LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Muhammad Raihan NIM. 2310817110008

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE
MODUL 4

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

Nama Praktikan : Muhammad Raihan
NIM : 2310817110008

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Muhammad Raka Azwar NIM. 2210817210012	Eka Setya Wijaya S.T., M.Kom NIP. 198205082008011010

DAFTAR ISI

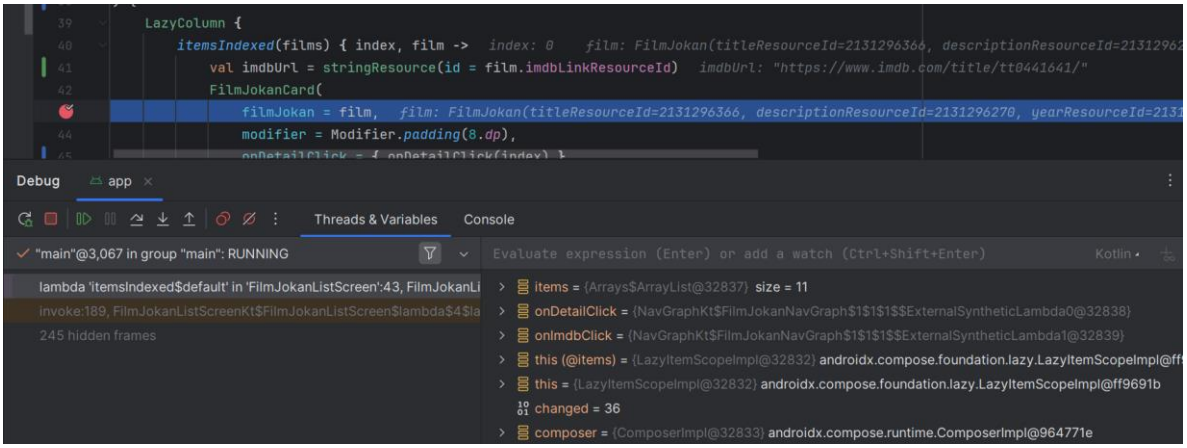
DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

Lanjutkan aplikasi Android berbasis XML dan Jetpack Compose yang sudah dibuat pada Modul 3 dengan menambahkan modifikasi sesuai ketentuan berikut:

a. Buatlah sebuah ViewModel untuk menyimpan dan mengelola data dari list item. Data tidak boleh disimpan langsung di dalam Fragment atau Activity.
b. Gunakan ViewModelFactory untuk membuat parameter dengan tipe data String di dalam ViewModel
c. Gunakan StateFlow untuk mengelola event onClick dan data list item dari ViewModel ke Fragment
d. Install dan gunakan library Timber untuk logging event berikut:
    a. Log saat data item masuk ke dalam list
    b. Log saat tombol Detail dan tombol Explicit Intent ditekan
    c. Log data dari list yang dipilih ketika berpindah ke halaman Detail
e. Gunakan tool Debugger di Android Studio untuk melakukan debugging pada aplikasi. Cari setidaknya satu breakpoint yang relevan dengan aplikasi. Lalu, gunakan fitur Step Into, Step Over, dan Step Out. Setelah itu, jelaskan fungsi Debugger, cara menggunakan Debugger, serta fitur Step Into, Step Over, dan Step Out



