

**LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4**



VIEWMODEL AND DEBUGGING

Oleh:

Muhammad Azwin Hakim

NIM. 2310817310012

**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 Azwin Hakim
NIM : 2310817310012

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Zulfa Auliya Akbar
NIM. 2210817210026

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

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI	3
DAFTAR GAMBAR.....	4
DAFTAR TABEL	5
SOAL 1	6
A. Source Code	7
B. Output Program.....	17
C. Pembahasan	19
D. Tautan Git	23

DAFTAR GAMBAR

Gambar 1. Contoh Penggunaan Debugger	6
Gambar 2. Screenshot Hasil Jawaban Soal 1	17
Gambar 3. Screenshot Hasil Jawaban Soal 1	18
Gambar 4. Screenshot Hasil Jawaban Soal 1	18
Gambar 5. Screenshot Contoh Menambahkan Breakpoint pada kode	22
Gambar 6. Screenshot Contoh Step Over Kebaris Selanjutnya	22
Gambar 7. Screenshot Contoh Step On Kebaris ini	22
Gambar 8. Screenshot Contoh Bagian Dalam Fungsi	23

DAFTAR TABEL

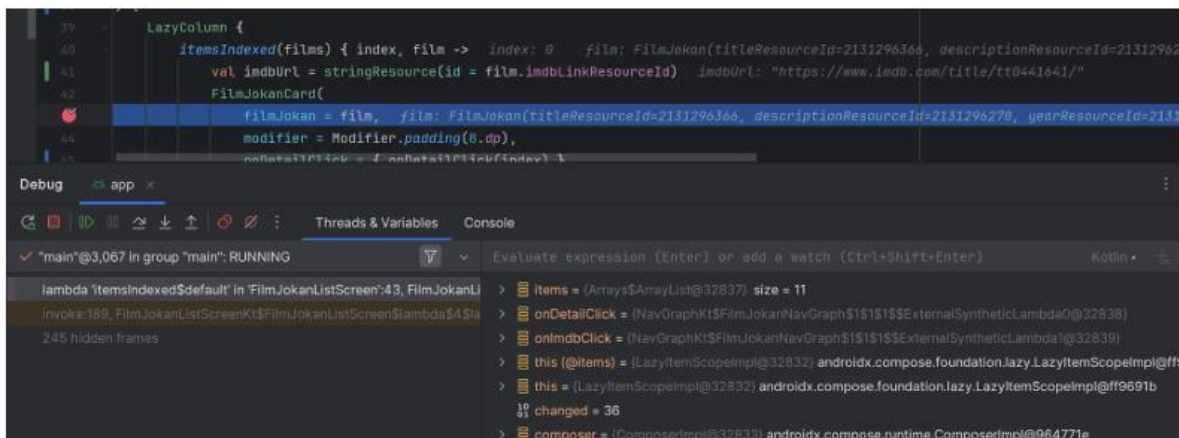
Tabel 1. Source Code Jawaban Soal 1.....	8
Tabel 2. Source Code Jawaban Soal 1.....	8
Tabel 3. Source Code Jawaban Soal 1.....	10
Tabel 4. Source Code Jawaban Soal 1.....	11
Tabel 5. Source Code Jawaban Soal 1.....	14
Tabel 6. Source Code Jawaban Soal 1.....	16

SOAL 1

Soal Praktikum:

1. Lanjutkan aplikasi Android berbasis XML dan Jetpack Compose yang sudah dibuat pada Modul 3 dengan menambahkan modifikasi sesuai ketentuan berikut:
 - a. Buatlah sebuah ViewModel untuk menyimpan dan mengelola data dari list item. Data tidak boleh disimpan langsung di dalam Fragment atau Activity.
 - b. Gunakan ViewModelFactory dalam pembuatan ViewModel
 - c. Gunakan StateFlow untuk mengelola event onClick dan data list item dari ViewModel ke Fragment
 - d. gunakan logging untuk event berikut:
 - a. Log saat data item masuk ke dalam list
 - b. Log saat tombol Detail dan tombol Explicit Intent ditekan
 - c. Log data dari list yang dipilih ketika berpindah ke halaman Detail
 - e. Gunakan tool Debugger di Android Studio untuk melakukan debugging pada aplikasi. Cari setidaknya satu breakpoint yang relevan dengan aplikasi. Lalu, gunakan fitur Step Into, Step Over, dan Step Out. Setelah itu, jelaskan fungsi Debugger, cara menggunakan Debugger, serta fitur Step Into, Step Over, dan Step Out
2. Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

