unleash the Beast".

Semua data hero tersebut disimpan dalam list `heroList` dan dapat digunakan langsung di dalam aplikasi untuk menampilkan daftar karakter ke dalam UI.

HomeScreen.kt

Pada baris 1, `package com.example.mobilelegendcharacterlist.ui.screen` menyatakan bahwa file ini berada dalam package `ui.screen`, yang berfungsi untuk menyimpan layar (screen) utama dan detail berbasis Jetpack Compose. Pada baris 3–5, `import android.content.Intent` dan `import android.net.Uri` digunakan untuk membuat dan membuka intent eksplisit ke browser dengan URL dari data hero. Pada baris 6–15, berbagai `import androidx.compose.foundation.layout.*` digunakan untuk mengatur tata letak komponen UI secara fleksibel seperti `Row`, `Column`, `Spacer`, dan `Modifier`.

Pada baris 16, `import androidx.compose.foundation.lazy.LazyColumn` digunakan untuk membuat daftar scrollable secara efisien dalam Compose, seperti RecyclerView. Pada baris 17, `import androidx.compose.foundation.shape.RoundedCornerShape` digunakan untuk membentuk sudut elemen menjadi melengkung. Pada baris 18–19, `import androidx.compose.material3.Button` dan `Card` digunakan untuk membuat komponen tombol dan kartu dengan Material 3. Pada baris 20, `import androidx.compose.material3.Text` digunakan untuk menampilkan teks.

Pada baris 21–22, `import androidx.compose.runtime.Composable` dan `collectAsState` digunakan untuk mendeklarasikan fungsi UI Compose dan mengamati StateFlow dari ViewModel. Pada baris 23–27, `import androidx.compose.ui.*` digunakan untuk mengatur properti tampilan seperti ukuran (dp), font, dan overflow teks. Pada baris 28, `import androidx.navigation.NavController` digunakan untuk mengontrol navigasi antar composable dengan Navigation Compose. Pada baris 29, `import com.example.mobilelegendcharacterlist.viewmodel.HeroViewModel` mengimpor ViewModel yang digunakan untuk mengelola data hero. Pada baris 30, `import timber.log.Timber` digunakan untuk mencatat log aktivitas ke logcat.

Pada baris 32, dideklarasikan fungsi `HomeScreen` sebagai fungsi `@Composable`. Fungsi ini menampilkan daftar hero dalam bentuk `LazyColumn`. Pada baris 33, `val heroList = viewModel.heroList.collectAsState().value` digunakan untuk mengambil data dari StateFlow di dalam ViewModel dan mengubahnya menjadi state Compose.

Pada baris 34–39, `LazyColumn` digunakan untuk menampilkan daftar hero secara vertikal. Modifier `fillMaxWidth()` membuat daftar mengisi seluruh lebar layar, sedangkan `padding(16.dp)` memberi jarak dari tepi. Pada baris 36–38, dilakukan iterasi terhadap seluruh `heroList`, dan untuk setiap elemen, fungsi `HeroCard()` dipanggil dengan parameter `name`, `image`, `url`, `description`, `controller`, dan `viewModel`.

Pada baris 41–91, didefinisikan composable `HeroCard()` untuk menampilkan satu hero dalam bentuk kartu. Parameter `name`, `image`, `url`, dan `description` adalah data hero, `navController` digunakan untuk navigasi, dan `viewModel` digunakan untuk logging.

Pada baris 42, `val context = LocalContext.current` digunakan untuk mendapatkan konteks saat ini untuk menjalankan Intent.

Pada baris 44–49, `Card` digunakan sebagai container UI. Modifier mengatur ukuran dan `padding`, `RoundedCornerShape` membentuk sudut, dan `CardDefaults.cardElevation()` memberi efek bayangan.

Pada baris 50–52, Row digunakan sebagai layout horizontal, dan verticalAlignment = Alignment.CenterVertically memastikan semua item di tengah secara vertikal.

Pada baris 53–56, Image () digunakan untuk menampilkan gambar hero berdasarkan image dari drawable. Ukurannya diatur dengan Modifier.size().

Pada baris 58, Spacer(modifier = Modifier.width(12.dp)) memberikan jarak antara gambar dan kolom teks.

Pada baris 60–71, Column digunakan untuk menampilkan teks nama dan deskripsi hero:

- Pada baris 61, Text(text = name) menampilkan nama hero dengan font tebal.
- Pada baris 62–66, Text kedua menampilkan deskripsi, dengan maxLines = 3 dan overflow = TextOverflow.Ellipsis untuk memotong teks jika terlalu panjang.
- Spacer digunakan untuk memberi jarak sebelum tombol.

Pada baris 72–88, Row digunakan untuk meletakkan dua Button secara horizontal di bagian bawah kartu.

- Pada baris 74–80, tombol **Detail** dibuat. Saat diklik, akan mencatat log dengan viewModel.logHeroClick(name) dan membuka browser dengan Intent.ACTION_VIEW menggunakan URL hero.
- Pada baris 82–88, tombol **Penjelasan** dibuat. Saat diklik, akan mencatat log, lalu encode deskripsi dan melakukan navigasi ke route penjelasan/{description}/{image} menggunakan navController.navigate().