Gambar 1. Contoh Penggunaan Debugger

A. Source Code

1) Versi Jetpack Compose

Card.kt

Tabel 1. Source Code Jetpack Compose Card.kt Soal 1

```
1  package com.example.scrollablecompose.screens
2
3  import android.content.Context
```

```
4   import android.content.Intent
5   import android.net.Uri
6   import androidx.compose.foundation.Image
7   import
8   androidx.compose.foundation.layout.Arrangement
9   import androidx.compose.foundation.layout.Column
10  import
11  androidx.compose.foundation.layout.PaddingValues
12  import androidx.compose.foundation.layout.Row
13  import androidx.compose.foundation.layout.Spacer
14  import
15  androidx.compose.foundation.layout.fillMaxWidth
16  import androidx.compose.foundation.layout.height
17  import androidx.compose.foundation.layout.padding
18  import androidx.compose.foundation.layout.size
19  import androidx.compose.foundation.layout.width
20  import
21  androidx.compose.foundation.layout.wrapContentHeight
22  import
23  androidx.compose.foundation.shape.RoundedCornerShape
24  import androidx.compose.material3.Button
25  import androidx.compose.material3.Card
26  import androidx.compose.material3.CardDefaults
27  import androidx.compose.material3.MaterialTheme
28  import androidx.compose.material3.Text
29  import androidx.compose.runtime.Composable
30  import androidx.compose.ui.Alignment
31  import androidx.compose.ui.Modifier
32  import androidx.compose.ui.draw.clip
33  import androidx.compose.ui.layout.ContentScale
34  import androidx.compose.ui.res.painterResource
35  import androidx.compose.ui.res.stringResource
36  import androidx.compose.ui.text.font.FontWeight
37  import androidx.compose.ui.text.style.TextAlign
38  import androidx.compose.ui.unit.dp
39  import androidx.compose.ui.unit.sp
40  import com.example.scrollablecompose.R
41  import
42  com.example.scrollablecompose.model.KamenRider
43  import timber.log.Timber
44
45  @Composable
46  fun RiderCardPortrait(
47      rider: KamenRider,
48      context: Context,
49      onClick: () -> Unit
50  ){
```

```
51      Card(
52          modifier = Modifier
53              .padding(7.dp)
54              .fillMaxWidth()
55              .wrapContentHeight(),
56          shape = MaterialTheme.shapes.medium,
57          colors = CardDefaults.cardColors(
58              containerColor                                  =
59  MaterialTheme.colorScheme.surface
60          ),
61          elevation = CardDefaults.cardElevation(
62              defaultElevation = 5.dp
63          )
64      ) {
65          Row(
66              verticalAlignment = Alignment.Top,
67              modifier = Modifier.padding(5.dp)
68          ) {
69              Image(
70                  painter   =   painterResource(id   =
71  rider.posterRes),
72                  contentDescription     =     "Poster
73  ${rider.name}",
74                  modifier = Modifier
75                      .size(width  =  120.dp,  height  =
76  150.dp)
77                      .padding(8.dp)
78
79  .align(Alignment.CenterVertically)
80
81  .clip(RoundedCornerShape(15.dp)),
82                  contentScale                            =
83  ContentScale.FillBounds,
84              )
85              Column(Modifier.padding(8.dp)) {
86                  Row (
87                      modifier = Modifier
88                          .fillMaxWidth()
89                  ) {
90                      Text(
91                          text = rider.name,
92                          fontSize = 20.sp,
93                          fontWeight                      =
94  FontWeight.Bold,
95                          color                           =
96  MaterialTheme.colorScheme.onSurface,
97                          modifier = Modifier
```

```
98
99  .align(Alignment.CenterVertically)
100                             .weight(1f)
101                 )
102                 Spacer(Modifier.width(20.dp))
103                 Text(
104                     text                        =
105 rider.year.toString(),
106                     style                       =
107 MaterialTheme.typography.labelMedium,
108                     fontSize = 15.sp,
109                     color                       =
110 MaterialTheme.colorScheme.onSurface,
111                     modifier                    =
112 Modifier.align(Alignment.CenterVertically)
113                 )
114             }
115
116             Spacer(Modifier.height(8.dp))
117             Text(
118                 text                            =
119 stringResource(R.string.description),
120                 style                           =
121 MaterialTheme.typography.labelMedium,
122                 color                           =
123 MaterialTheme.colorScheme.onSurface
124             )
125             Text(
126                 text = rider.description,
127                 style                           =
128 MaterialTheme.typography.bodySmall,
129                 color                           =
130 MaterialTheme.colorScheme.onSurface,
131                 textAlign = TextAlign.Justify
132             )
133             Spacer(Modifier.height(8.dp))
134
135             Row (
136                 Modifier.fillMaxWidth(),
137                 horizontalArrangement           =
138 Arrangement.End
139             ) {
140                 Button(
141                     shape                       =
142 RoundedCornerShape(12.dp),
143                     contentPadding              =
144 PaddingValues(horizontal = 15.dp),
```

```
145                         onClick = {
146
147 Timber.tag("ListScreen").i("Tombol  Explicit  Intent
148 ditekan untuk: ${rider.name}, url: ${rider.imdbUrl}")
149                             val       intent       =
150 Intent(Intent.ACTION_VIEW)
151                             intent.data            =
152 Uri.parse(rider.imdbUrl)
153
154 intent.setPackage("com.android.chrome")
155
156 context.startActivity(intent)
157                         }
158                     ) {
159
160 Text(stringResource(R.string.imdb), fontSize = 13.sp)
161                     }
162                     Spacer(Modifier.width(10.dp))
163                     Button(
164                         shape                      =
165 RoundedCornerShape(12.dp),
166                         contentPadding             =
167 PaddingValues(horizontal = 15.dp),
168                         onClick = onClick
169                     ) {
170
171 Text(stringResource(R.string.detail),   fontSize   =
172 13.sp)
173                     }
174                 }
175             }
176         }
177     }
178 }
179
180 @Composable
181 fun RiderCardLandscape(
182     rider: KamenRider,
183     context: Context,
184     onClick: () -> Unit
185 ) {
186     Card(
187         modifier = Modifier
188             .padding(7.dp)
189             .fillMaxWidth()
190             .wrapContentHeight(),
191         shape = MaterialTheme.shapes.medium,
```

```
192         colors = CardDefaults.cardColors(
193             containerColor                                  =
194 MaterialTheme.colorScheme.surface
195         ),
196         elevation                                           =
197 CardDefaults.cardElevation(defaultElevation = 5.dp)
198     ) {
199         Row(
200             verticalAlignment = Alignment.Top,
201             modifier = Modifier.padding(5.dp)
202         ) {
203             Image(
204                 painter   =   painterResource(id   =
205 rider.posterRes),
206                 contentDescription     =     "Poster
207 ${rider.name}",
208                 modifier = Modifier
209                     .size(width  =  192.dp,  height  =
210 240.dp)
211                     .padding(8.dp)
212
213 .align(Alignment.CenterVertically)
214
215 .clip(RoundedCornerShape(15.dp)),
216                 contentScale                                =
217 ContentScale.FillBounds,
218             )
219             Spacer(Modifier.width(5.dp))
220             Column(Modifier.padding(8.dp)) {
221                 Row(modifier                                =
222 Modifier.fillMaxWidth()) {
223                     Text(
224                         text = rider.name,
225                         fontSize = 30.sp,
226                         fontWeight                          =
227 FontWeight.Bold,
228                         color                               =
229 MaterialTheme.colorScheme.onSurface,
230                         modifier = Modifier
231
232 .align(Alignment.CenterVertically)
233                             .weight(1f)
234                     )
235                     Spacer(Modifier.width(20.dp))
236                     Text(
237                         text                                =
238 rider.year.toString(),
```

```
239                         style                           =
240 MaterialTheme.typography.labelMedium,
241                         fontSize = 20.sp,
242                         color                           =
243 MaterialTheme.colorScheme.onSurface,
                            modifier                        =
    Modifier.align(Alignment.CenterVertically)
                        )
                    }

                    Spacer(Modifier.height(15.dp))
                    Text(
                        text                                =
    stringResource(R.string.description),
                        style                               =
    MaterialTheme.typography.labelMedium,
                        fontSize = 20.sp,
                        color                               =
    MaterialTheme.colorScheme.onSurface
                    )
                    Spacer(Modifier.height(5.dp))
                    Text(
                        text = rider.description,
                        style                               =
    MaterialTheme.typography.bodyMedium,
                        fontSize = 18.sp,
                        color                               =
    MaterialTheme.colorScheme.onSurface,
                        textAlign = TextAlign.Justify
                    )

                    Spacer(Modifier.height(50.dp))
                    Row(
                        Modifier.fillMaxWidth(),
                        horizontalArrangement               =
    Arrangement.End
                    ) {
                        Button(
                            shape                           =
    RoundedCornerShape(16.dp),
                            contentPadding                  =
    PaddingValues(vertical = 15.dp, horizontal = 20.dp),
                            onClick = {

    Timber.tag("ListScreen").i("Tombol  Explicit  Intent
    ditekan untuk: ${rider.name}, url: ${rider.imdbUrl}")
```

```
                                val       intent       =
Intent(Intent.ACTION_VIEW)
                                intent.data            =
Uri.parse(rider.imdbUrl)

intent.setPackage("com.android.chrome")

context.startActivity(intent)
                            }
                        ) {

Text(stringResource(R.string.imdb), fontSize = 18.sp)
                        }

                        Spacer(Modifier.width(10.dp))

                        Button(
                            shape                      =
RoundedCornerShape(16.dp),
                            contentPadding             =
PaddingValues(vertical = 15.dp, horizontal = 20.dp),
                            onClick = onClick
                        ) {

Text(stringResource(R.string.detail),   fontSize   =
18.sp)
                        }
                    }
                }
            }
        }
}
```