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



Gambar 1. Contoh Penggunaan Debugger

A. Source Code

1. MainActivity.kt

```
1 package com.example.ultraman
2
3 import android.os.Bundle
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import androidx.compose.runtime.Composable
7 import androidx.lifecycle.viewmodel.compose.viewModel
8 import androidx.navigation.NavHostController
9 import androidx.navigation.compose.NavHost
10 import androidx.navigation.compose.composable
11 import androidx.navigation.compose.rememberNavController
12 import androidx.navigation.NavType
13 import androidx.navigation.navArgument
14 import com.example.ultraman.ui.ViewModel.UltramanViewModel
15 import
16     com.example.ultraman.ui.ViewModel.UltramanViewModelFactory
17 import com.example.ultraman.ui.screens.DetailScreen
18 import com.example.ultraman.ui.screens.ListScreen
19 import com.example.ultraman.ui.theme.UltramanTheme
20
21 class MainActivity : ComponentActivity() {
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContent {
25             UltramanTheme {
26                 val navController = rememberNavController()
27                 val viewModel: UltramanViewModel =
28                     viewModel(factory = UltramanViewModelFactory())
29                 AppNavHost(navController, viewModel)
30             }
31         }
32     }
33 }
34
35 @Composable
36 fun AppNavHost(navController: NavHostController, viewModel:
37     UltramanViewModel) {
38     NavHost(navController = navController, startDestination =
39     "list") {
40         composable("list") {
41             ListScreen(navController, viewModel)
42         }
43         composable(
44             "detail/{itemId}",
45             arguments = listOf(navArgument("itemId") { type =
46                 NavType.IntType })
47         ) { backStackEntry ->
48             val itemId =
49                 backStackEntry.arguments?.getInt("itemId")
50         }
```

44	DetailScreen(itemId, viewModel)
45	}
46	}
47	}

Tabel 1. Source Code Jawaban Soal 1

2. ListUltraman.kt

1	package com.example.ultraman.model
2	
3	data class ListUltraman(
4	val id: Int,
5	val Name: String,
6	val HumanHost: String,
7	val Height: String,
8	val Weight: String,
9	val Detail: String,
10	val imageUrl: String,
11	val link: String
12)

Tabel 2. Source Code Jawaban Soal 1

3. DataUltraman.kt

1	package com.example.ultraman
2	
3	import com.example.ultraman.model.ListUltraman
4	
5	val ultramans = listOf(
6	ListUltraman(
	id = 1,
	name = "Ultraman Noa",
	host = "Kazuki Komon",
	height = "55 m",
	weight = "55,000 t",
	description = "Ultraman Noa (ウルトラマンノア, Urutoraman Noa) is an ancient Ultraman from the World of N who pursued his evil counterpart, Dark Zagi, throughout various universes and countless alien worlds, including the World of the Land of Light. He is an Ultra who has been active across the multiverse gaining the moniker of a legendary figure for his power and feats. Noa is also the true form of Ultraman the Next/Ultraman Nexus.",
	imageUrl =
	"https://th.bing.com/th/id/OIP.1rd_goNDK6agElvRe4AMyAHaKX?rs=1&pid=ImgDetMain",
	wikiUrl = "https://ultra.fandom.com/wiki/Ultraman_Noa"
),
7	ListUltraman(
	id = 2,
	name = "Ultraman Nexus",

<p>8</p>	<pre> host = "Kazuki Komon", height = "49 m", weight = "40,000 t", description = "Ultraman Nexus (ウルトラマンネクサス, Urutoraman Nekusasu) is Ultraman Noa's second devolved form, and is the form a Dunamist assumes after using the Evoltruster. Starring in Ultraman Nexus, his story represented as the final phase of Ultra N Project. After evolving from Ultraman the Next, Nexus would resurface in 2008 to fight against a wave of Space Beast assaults, slowly growing in strength after bonding with multiple Dunamists and finally regaining his long lost form Ultraman Noa in his battle with Dark Zagi.", imageUrl = "https://pbs.twimg.com/media/FVhSSBHagAAWSju.jpg", wikiUrl = "https://ultra.fandom.com/wiki/Ultraman_Nexus_(character)"), ListUltraman(id = 3, name = "Ultraman Mebius", host = "Mirai Hibino", height = "49 m", weight = "35,000 t", description = "Ultraman Mebius (ウルトラマンメビウス, Urutoraman Mebiusu) is the protagonist of the eponymous series Ultraman Mebius. He was sent to Earth 25 years after Ultraman 80 to protect the planet from a new wave of monsters and aliens, many of which arrived due to Alien Empera's resurgence. During his tenure on Earth, he took the form of Mirai Hibino, and joined Crew GUYS.", imageUrl = "https://pbs.twimg.com/media/FVhVh22akAA69s3.jpg", wikiUrl = "https://ultra.fandom.com/wiki/Ultraman_Mebius_(character)"), </pre>
<p>9</p>	<pre> ListUltraman(id = 4, name = "Ultraman Tiga", host = "Daigo Madoka", height = "Micro ~ 53 m", weight = "44,000 t", description = "Ultraman Tiga (ウルトラマンティガ, Urutoraman Tiga) is the eponymous Ultra Hero of Ultraman Tiga. He is notable for being the first Ultraman in the franchise to be able to change forms, as well as being the first Ultra to appear in a televised series since Ultraman 80, ending a 15- year semi-hiatus.", imageUrl = "https://images-wixmp- ed30a86b8c4ca887773594c2.wixmp.com/f/aad45e4a-312a-469e-b82d- 3dc010e4a687/df0fafd-cc54df22-844a-424d-b730- af3c8caee3d3.jpg/v1/fill/w_1280,h_2157,q_75,strp/ultraman_tig </pre>