DetailScreen.kt

Pada baris 1, package com.example.mobilelegendcharacterlist.ui.screen menyatakan bahwa file ini berada dalam package ui.screen, yang digunakan untuk menyimpan file-file tampilan layar (screen) dari aplikasi berbasis Jetpack Compose.

Pada baris 3, import androidx.compose.foundation.Image mengimpor komponen Image dari Jetpack Compose yang digunakan untuk menampilkan gambar. Pada baris 4–8, import androidx.compose.foundation.layout.* digunakan untuk mengatur layout seperti Column, Spacer, dan modifier ukuran seperti fillMaxSize, fillMaxWidth, dan height.

Pada baris 9, `import androidx.compose.material3.Text` mengimpor komponen Text dari Material 3 yang digunakan untuk menampilkan teks ke layar. Pada baris 10, `import androidx.compose.runtime.Composable` menandai fungsi sebagai composable, artinya fungsi ini menghasilkan UI dalam paradigma deklaratif Jetpack Compose.

Pada baris 11, `import androidx.compose.ui.Alignment` digunakan untuk menyusun elemen UI secara horizontal dalam layout Column. Pada baris 12, `import androidx.compose.ui.Modifier` digunakan untuk menerapkan properti layout dan gaya terhadap elemen UI. Pada baris 13, `import androidx.compose.ui.res.painterResource` digunakan untuk mengambil gambar dari resource drawable berdasarkan ID. Pada baris 14–15, `import androidx.compose.ui.unit.*` digunakan untuk mengatur ukuran seperti padding (dp) dan ukuran teks (sp).

Pada baris 17, fungsi `DetailScreen()` dideklarasikan sebagai fungsi `@Composable`. Fungsi ini menerima dua parameter: `description` bertipe String untuk menampilkan deskripsi hero, dan `image` bertipe Int sebagai ID resource gambar hero.

Pada baris 18–23, `Column` digunakan sebagai layout vertikal yang menyusun elemen UI dari atas ke bawah. Modifier `fillMaxSize()` membuat column mengisi seluruh layar, dan `padding(16.dp)` memberikan jarak dari tepi layar. Properti `horizontalAlignment = Alignment.CenterHorizontally` memastikan semua elemen di dalam column disejajarkan secara horizontal ke tengah.

Pada baris 24–28, `Image` digunakan untuk menampilkan gambar hero berdasarkan ID resource yang diterima sebagai parameter. Modifier `fillMaxWidth()` membuat gambar mengisi lebar layar, dan `height(200.dp)` membatasi tinggi gambar.

Pada baris 30, `Spacer(modifier = Modifier.height(16.dp))` digunakan untuk memberikan jarak vertikal antara gambar dan teks deskripsi.

Pada baris 32–34, `Text` digunakan untuk menampilkan deskripsi hero. Nilai `text` diambil dari parameter `description`, dan `fontSize = 16.sp` digunakan untuk mengatur ukuran teks agar mudah dibaca.

HeroViewModel.kt

Pada baris 1, `package com.example.mobilelegendcharacterlist.viewmodel` menyatakan bahwa file ini berada di dalam package `viewmodel`, yang digunakan untuk menyimpan class-

class ViewModel sebagai bagian dari arsitektur MVVM (Model-View-ViewModel). Pada baris 3, `import androidx.lifecycle.ViewModel` mengimpor class `ViewModel` dari AndroidX, yang merupakan class dasar untuk menyimpan dan mengelola UI-related data secara lifecycle-aware.

Pada baris 4, `import com.example.mobilelegendcharacterlist.model.DataHero` mengimpor data class `DataHero` yang merepresentasikan struktur data karakter hero dalam aplikasi. Pada baris 5, `import com.example.mobilelegendcharacterlist.data.HeroData` mengimpor object `HeroData` yang menyediakan daftar hero secara statis.

Pada baris 6–8, `import kotlinx.coroutines.flow.*` digunakan untuk mengimpor `StateFlow`, `MutableStateFlow`, dan `asStateFlow` dari Kotlin coroutines. `StateFlow` digunakan untuk mengelola dan mengobservasi state secara reaktif dalam Compose. Pada baris 9, `import timber.log.Timber` mengimpor library logging `Timber`, yang digunakan untuk mencatat log aktivitas ke Logcat.

Pada baris 11, dideklarasikan class `HeroViewModel` yang merupakan turunan dari `ViewModel`. Class ini bertugas untuk menyimpan dan menyediakan data daftar hero ke layer UI secara aman dan efisien.

Pada baris 13, variabel `_heroList` dideklarasikan sebagai `MutableStateFlow` dengan nilai awal berupa list kosong bertipe `DataHero`. `MutableStateFlow` memungkinkan perubahan data dari dalam `ViewModel`. Pada baris 14, variabel `heroList` bertipe `StateFlow<List<DataHero>>` digunakan untuk memberikan akses hanya-baca ke UI, dengan memanggil fungsi `asStateFlow()` agar tidak bisa dimodifikasi langsung dari luar.