ListScreen.kt

Tabel 2. Source Code Jetpack Compose ListScreen.kt Soal 1

```
1   package com.example.scrollablecompose.screens
2
3   import android.content.res.Configuration
4   import
5   androidx.compose.foundation.layout.PaddingValues
6   import
7   androidx.compose.foundation.layout.fillMaxWidth
8   import androidx.compose.foundation.layout.padding
9   import androidx.compose.foundation.lazy.LazyColumn
10  import androidx.compose.foundation.lazy.items
11
```

```
12  import
13  androidx.compose.material3.ExperimentalMaterial3Api
14  import androidx.compose.material3.MaterialTheme
15  import androidx.compose.material3.Scaffold
16  import androidx.compose.material3.Text
17  import androidx.compose.material3.TopAppBar
18  import androidx.compose.material3.TopAppBarDefaults
19  import androidx.compose.runtime.Composable
20  import androidx.compose.runtime.LaunchedEffect
21  import androidx.compose.runtime.collectAsState
22  import androidx.compose.runtime.getValue
23  import androidx.compose.runtime.remember
24  import androidx.compose.ui.Modifier
25  import
26  androidx.compose.ui.platform.LocalConfiguration
27  import androidx.compose.ui.platform.LocalContext
28  import androidx.compose.ui.res.stringResource
29  import androidx.compose.ui.tooling.preview.Preview
30  import androidx.compose.ui.unit.dp
31  import
32  androidx.lifecycle.viewmodel.compose.viewModel
33  import androidx.navigation.NavController
34  import com.example.scrollablecompose.R
35  import com.example.scrollablecompose.Routes
36  import
37  com.example.scrollablecompose.ui.theme.ScrollableCom
38  poseTheme
39  import
40  com.example.scrollablecompose.viewmodel.RiderListVie
41  wModel
42  import
43  com.example.scrollablecompose.viewmodel.RiderListVie
44  wModelFactory
45  import timber.log.Timber
46
47  @OptIn(ExperimentalMaterial3Api::class)
48  @Composable
49  fun ListScreen(navController: NavController? = null)
50  {
51      val                    title                    =
52  stringResource(R.string.topappbarr_title)
53      val     factory     =     remember(title)     {
54  RiderListViewModelFactory(title) }
55      val     viewModel:     RiderListViewModel     =
56  viewModel(factory = factory, key = title)
57
58
```

```
    val                kamenRiderList                by
viewModel.riderList.collectAsState()
    val                 clickedRider                 by
viewModel.onClickRider.collectAsState()

    val                orientationMode                =
LocalConfiguration.current.orientation
    val context = LocalContext.current

    LaunchedEffect(clickedRider) {
        clickedRider?.let { rider ->

navController?.navigate(Routes.detailScreen         +
"/${rider.id}")
            viewModel.clearRiderClick()
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(title) },
                colors                                 =
TopAppBarDefaults.topAppBarColors(
                    containerColor                     =
MaterialTheme.colorScheme.primary,
                    titleContentColor                  =
MaterialTheme.colorScheme.onPrimary
                )
            )
        }
    ) { innerPadding ->
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .padding(innerPadding),
            contentPadding = PaddingValues(16.dp)
        ) {
            items(kamenRiderList) { rider ->
                val onClick = {

Timber.tag("ListScreen").i("Tombol   Detail   ditekan
untuk: ${rider.name}")
                    viewModel.onRiderClick(rider)
                }


```

```
                if (orientationMode == Configuration.ORIENTATION_PORTRAIT) {
                    RiderCardPortrait(rider = rider, context = context, onClick = onClick)
                } else {
                    RiderCardLandscape(rider = rider, context = context, onClick = onClick)
                }
            }
        }
    }
}

@Preview(
    showBackground = true,
    widthDp = 360,
    heightDp = 800
)
@Composable
fun ListScreenPortPreview() {
    ScrollableComposeTheme {
        ListScreen()
    }
}

//@Preview(
//    showBackground = true,
//    widthDp = 800,
//    heightDp = 360
//)
//@Composable
//fun ListScreenLandPreview() {
//    ScrollableComposeTheme {
//        ListScreen()
//    }
//}
```