	<pre> a_dark_and_multi_by_ultrahorizongax2021_dfofafd- fullview.jpg", wikiUrl "https://ultra.fandom.com/wiki/Ultraman_Tiga_(character)"), 10 ListUltraman(id = 5, name = "Ultraman Belial", host = "Arie Ishikari", height = "55 m", weight = "60,000 t", description = "Ultraman Belial (ウルトラマンベリアル, Urutoraman Beriaru) was an evil Ultraman from the Land of Light, who was best known as Ultraman Zero's arch enemy and the father of Ultraman Geed. He was once a great fighter who took part in the Ultimate Wars, but his pride and greed got the better of him. This ultimately led him to commit the heinous crime of attempting to steal the Plasma Spark, for which he was banished. Injured and cast out, Belial was approached and transformed by Alien Reiblood into a Reionics, forever turning him to the darkness.\n" + 11 "Belial seemingly met his final end at the hands of his synthetic son, Ultraman Geed, in a climactic final battle that decided the fate of the universe. However, his existence continues to send ripples through the multiverse in the form of the Devil Splinters derived from his cells, as well as a parallel version of himself who lives on in the present day, having had his fate forever changed by a chance encounter with Absolute Tartarus.", "https://tsuburaya-prod.com/wp- content/uploads/2019/12/berial2.jpg", "https://ultra.fandom.com/wiki/Ultraman_Belial"), 12 ListUltraman(id = 6, name = "Ultraman Zero", host = "Run (ラン, Ran)", height = "Micro ~ 49 m", weight = "35,000 t", description = "Ultraman Zero (ウルトラマンゼロ, Urutoraman Zero) is the son of Ultraseven. He was trained under Ultraman Leo after he was banished from the Land of Light by his father for attempting to take the Plasma Spark for himself. His training ultimately led to his redemption in his battle with Belial.\n" + 13 "Zero is one of the most popular Ultras in the Ultraman Series' history, appearing many times over a decade since his debut.", "https://th.bing.com/th/id/OIP.E9iggNICwtyI7UR4WSD- 1wHaKm?rs=1&pid=ImgDetMain", 14 "https://ultra.fandom.com/wiki/Ultraman_Zero")) </pre>
--	---

Tabel 3. Source Code Jawaban Soal 1

4. UltramanViewModel.kt

```
1 package com.example.ultraman.ui.ViewModel
2
3 import android.util.Log
4 import androidx.lifecycle.ViewModel
5 import androidx.lifecycle.ViewModelProvider
6 import com.example.ultraman.model.ListUltraman
7 import com.example.ultraman.ultramans
8 import kotlinx.coroutines.flow.MutableStateFlow
9 import kotlinx.coroutines.flow.StateFlow
10
11 class UltramanViewModel : ViewModel() {
12
13     private val _ultramanList =
14     MutableStateFlow<List<ListUltraman>>(emptyList())
15     val ultramanList: StateFlow<List<ListUltraman>> =
16     _ultramanList
17
18     init {
19         _ultramanList.value = ultramans
20         Log.d("UltramanViewModel", "Data list dimasukkan ke
21         ViewModel (${ultramans.size} item)")
22     }
23
24     fun getItemById(id: Int): ListUltraman? {
25         return _ultramanList.value.find { it.id == id }
26     }
27 }
28
29 class UltramanViewModelFactory : ViewModelProvider.Factory {
30     @Suppress("UNCHECKED_CAST")
31     override fun <T : ViewModel> create(modelClass: Class<T>):
32     T {
33         return UltramanViewModel() as T
34     }
35 }
```