Pada baris 17, blok `init` dijalankan saat `ViewModel` pertama kali dibuat. Di sini, fungsi `loadHeroes()` dipanggil untuk memuat daftar hero dari data lokal.

Pada baris 19–22, fungsi `loadHeroes()` digunakan untuk mengambil data hero dari `HeroData.heroList`. Pada baris 20, `Timber.d("Memuat data hero...")` digunakan untuk mencatat log proses pemuatan data. Pada baris 21, data hero yang tersedia disimpan ke dalam `_heroList.value`, yang otomatis akan memberitahu UI bahwa data telah berubah.

Pada baris 24–26, fungsi `logHeroClick(name: String)` digunakan untuk mencatat log saat pengguna menekan tombol **Detail** atau **Penjelasan**. Pada baris 25, `Timber.d("Tombol 'Detail' diklik untuk hero: $name")` mencetak nama hero ke Logcat sebagai bagian dari proses debugging dan pelacakan interaksi pengguna.

HeroViewModelFactory.kt

Pada baris 1, package com.example.mobilelegendcharacterlist.viewmodel menyatakan bahwa file ini berada dalam package.viewmodel, yang digunakan untuk menyimpan class-class ViewModel beserta factory-nya dalam pola arsitektur MVVM. Pada baris 3, import androidx.lifecycle.ViewModel mengimpor class ViewModel dari AndroidX, yang merupakan superclass dari semua ViewModel. Pada baris 4, import androidx.lifecycle.ViewModelProvider mengimpor class ViewModelProvider, yang digunakan untuk membuat dan mengelola instance ViewModel secara lifecycle-aware.

Pada baris 6, dideklarasikan class HeroViewModelFactory yang mengimplementasikan interface ViewModelProvider.Factory. Class ini berfungsi sebagai factory atau pembuat instance HeroViewModel, terutama jika HeroViewModel memerlukan parameter di dalam konstruktor.

Pada baris 7, variabel title dideklarasikan sebagai properti bertipe String yang akan diberikan saat membuat factory. Meskipun dalam kode ini title tidak digunakan, properti ini bisa disiapkan untuk penggunaan di masa depan jika HeroViewModel memerlukan parameter judul atau konteks lainnya.

Pada baris 9–13, method override fun <T : ViewModel> create(modelClass: Class<T>) : T digunakan untuk membuat dan mengembalikan instance dari ViewModel. Pada baris 10, dilakukan pengecekan apakah modelClass merupakan subclass dari HeroViewModel menggunakan isAssignableFrom(). Jika benar, maka pada baris 11, akan dikembalikan objek HeroViewModel() yang dikonversi ke tipe T. Komentar di samping return memberi catatan bahwa baris ini perlu disesuaikan jika konstruktor HeroViewModel membutuhkan parameter, misalnya title.

Pada baris 13, jika modelClass bukan HeroViewModel, maka akan dilemparkan IllegalArgumentException dengan pesan "Unknown ViewModel class", yang menunjukkan bahwa ViewModel yang diminta tidak dikenali oleh factory ini.

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.

Dalam arsitektur aplikasi Android, Application class adalah komponen penting yang digunakan untuk mengelola state global dari seluruh aplikasi. Kelas ini merupakan titik awal yang pertama kali diinisialisasi oleh sistem Android sebelum Activity, Service, atau komponen lainnya dijalankan. Fungsinya sangat beragam, mulai dari inisialisasi library pihak ketiga seperti Timber, Firebase, atau Dagger, hingga menyediakan context aplikasi yang bersifat global dan dapat digunakan sepanjang siklus hidup aplikasi. Context ini sangat berguna dalam berbagai kebutuhan, misalnya saat ViewModel atau komponen lain membutuhkan akses ke sumber daya aplikasi tanpa harus bergantung pada context milik Activity.

Selain itu, Application class juga sering dimanfaatkan untuk mencatat peristiwa global seperti logging dan analitik, serta mendukung konsistensi arsitektur, terutama dalam pola MVVM atau Clean Architecture. Sebagai contoh pada praktikum Modul 4, penggunaan Timber untuk logging diinisialisasi di dalam class turunan dari Application, sehingga fitur pencatatan log dapat berjalan di seluruh bagian aplikasi, baik dalam antarmuka berbasis XML maupun Jetpack Compose.

Dengan mendefinisikan Application class dan mendaftarkannya di dalam `AndroidManifest.xml`, aplikasi dapat mempertahankan dan menjalankan berbagai fitur penting yang telah dibuat pada modul sebelumnya, seperti logging interaksi pengguna, pencatatan event pada ViewModel, dan menjaga efisiensi manajemen dependensi. Hal ini menjadikan Application class sebagai fondasi penting dalam pengembangan aplikasi Android yang terstruktur, modular, dan mudah dikembangkan lebih lanjut.

Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/Easydaf/Praktikum_Mobile/tree/main/Modul3