DetailScreen.kt

Tabel 3. Source Code Jetpack Compose DetailScreen.kt Soal 1

```
1  package com.example.scrollablecompose.screens
2
3  import android.content.res.Configuration
4  import androidx.compose.foundation.Image
5  import androidx.compose.foundation.layout.Column
6  import
7  androidx.compose.foundation.layout.PaddingValues
```

```
8   import androidx.compose.foundation.layout.Row
9   import androidx.compose.foundation.layout.Spacer
10  import
11  androidx.compose.foundation.layout.fillMaxSize
12  import
13  androidx.compose.foundation.layout.fillMaxWidth
14  import androidx.compose.foundation.layout.height
15  import androidx.compose.foundation.layout.padding
16  import androidx.compose.foundation.layout.size
17  import androidx.compose.foundation.layout.width
18  import
19  androidx.compose.foundation.rememberScrollState
20  import
21  androidx.compose.foundation.shape.RoundedCornerShape
22  import androidx.compose.foundation.verticalScroll
23  import androidx.compose.material.icons.Icons
24  import
25  androidx.compose.material.icons.automirrored.filled.
26  ArrowBack
27  import
28  androidx.compose.material3.ExperimentalMaterial3Api
29  import androidx.compose.material3.Icon
30  import androidx.compose.material3.IconButton
31  import androidx.compose.material3.MaterialTheme
32  import androidx.compose.material3.Scaffold
33  import androidx.compose.material3.Text
34  import androidx.compose.material3.TopAppBar
35  import androidx.compose.material3.TopAppBarDefaults
36  import androidx.compose.runtime.Composable
37  import androidx.compose.runtime.collectAsState
38  import androidx.compose.runtime.remember
39  import androidx.compose.ui.Alignment
40  import androidx.compose.ui.Modifier
41  import androidx.compose.ui.draw.clip
42  import androidx.compose.ui.layout.ContentScale
43  import
44  androidx.compose.ui.platform.LocalConfiguration
45  import androidx.compose.ui.res.painterResource
46  import androidx.compose.ui.res.stringResource
47  import androidx.compose.ui.text.font.FontWeight
48  import androidx.compose.ui.text.style.TextAlign
49  import androidx.compose.ui.tooling.preview.Preview
50  import androidx.compose.ui.unit.dp
51  import androidx.compose.ui.unit.sp
52  import
53  androidx.lifecycle.viewmodel.compose.viewModel
54  import androidx.navigation.NavController
```

```
55  import com.example.scrollablecompose.R
56  import
57  com.example.scrollablecompose.model.KamenRider
58  import
59  com.example.scrollablecompose.model.KamenRiderReposi
60  tory.getKamenRiderList
61  import
62  com.example.scrollablecompose.ui.theme.ScrollableCom
63  poseTheme
64  import
65  com.example.scrollablecompose.viewmodel.DetailViewMo
66  del
67  import
68  com.example.scrollablecompose.viewmodel.DetailViewMo
69  delFactory
70
71  @OptIn(ExperimentalMaterial3Api::class)
72  @Composable
73  fun  DetailScreen(navController:  NavController?  =
74  null, id: Int) {
75      val       factory       =       remember       {
76  DetailViewModelFactory(id) }
77      val      viewModel:      DetailViewModel      =
78  viewModel(factory = factory)
79      val riderState = viewModel.rider.collectAsState()
80      val rider = riderState.value
81      val                 orientation                =
82  LocalConfiguration.current.orientation
83
84      Scaffold(
85          topBar = {
86              TopAppBar(
87                  title               =               {
88  Text(stringResource(R.string.detail)) },
89                  navigationIcon = {
90                      IconButton(onClick      =      {
91  navController?.navigateUp() }) {
92                          Icon(
93                              imageVector            =
94  Icons.AutoMirrored.Filled.ArrowBack,
95                              contentDescription     =
96  "Back"
97                          )
98                      }
99                  },
100                 colors                             =
101 TopAppBarDefaults.topAppBarColors(
```

```
                    containerColor                    =
MaterialTheme.colorScheme.primary,
                    titleContentColor                 =
MaterialTheme.colorScheme.onPrimary,
                    navigationIconContentColor        =
MaterialTheme.colorScheme.onPrimary
                )
            )
        }
    ) { padding ->
        if (rider != null) {
            if            (orientation             ==
Configuration.ORIENTATION_PORTRAIT) {
                DetailPortrait(rider, padding)
            } else {
                DetailLandscape(rider, padding)
            }
        } else {
            Text(
                text = "Data not found",
                fontSize = 25.sp,
                modifier = Modifier.padding(padding)
            )
        }
    }
}


@Composable
fun DetailPortrait(rider: KamenRider, paddingValues:
PaddingValues) {
    Column(
        modifier = Modifier
            .padding(paddingValues)
            .padding(16.dp)
            .verticalScroll(rememberScrollState())
            .fillMaxSize(),
        horizontalAlignment                           =
Alignment.CenterHorizontally
    ) {
        RiderImage(rider)
        Spacer(Modifier.height(15.dp))
        RiderTextInfo(rider)
    }
}