Tabel 4. Source Code Jawaban Soal 1

5. ListScreen.kt

```
1 package com.example.ultraman.ui.screens
2
3 import android.content.Intent
4 import android.util.Log
5 import androidx.compose.foundation.layout.Arrangement
6 import androidx.compose.foundation.layout.Column
7 import androidx.compose.foundation.layout.Row
8 import androidx.compose.foundation.layout.Spacer
9 import androidx.compose.foundation.layout.WindowInsets
```

```

10 import androidx.compose.foundation.layout.fillMaxSize
11 import androidx.compose.foundation.layout.fillMaxWidth
12 import androidx.compose.foundation.layout.height
13 import androidx.compose.foundation.layout.padding
14 import androidx.compose.foundation.layout.systemBars
15 import androidx.compose.foundation.layout.asPaddingValues
16 import androidx.compose.ui.text.style.TextOverflow
17 import androidx.compose.foundation.layout.width
18 import androidx.compose.foundation.lazy.LazyColumn
19 import androidx.compose.foundation.shape.RoundedCornerShape
20 import androidx.compose.material3.Button
21 import androidx.compose.material3.Card
22 import androidx.compose.material3.CardDefaults
23 import androidx.compose.material3.MaterialTheme
24 import androidx.compose.material3.Text
25 import androidx.compose.runtime.Composable
26 import androidx.compose.ui.Modifier
27 import androidx.compose.ui.draw.clip
28 import androidx.compose.ui.layout.ContentScale
29 import androidx.compose.ui.unit.dp
30 import androidx.navigation.NavHostController
31 import androidx.compose.foundation.lazy.items
32 import androidx.compose.runtime.collectAsState
33 import androidx.compose.ui.text.font.FontWeight
34 import androidx.core.net.toUri
35 import com.example.ultraman.ui.ViewModel.UltramanViewModel
36
37 @Composable
38 fun ListScreen(navController: NavHostController, viewModel:
UltramanViewModel) {
39     val ultramanList =
viewModel.ultramanList.collectAsState().value
40
41     LazyColumn(
42         modifier = Modifier
43             .fillMaxSize()
44             .padding(8.dp)
45
46         .padding(WindowInsets.systemBars.asPaddingValues())
47     ) {
48         items(ultramanList) { item ->
49             Card(
50                 shape = RoundedCornerShape(16.dp),
51                 elevation = CardDefaults.cardElevation(8.dp),
52                 modifier = Modifier
53                     .fillMaxWidth()
54                     .padding(vertical = 8.dp)
55             ) {
56                 Row(
57                     modifier = Modifier
58                         .padding(16.dp)
59                         .fillMaxWidth()

```

```

59         ) {
60             GlideImage(
61                 imageUrl = item.imageUrl,
62                 contentDescription = item.Name,
63                 modifier = Modifier
64                     .height(150.dp)
65                     .width(100.dp)
66                     .clip(RoundedCornerShape(12.dp)),
67                 contentScale = ContentScale.Crop
68             )
69
70             Spacer(modifier = Modifier.width(16.dp))
71
72             Column(
73                 modifier = Modifier.weight(1f)
74             ) {
75                 Row(
76                     modifier
77                     Modifier.fillMaxWidth(),
78                     horizontalArrangement
79                     Arrangement.SpaceBetween
80                     ) {
81                         Text(
82                             text = item.Name,
83                             style
84                             MaterialTheme.typography.titleMedium
85                             )
86                         Text(
87                             text = item.HumanHost,
88                             style
89                             MaterialTheme.typography.bodyMedium
90                             )
91                     }
92
93                     Spacer(modifier
94                     Modifier.height(4.dp))
95
96                     Row(
97                         modifier
98                         Modifier.fillMaxWidth(),
99                         horizontalArrangement
100                     Arrangement.SpaceBetween
101                     ) {
102                         Text(
103                             text = "Info: ",
104                             style
105                             MaterialTheme.typography.bodySmall,
106                             fontWeight = FontWeight.Bold
107                         )
108                         Text(
109                             text = item.Detail,

```

102	style	=
103	MaterialTheme.typography.bodySmall,	
104	maxLines = 4,	=
	overflow	
	TextOverflow.Ellipsis	
105)	
106	}	
107		
108	Spacer(modifier	=
	Modifier.height(8.dp))	
109		
110	Row(
111	horizontalArrangement	=
	Arrangement.spacedBy(30.dp)	
112) {	
113	Button(
114	onClick = {	
115	Log.d("ListScreen",	
	"Tombol Wiki ditekan untuk \${item.Name}")	
116	val intent	=
	Intent(Intent.ACTION_VIEW, item.link.toUri())	
117	navController.context.startActivity(intent)	
118	}	
119) {	
120	Text("Wiki")	
121	}	
122		
123	Button(
124	onClick = {	
125	Log.d("ListScreen",	
	"Tombol Detail ditekan untuk \${item.Name}")	
126	navController.navigate("detail/\${item.id}")	
127	}	
128) {	
129	Text("Detail")	
130	}	
131	}	
132	}	
133	}	
134	}	
135	}	
136	}	
137	}	

Tabel 5. Source Code Jawaban Soal 1

6. DetailScreen.kt

1	package com.example.ultraman.ui.screens
2	
3	import android.util.Log

```

4 import android.widget.ImageView
5 import androidx.compose.foundation.layout.Column
6 import androidx.compose.foundation.layout.Spacer
7 import androidx.compose.foundation.layout.WindowInsets
8 import androidx.compose.foundation.layout.asPaddingValues
9 import androidx.compose.foundation.layout.fillMaxSize
10 import androidx.compose.foundation.layout.fillMaxWidth
11 import androidx.compose.foundation.layout.height
12 import androidx.compose.foundation.layout.padding
13 import androidx.compose.foundation.layout.systemBars
14 import androidx.compose.foundation.rememberScrollState
15 import androidx.compose.foundation.shape.RoundedCornerShape
16 import androidx.compose.foundation.verticalScroll
17 import androidx.compose.material3.MaterialTheme
18 import androidx.compose.material3.Text
19 import androidx.compose.runtime.Composable
20 import androidx.compose.ui.Modifier
21 import androidx.compose.ui.draw.clip
22 import androidx.compose.ui.layout.ContentScale
23 import androidx.compose.ui.platform.LocalContext
24 import androidx.compose.ui.text.font.FontWeight
25 import androidx.compose.ui.text.style.TextAlign
26 import androidx.compose.ui.unit.dp
27 import androidx.compose.ui.viewinterop.AndroidView
28 import com.bumptech.glide.Glide
29 import com.example.ultraman.ui.ViewModel.UltramanViewModel
30
31 @Composable
32 fun DetailScreen(itemId: Int?, viewModel: UltramanViewModel) {
33     val item = viewModel.getItemById(itemId ?: -1)
34     Log.d("DetailScreen", "Navigasi ke DetailScreen untuk ID:
35     $itemId - ${item?.Name}")
36
37     item?.let {
38         Column(
39             modifier = Modifier
40                 .fillMaxSize()
41                 .padding(16.dp)
42
43             .padding(WindowInsets.systemBars.asPaddingValues())
44                 .verticalScroll(rememberScrollState())
45         ) {
46             GlideImage(
47                 imageUrl = it.imageUrl,
48                 contentDescription = it.Name,
49                 modifier = Modifier
50                     .fillMaxWidth()
51                     .height(600.dp)
52                     .clip(RoundedCornerShape(16.dp)),
53                 contentScale = ContentScale.Crop
54             )
55             Spacer(modifier = Modifier.height(12.dp))
56         }
57     }
58 }

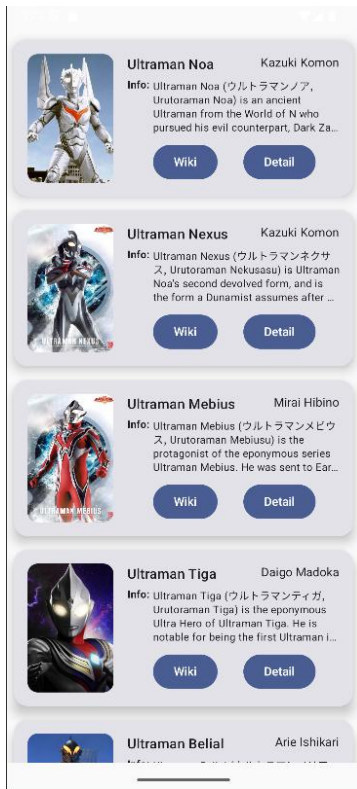
```

54	Text(it.Name,	style	=
	MaterialTheme.typography.headlineSmall,	fontWeight	=
	FontWeight.Bold)		
55	Text(text = "Human Host: ",	fontWeight	=
	FontWeight.Bold)		
56	Text(it.HumanHost)		
57	Text(text = "Height: ",	fontWeight	=
	FontWeight.Bold)		
58	Text(it.Height)		
59	Text(text = "Weight: ",	fontWeight	=
	FontWeight.Bold)		
60	Text(it.Weight)		
61	Text(text = "\nUltraman Info: ",	fontWeight	=
	FontWeight.Bold)		
62	Text(text = it.Detail,	textAlign	=
	TextAlign.Justify)		
63	}		
64	}		
65	}		
66			
67	@Composable		
68	fun GlideImage(
69	imageUrl: String,		
70	contentDescription: String?,		
71	modifier: Modifier = Modifier,		
72	contentScale: ContentScale = ContentScale.Crop		
73) {		
74	val context = LocalContext.current		
75	AndroidView(
76	factory = {		
77	ImageView(context).apply {		
78	scaleType = when (contentScale) {		
79	ContentScale.Crop		->
	ImageView.ScaleType.CENTER_CROP		
80	ContentScale.Fit		->
	ImageView.ScaleType.FIT_CENTER		
81	else -> ImageView.ScaleType.CENTER_CROP		
82	}		
83	contentDescription?.let		{
	this.contentDescription = it }		
84	}		
85	},		
86	update = {		
87	Glide.with(context)		
88	.load(imageUrl)		
89	.into(it)		
90	},		
91	modifier = modifier		
92)		
93	}		

Tabel 6. Source Code Jawaban Soal 1

B. Output Program

- Tampilan halaman list ultraman :



Gambar 2. Screenshot Hasil Jawaban Soal 1

- Tampilan Halaman detail ultraman :



Gambar 3. Screenshot Hasil Jawaban Soal 1



Gambar 4. Screenshot Hasil Jawaban Soal 1

C. Pembahasan

Pembahasan Kode :

1. MainActivity.kt

Di dalam onCreate, saya memanggil setContentView untuk mengatur isi UI menggunakan UltramanTheme agar tampilannya konsisten dengan tema aplikasi. Lalu saya membuat instance NavController dengan rememberNavController() untuk mengatur navigasi. Saya juga membuat UltramanViewModel dengan viewModel(factory = UltramanViewModelFactory()) agar data bisa dipisahkan dari UI dan mudah dikelola.