```

```
@Composable
fun DetailLandscape(rider: KamenRider, paddingValues:
PaddingValues) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .verticalScroll(rememberScrollState())
            .padding(paddingValues)
            .padding(16.dp)
    ) {
        RiderImage(rider,
Modifier.align(Alignment.CenterVertically))
        Spacer(Modifier.width(12.dp))
        RiderTextInfo(rider)
    }
}

@Composable
fun RiderImage(rider: KamenRider, modifier: Modifier
= Modifier) {
    Image(
        painter      =      painterResource(id      =
rider.posterRes),
        contentDescription = "Poster ${rider.name}",
        contentScale = ContentScale.FillBounds,
        modifier = modifier
            .size(width = 240.dp, height = 320.dp)
            .clip(RoundedCornerShape(20.dp))
    )
}

@Composable
fun RiderTextInfo(rider: KamenRider) {
    val                orientation                =
LocalConfiguration.current.orientation
    val     isPortrait     =     orientation     ==
Configuration.ORIENTATION_PORTRAIT

    Column (
        modifier = Modifier.fillMaxWidth()
    ) {
        Text(
            text = rider.name,
            fontSize = 28.sp,
            fontWeight = FontWeight.Bold,
            textAlign     =     if     (isPortrait)
TextAlign.Center else TextAlign.Start,
```

```
196             modifier = Modifier.fillMaxWidth()
197         )
198         Spacer(Modifier.height(12.dp))
199         Text(
200             text   =   stringResource(R.string.year,
201 rider.year),
202             fontSize = 18.sp,
203             fontWeight = FontWeight.Medium
204         )
        Spacer(Modifier.height(12.dp))
        Text(
            text                                        =
    stringResource(R.string.description),
            fontSize = 18.sp,
            fontWeight = FontWeight.Bold
        )
        Spacer(Modifier.height(4.dp))
        Text(
            text = rider.description,
            textAlign = TextAlign.Justify
        )
    }
    }



    @Preview(
        showBackground = true,
        widthDp = 390,
        heightDp = 800,
        name = "Redmi Note 13 Portrait"
    )
    @Composable
    fun DetailScreenPortraitPreview() {
        ScrollableComposeTheme {
            DetailScreen(null,
    getKamenRiderList()[0].id)
        }
    }

    //@Preview(
    //    showBackground = true,
    //    widthDp = 800,
    //    heightDp = 390,
    //    name = "Redmi Note 13 Landscape"
    //)
    //@Composable
```

```
//fun DetailScreenLandscapePreview() {
//    ScrollableComposeTheme {
//        val navController = rememberNavController()
//                        DetailScreen(navController,
getKamenRiderList()[0].id)
//    }
//}
```

RiderListViewModel.kt

Tabel 4. Source Code Jetpack Compose RiderListViewModel.kt Soal 1

```
1   package com.example.scrollablecompose.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import androidx.lifecycle.viewModelScope
6   import
7   com.example.scrollablecompose.model.KamenRider
8   import
9   com.example.scrollablecompose.model.KamenRiderReposi
10  tory
11  import kotlinx.coroutines.flow.MutableStateFlow
12  import kotlinx.coroutines.flow.StateFlow
13  import kotlinx.coroutines.launch
14  import timber.log.Timber
15
16  class RiderListViewModel(private val title: String) :
17  ViewModel() {
18
19      private         val         _riderList         =
20  MutableStateFlow<List<KamenRider>>(emptyList())
21      val   riderList:   StateFlow<List<KamenRider>>   =
22  _riderList
23
24      private        val        _onClickRider        =
25  MutableStateFlow<KamenRider?>(null)
26      val   onClickRider:   StateFlow<KamenRider?>   =
27  _onClickRider
28
29      init {
30          loadKamenRider()
31      }
32
33      private fun loadKamenRider() {
34          viewModelScope.launch {
35              val              allRiders              =
36  KamenRiderRepository.getKamenRiderList()
```

```
37              _riderList.value = allRiders
38              Timber.tag("ListViewModel").i("Data item
39  berhasil  dimuat  ke  dalam  list:  ${allRiders.size}
40  item")
41          }
42      }
43
44      fun onRiderClick(rider: KamenRider) {
45          _onClickRider.value = rider
46      }
47
48      fun clearRiderClick() {
49          _onClickRider.value = null
50      }
51  }

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

DetailViewModel.kt

Tabel 5. Source Code Jetpack Compose ViewModel.kt Soal 1

```
1   package com.example.scrollablecompose.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import androidx.lifecycle.viewModelScope
6   import
7   com.example.scrollablecompose.model.KamenRider
8   import
9   com.example.scrollablecompose.model.KamenRiderReposi
10  tory
11  import kotlinx.coroutines.flow.MutableStateFlow
12  import kotlinx.coroutines.flow.StateFlow
13  import kotlinx.coroutines.launch
```

```
14  import timber.log.Timber
15
16  class DetailViewModel(private val id: Int) :
17  ViewModel() {
18
19      private val _rider =
20  MutableStateFlow<KamenRider?>(null)
21      val rider: StateFlow<KamenRider?> get() = _rider
22
23      init {
24          loadRiderById()
25      }
26
27      private fun loadRiderById() {
28          viewModelScope.launch {
29              val selected =
30  KamenRiderRepository.getKamenRiderList().find {
31  it.id == id }
32              if (selected != null) {
33
34  Timber.tag("DetailViewModel").i("Navigasi ke
35  DetailScreen dengan data: ${selected.name}, tahun:
36  ${selected.year}")
37              } else {
38
39  Timber.tag("DetailViewModel").w("Rider dengan ID $id
40  tidak ditemukan")
41              }
42              _rider.value = selected
43          }
44      }
    }