Navigasinya saya definisikan di fungsi AppNavHost, dengan dua rute utama:

- "list" untuk menampilkan daftar Ultraman melalui ListScreen.
- "detail/{itemId}" untuk menampilkan detail tiap Ultraman yang dipilih. Argumen itemId saya ambil dari back stack dan dikirim ke DetailScreen untuk menampilkan informasi lengkap berdasarkan ID-nya.

Struktur ini sengaja saya buat agar navigasi antar layar tetap sederhana tapi fleksibel, sekaligus mengikuti arsitektur MVVM yang rapi.

2. ListUltraman.kt

ListUltraman.kt ini fungsinya adalah sebagai bagian dari struktur data atau model dalam aplikasi Ultraman saya. Di dalamnya, saya mendefinisikan sebuah data class bernama ListUltraman. Data class ini berfungsi sebagai blueprint atau cetakan data untuk setiap karakter Ultraman yang akan saya tampilkan di aplikasi, baik dalam bentuk daftar maupun detail.

Data class ListUltraman memiliki delapan properti utama. Pertama, ada id bertipe Int sebagai penanda unik untuk setiap Ultraman. Kemudian, Name menyimpan nama Ultramanya, dan HumanHost menyimpan nama host manusianya jika ada. Dua properti berikutnya adalah Height dan Weight, yang menyimpan tinggi dan berat tubuh Ultraman dalam bentuk String, karena biasanya data ini datang dalam satuan teks seperti "49 m" atau "40,000 t".

Lalu ada properti Detail, yang berisi deskripsi atau penjelasan lengkap tentang Ultraman tersebut, serta imageUrl, yaitu link gambar yang akan saya tampilkan di aplikasi. Terakhir, saya juga menambahkan properti link, yang berisi URL ke halaman Wikipedia atau sumber lain jika pengguna ingin membaca lebih lanjut.

3. DataUltraman.kt

Di awal file, saya mengimpor ListUltraman dari package model, karena saya akan membuat daftar karakter Ultraman yang disusun menggunakan data class tersebut. Model ini sebelumnya sudah saya siapkan dengan properti seperti id, Name, HumanHost, Height, Weight, Detail, imageUrl, dan link.

Selanjutnya, saya mendeklarasikan variabel ultramans sebagai val, yang berarti nilainya tetap dan tidak bisa diubah. Variabel ini merupakan listOf dari objek ListUltraman, jadi isinya adalah daftar karakter Ultraman lengkap beserta semua data penting yang diperlukan untuk ditampilkan di UI, seperti nama, host manusianya, tinggi dan berat, deskripsi, gambar, serta link Wikipedia untuk informasi lebih lanjut.

Setiap item dalam list merepresentasikan satu karakter Ultraman. Misalnya, karakter pertama adalah Ultraman Noa dengan host bernama Kazuki Komon. Deskripsinya saya ambil dari sumber referensi fandom resmi agar informasinya akurat dan lengkap. Saya juga menyertakan URL gambar yang akan ditampilkan dalam aplikasi dan tautan ke halaman wiki untuk tiap Ultraman.

Data ini nantinya akan digunakan oleh tampilan ListScreen untuk menampilkan daftar karakter, dan saat salah satu dipilih, itemId dari masing-masing objek akan dikirim ke halaman DetailScreen untuk menampilkan informasi lengkap.

4. UltramanViewModel.kt

Kode ini saya buat sebagai ViewModel untuk menyimpan dan mengelola data Ultraman secara terpisah dari UI. Tujuannya adalah agar data tetap hidup meskipun terjadi rotasi layar atau perubahan konfigurasi lainnya, dan untuk mempermudah observasi data di Jetpack Compose.

Pada bagian atas, saya deklarasikan `_ultramanList` sebagai `MutableStateFlow`, lalu saya expose versi read-only-nya lewat `val ultramanList`. Ini berguna supaya composable bisa mengamati data tapi tidak bisa mengubahnya langsung, sesuai prinsip encapsulation.

Di dalam blok init, saya langsung isi `_ultramanList` dengan data dummy ultramans, yang berasal dari file statis. Saya juga menambahkan log agar saat debugging, saya tahu kapan dan berapa banyak data yang berhasil dimuat ke dalam ViewModel.

Saya juga buat fungsi `getItemById(id: Int)` agar bisa mengambil satu item berdasarkan id. Ini sangat berguna ketika navigasi ke halaman detail karena hanya perlu mengambil satu objek dari list, bukan mengoper semua data.

Terakhir, saya buat `UltramanViewModelFactory`, karena saat inject ViewModel ke dalam Compose, saya perlu factory custom untuk membuat instance-nya. Dengan cara ini, ViewModel saya bisa dipakai dengan `viewModel(factory = ...)`.