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

TimberApp.kt

Tabel 6. Source Code Jetpack Compose TimberApp.kt Soal 1

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

2) Versi XML

MainAdapter.kt

Tabel 7. Source Code XML MainAdapter.kt

```
1  package com.example.scrollablexml.adapter
2
3  import android.annotation.SuppressLint
4  import android.view.LayoutInflater
5  import android.view.ViewGroup
6  import androidx.recyclerview.widget.RecyclerView
7  import
8  com.example.scrollablexml.databinding.AdapterMainBin
9  ding
10 import com.example.scrollablexml.model.KamenRider
11 import timber.log.Timber
12 import java.util.Locale
13
14 class MainAdapter(
15     initialList: List<KamenRider>,
16     private val onImdbClick: (String) -> Unit,
17     private val onDetailClick: (KamenRider) -> Unit
18 ) : RecyclerView.Adapter<MainAdapter.ViewHolder>() {
19
20     private          val          riderList          =
21 initialList.toMutableList()
22
23
```

```
    class ViewHolder(private val binding: AdapterMainBinding) :
RecyclerView.ViewHolder(binding.root) {
        fun bind(
            rider: KamenRider,
            onImdbClick: (String) -> Unit,
            onDetailClick: (KamenRider) -> Unit
        ) {

binding.riderImage.setImageResource(rider.posterRes)
            binding.riderName.text = rider.name
            binding.riderYear.text =
String.format(Locale.getDefault(), "%d", rider.year)
            binding.descBody.text =
rider.description
            binding.imdbBtn.setOnClickListener {
                Timber.tag("MainAdapter").i("Tombol
Eksplisit Intent ditekan untuk: ${rider.name}, URL:
${rider.imdbUrl}")
                onImdbClick(rider.imdbUrl)
            }
            binding.detailBtn.setOnClickListener {
                Timber.tag("MainAdapter").i("Tombol
Detail ditekan untuk: ${rider.name}, ID:
${rider.id}")
                onDetailClick(rider)
            }
        }
    }

    override fun onCreateViewHolder(parent:
ViewGroup, viewType: Int): ViewHolder {
        val binding =
AdapterMainBinding.inflate(LayoutInflater.from(paren
t.context), parent, false)
        return ViewHolder(binding)
    }

    override fun getItemCount(): Int = riderList.size

    override fun onBindViewHolder(holder: ViewHolder,
position: Int) {
        holder.bind(riderList[position],
onImdbClick, onDetailClick)
    }

    @SuppressLint("NotifyDataSetChanged")
```

```
    fun updateData(newList: List<KamenRider>) {
        riderList.clear()
        riderList.addAll(newList)
        notifyDataSetChanged()
    }
}
```

ListFragment.kt

Tabel 8. Source Code XML ListFragment.kt

```
1   package com.example.scrollablexml.fragment
2
3   import android.content.Intent
4   import android.net.Uri
5   import android.os.Bundle
6   import android.view.LayoutInflater
7   import android.view.View
8   import android.view.ViewGroup
9   import androidx.fragment.app.Fragment
10  import androidx.fragment.app.viewModels
11  import androidx.lifecycle.lifecycleScope
12  import
13  androidx.navigation.fragment.findNavController
14  import com.example.scrollablexml.adapter.MainAdapter
15  import
16  com.example.scrollablexml.databinding.FragmentListBi
17  nding
18  import
19  com.example.scrollablexml.viewmodel.RiderListViewMod
20  el
21  import
22  com.example.scrollablexml.viewmodel.RiderListViewMod
23  elFactory
24  import kotlinx.coroutines.flow.collectLatest
25  import kotlinx.coroutines.launch
26
27  class ListFragment : Fragment() {
28
29      private var _binding: FragmentListBinding? = null
30      private val binding get() = _binding!!
31
32      private  val  viewModel:  RiderListViewModel  by
33  viewModels {
34          RiderListViewModelFactory("Kamen       Rider
35  List")
36      }
37
```

```
38      private lateinit var mainAdapter: MainAdapter
39
40      override fun onCreateView(
41          inflater:     LayoutInflater,     container:
42  ViewGroup?,
43          savedInstanceState: Bundle?
44      ): View {
45          _binding                                    =
46  FragmentListBinding.inflate(inflater,     container,
47  false)
48
49          mainAdapter = MainAdapter(
50              initialList = emptyList(),
51              onImdbClick = { url ->
52                  val             intent             =
53  Intent(Intent.ACTION_VIEW, Uri.parse(url)).apply {
54                      setPackage("com.android.chrome")
55                  }
56                  startActivity(intent)
57              },
58              onDetailClick = { rider ->
59                  viewModel.onRiderClick(rider)
60              }
61          )
62
63          binding.recyclerView.adapter = mainAdapter
64
65          observeViewModel()
66
67          return binding.root
68      }
69
70      private fun observeViewModel() {
71          viewLifecycleOwner.lifecycleScope.launch {
72              viewModel.riderList.collectLatest { list
73  ->
74                  mainAdapter.updateData(list)
75              }
76          }
77
78          viewLifecycleOwner.lifecycleScope.launch {
79              viewModel.onClickRider.collectLatest   {
80  rider ->
                    rider?.let {
                        val           action           =
    ListFragmentDirections.actionListFragmentToDetailFra
    gment(it.id)
```

```
        findNavController().navigate(action)
                        viewModel.clearRiderClick()
                    }
                }
            }
        }

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

DetailFragment.kt

Tabel 9. Source Code XML DetailFragment.kt