5. ListScreen.kt

Kode ini saya buat untuk menampilkan daftar karakter Ultraman di halaman utama aplikasi, menggunakan `LazyColumn` agar performanya tetap efisien meskipun datanya banyak. Setiap item dalam daftar ditampilkan dalam bentuk Card, yang di dalamnya berisi gambar, nama Ultraman, nama Human Host, deskripsi singkat, serta dua tombol aksi: Wiki dan Detail.

Saya ambil datanya dari `UltramanViewModel` menggunakan `collectAsState()` agar data tersebut bisa direaktifkan dan Compose bisa otomatis melakukan re-composition saat data berubah. Gambarnya saya tampilkan dengan `GlideImage`, yang saya buat sendiri menggunakan `AndroidView` dan `Glide`, karena `Glide` belum punya dukungan Compose secara langsung.

Teks "Info" saya buat dengan gaya lebih tebal menggunakan `fontWeight = FontWeight.Bold`, sementara detailnya dibatasi maksimal 4 baris agar tidak memakan terlalu banyak ruang dan diberi `Ellipsis` jika kepanjangan. Ini menjaga UI tetap rapi.

Untuk tombol Wiki, saya pakai `Intent.ACTION_VIEW` untuk membuka tautan eksternal (biasanya halaman Wikipedia atau sumber lain) di browser. Untuk tombol Detail, saya arahkan ke halaman detail menggunakan `navController.navigate("detail/${item.id}")`, dan ID tersebut nanti akan digunakan untuk mengambil data spesifik di halaman detail.

Saya juga menambahkan baris log menggunakan `Log.d()` sebagai alat bantu debugging yang sederhana namun efektif. Log ini dicetak saat pengguna menekan tombol "Wiki" atau "Detail" pada setiap item Ultraman. Untuk tombol "Wiki", log mencatat pesan bahwa tombol tersebut ditekan, lengkap dengan nama karakter Ultraman yang sedang dibuka, sehingga saya bisa memantau apakah intent menuju halaman Wikipedia berhasil dipicu. Begitu juga dengan tombol "Detail", saya mencatat interaksi pengguna saat mereka ingin melihat detail lebih lanjut tentang karakter tersebut. Log semacam ini sangat berguna saat pengembangan aplikasi karena memberikan gambaran langsung di logcat tentang bagaimana pengguna berinteraksi dengan UI. Dari log, saya bisa tahu apakah tombol-tombol berfungsi sesuai harapan dan dapat segera melacak jika terjadi kesalahan atau navigasi yang tidak berjalan sebagaimana mestinya.

6. DetailScreen.kt

Dalam kode `DetailScreen`, saya menggunakan `Log.d()` sebagai alat bantu untuk mencatat aktivitas navigasi ke layar detail. Log ini mencetak ID dari item yang dipilih, dan jika item berhasil ditemukan dari `ViewModel`, maka nama karakter Ultraman-nya juga ikut dicetak. Hal ini sangat membantu saya untuk memastikan bahwa data yang dikirim dari halaman sebelumnya (melalui navigasi) benar-benar sampai dan diproses dengan baik. Bila suatu saat halaman ini tidak menampilkan data, saya bisa langsung melihat di logcat apakah nilai `itemId` kosong, tidak sesuai, atau jika `getItemById()` gagal menemukan datanya. Dengan kata lain, log ini menjadi bukti bahwa proses pengambilan data sudah berjalan — atau tempat pertama yang saya periksa jika ada bug.

Selain itu, layar ini memanfaatkan komponen `GlideImage` untuk memuat gambar dari URL. Komponen ini menggunakan `AndroidView` agar bisa memakai `ImageView` klasik dan `Glide`, karena `Coil` memang belum saya gunakan di proyek ini. Ini memungkinkan gambar tetap dimuat dengan efisien meskipun saya menggunakan `Jetpack Compose` sebagai framework UI utama. Saya juga sengaja menggunakan `verticalScroll` untuk memastikan bahwa seluruh detail dari karakter Ultraman bisa terlihat meskipun panjang, dan styling teks saya atur agar rapi, dengan bagian deskripsi rata kiri-kanan menggunakan `TextAlign.Justify` untuk kenyamanan membaca.

Penjelasan fungsi Debugger, cara menggunakan Debugger, serta fitur Step Into, Step Over, dan Step Out

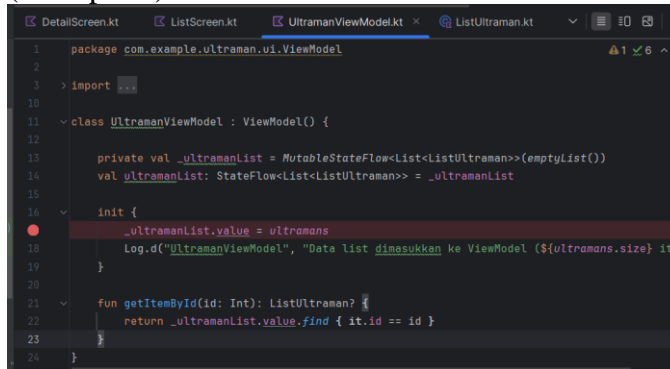
Debugger merupakan alat penting dalam pengembangan aplikasi Android yang digunakan untuk mendeteksi, melacak, dan memperbaiki kesalahan logika dalam kode program saat aplikasi berjalan. Tidak seperti log biasa yang hanya menampilkan hasil keluaran, debugger memberikan kontrol penuh kepada developer untuk menghentikan jalannya aplikasi di titik tertentu, memeriksa kondisi variabel, serta memahami alur eksekusi

secara real-time. Dengan begitu, debugger memungkinkan kita untuk menemukan bug yang tidak terlihat hanya dari hasil log.