```
1   package com.example.scrollablexml.fragment
2
3   import android.os.Bundle
4   import android.view.LayoutInflater
5   import android.view.View
6   import android.view.ViewGroup
7   import androidx.fragment.app.Fragment
8   import androidx.lifecycle.ViewModelProvider
9   import androidx.lifecycle.lifecycleScope
10  import androidx.navigation.fragment.navArgs
11  import com.example.scrollablexml.R
12  import
13  com.example.scrollablexml.databinding.FragmentDetail
14  Binding
15  import
16  com.example.scrollablexml.viewmodel.DetailViewModel
17  import
18  com.example.scrollablexml.viewmodel.DetailViewModelF
19  actory
20  import kotlinx.coroutines.flow.collectLatest
21  import kotlinx.coroutines.launch
22
23  class DetailFragment : Fragment() {
24
25      private var _binding: FragmentDetailBinding? =
26  null
27      private val binding get() = _binding!!
28
29      private val args: DetailFragmentArgs by navArgs()
30
```

```
31      private val viewModel: DetailViewModel by lazy {
32          val                 factory                 =
33  DetailViewModelFactory(args.riderId)
34          ViewModelProvider(this,
35  factory)[DetailViewModel::class.java]
36      }
37
38      override fun onCreateView(
39          inflater:     LayoutInflater,     container:
40  ViewGroup?,
41          savedInstanceState: Bundle?
42      ): View {
43          _binding                                    =
44  FragmentDetailBinding.inflate(inflater,   container,
45  false)
46
47
48  binding.topAppBar.setNavigationOnClickListener {
49
50  requireActivity().onBackPressedDispatcher.onBackPres
51  sed()
52          }
53
54          observeRider()
55
56          return binding.root
57      }
58
59      private fun observeRider() {
60          viewLifecycleOwner.lifecycleScope.launch {
61              viewModel.rider.collectLatest { rider ->
62                  rider?.let {
63
    binding.riderImage.setImageResource(it.posterRes)
                      binding.riderName.text = it.name
                      binding.riderYear.text          =
    getString(R.string.detail_year, it.year)
                      binding.descBody.text           =
    it.description
                  }
              }
          }
      }

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

```
    }
}
```

RiderListViewModel.kt

Tabel 10. Source Code XML RiderListViewModel.kt

```
1   package com.example.scrollablexml.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import androidx.lifecycle.viewModelScope
6   import com.example.scrollablexml.model.KamenRider
7   import
8   com.example.scrollablexml.model.KamenRiderRepository
9   import kotlinx.coroutines.flow.MutableStateFlow
10  import kotlinx.coroutines.flow.StateFlow
11  import kotlinx.coroutines.launch
12  import timber.log.Timber
13
14  class RiderListViewModel(private val title: String) :
15  ViewModel() {
16
17      private         val         _riderList         =
18  MutableStateFlow<List<KamenRider>>(emptyList())
19      val   riderList:   StateFlow<List<KamenRider>>   =
20  _riderList
21
22      private        val        _onClickRider        =
23  MutableStateFlow<KamenRider?>(null)
24      val   onClickRider:   StateFlow<KamenRider?>   =
25  _onClickRider
26
27      init {
28          loadKamenRider()
29      }
30
31      private fun loadKamenRider() {
32          viewModelScope.launch {
33              val              allRiders              =
34  KamenRiderRepository.getKamenRiderList()
35              _riderList.value = allRiders
36              Timber.tag("ListViewModel").i("Data item
37  berhasil  dimuat  ke  dalam  list:  ${allRiders.size}
38  item")
39          }
40      }
41
```

```
42      fun onRiderClick(rider: KamenRider) {
43          _onClickRider.value = rider
44      }
45
46      fun clearRiderClick() {
47          _onClickRider.value = null
48      }
49  }
50
51  class RiderListViewModelFactory(private val title:
    String) : ViewModelProvider.Factory {
        override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
            if
    (modelClass.isAssignableFrom(RiderListViewModel::cla
    ss.java)) {
                @Suppress("UNCHECKED_CAST")
                return RiderListViewModel(title) as T
            }
            throw      IllegalArgumentException("Unknown
    ViewModel class")
        }
    }
```

DetailViewModel.kt

Tabel 11. Source Code XML DetailViewModel.kt

```
1   package com.example.scrollablexml.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import androidx.lifecycle.viewModelScope
6   import com.example.scrollablexml.model.KamenRider
7   import
8   com.example.scrollablexml.model.KamenRiderRepository
9   import kotlinx.coroutines.flow.MutableStateFlow
10  import kotlinx.coroutines.flow.StateFlow
11  import kotlinx.coroutines.launch
12  import timber.log.Timber
13
14  class   DetailViewModel(private   val   id:   Int)   :
15  ViewModel() {
16
17      private          val          _rider          =
18  MutableStateFlow<KamenRider?>(null)
19      val rider: StateFlow<KamenRider?> get() = _rider
20
```

```
21      init {
22          loadRiderById()
23      }
24
25      private fun loadRiderById() {
26          viewModelScope.launch {
27              val              selected              =
28  KamenRiderRepository.getKamenRiderList().find      {
29  it.id == id }
30              if (selected != null) {
31
32  Timber.tag("DetailViewModel").i("Navigasi          ke
33  DetailScreen dengan data: ${selected.name}, tahun:
34  ${selected.year}")
35              } else {
36
37  Timber.tag("DetailViewModel").w("Rider dengan ID $id
38  tidak ditemukan")
39              }
40              _rider.value = selected
41          }
42      }
43  }
44
    class DetailViewModelFactory(private val id: Int) :
    ViewModelProvider.Factory {
        override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
            if
    (modelClass.isAssignableFrom(DetailViewModel::class.
    java)) {
                @Suppress("UNCHECKED_CAST")
                return DetailViewModel(id) as T
            }
            throw      IllegalArgumentException("Unknown
    ViewModel class")
        }
    }
```