Cara Menggunakan Debugger :

1. Pasang Breakpoint:

- Klik di sebelah kiri baris kode (gutter) untuk menambahkan titik merah (breakpoint).



Gambar 5. Screenshot Contoh Menambahkan Breakpoint pada kode

2. Jalankan dengan Mode Debug:

- Tekan tombol Debug di toolbar Android Studio, disamping tombol run.

3. Ketika Breakpoint Tercapai:

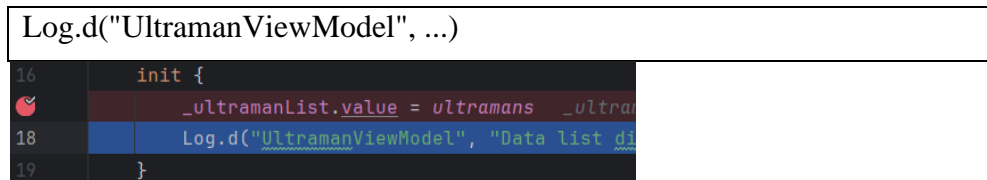
- Aplikasi akan pause di baris tersebut.

4. Menggunakan Step Into, Step Over, dan Step Out

Setelah aplikasi berhenti pada breakpoint, kamu bisa mulai menelusuri alur kode:

- Step Over (F8): Menjalankan baris kode saat ini dan langsung lompat ke baris berikutnya, tanpa masuk ke fungsi yang dipanggil.

Cocok digunakan di baris seperti ini:



Gambar 6. Screenshot Contoh Step Over Kebaris Selanjutnya

- Step Into (F7): Masuk ke dalam fungsi yang dipanggil di baris tersebut, jika kamu ingin menelusuri implementasi fungsi tersebut.

Misalnya :



Gambar 7. Screenshot Contoh Step On Kebaris ini

Jika ultramans adalah properti atau fungsi, kamu bisa menelusuri ke dalamnya.

```
1 package com.example.ultraman
2
3 import com.example.ultraman.model.ListUltraman
4
5 val ultramans = listOf(
6     ListUltraman(1, "Ultraman Noa", "Kazuki Komen", "55 m",
7     ListUltraman(2, "Ultraman Nexus", "Kazuki Komen", "49 m",
8     ListUltraman(3, "Ultraman Mebius", "Mira Hibino", "49 m",
9     ListUltraman(4, "Ultraman Tiga", "Daigo Madoka", "Micro",
10    ListUltraman(5, "Ultraman Belial", "Arie Ishikari", "55 m",
```

Gambar 8. Screenshot Contoh Bagian Dalam Fungsi

- Step Out (Shift + F8): Digunakan jika kamu sudah masuk terlalu dalam ke suatu fungsi dan ingin langsung keluar ke fungsi pemanggil sebelumnya.

Application class dalam arsitektur aplikasi Android dan fungsinya

Dalam arsitektur aplikasi Android, kelas Application berperan sebagai titik awal dan pusat dari siklus hidup aplikasi. Kelas ini merupakan turunan dari `android.app.Application` dan hanya dibuat sekali oleh sistem Android, sebelum Activity, Service, atau komponen lainnya dijalankan. Karena bersifat global dan memiliki rentang hidup sepanjang aplikasi berjalan, Application sering dimanfaatkan untuk menginisialisasi komponen-komponen penting yang digunakan secara luas di seluruh aplikasi.

Salah satu penggunaan utama kelas Application adalah untuk mengatur dependency injection, seperti Dagger atau Hilt, serta konfigurasi awal untuk Retrofit, Room, atau library lain yang dibutuhkan oleh berbagai bagian aplikasi. Selain itu, kelas ini juga kerap digunakan untuk menyimpan status global, seperti informasi pengguna yang sedang login, meskipun penggunaan ini harus dibatasi agar tidak menyebabkan memory leak.

Di luar itu, kelas Application juga menjadi tempat yang ideal untuk menginisialisasi library pihak ketiga, seperti Firebase, Crashlytics, Timber, atau layanan analitik lainnya. Bahkan, kamu bisa menambahkan logika khusus seperti mendeteksi apakah aplikasi sedang berada di foreground atau background, serta menjalankan observer untuk kondisi jaringan. Dengan kata lain, Application merupakan fondasi tempat berbagai inisialisasi dan konfigurasi awal dilakukan sebelum komponen UI dimulai.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

<https://github.com/Omelette719/Pemrograman-Mobile.git>