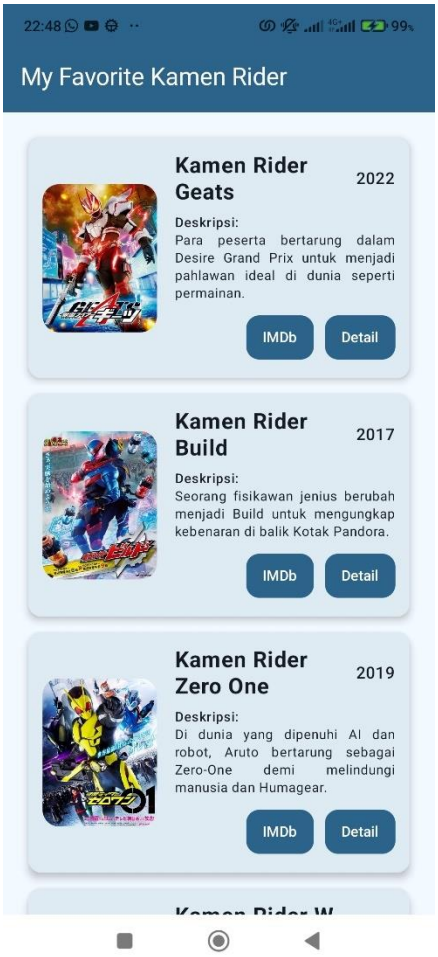
TimberApp.kt

Tabel 12. Source Code XML TimberApp.kt

```
1   package com.example.scrollablexml
2
3   import android.app.Application
4   import timber.log.Timber
5
```

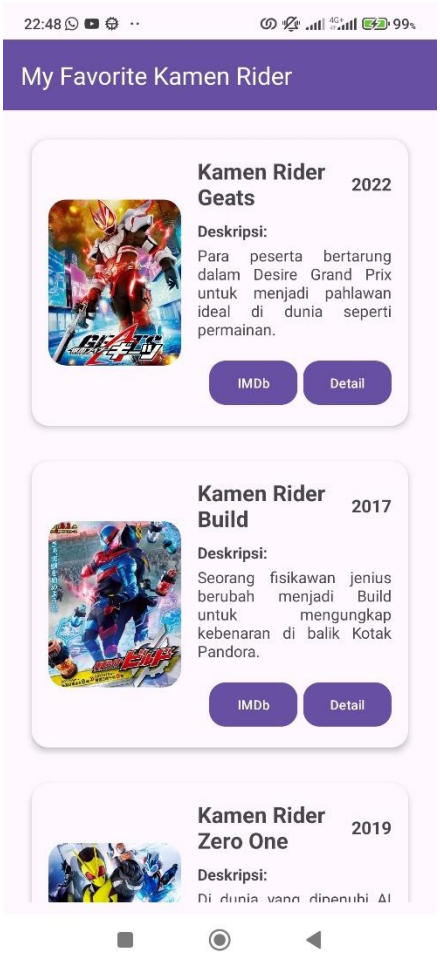
```
6   class TimberApp : Application() {
7       override fun onCreate() {
8           super.onCreate()
9
10          if(BuildConfig.DEBUG){
11              Timber.plant(Timber.DebugTree())
12          }
13      }
14  }
```

B. Output Program



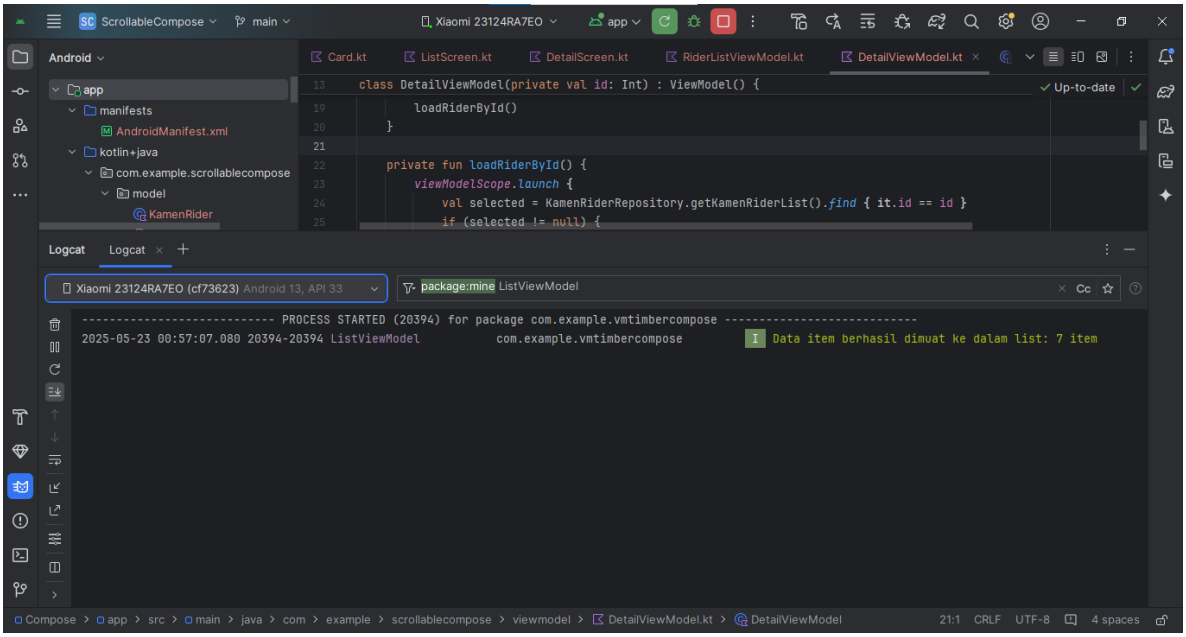
Gambar 2. Screenshot List Screen Soal 1 Versi Jetpack Compose

Gambar 3. Screenshot Detail Screen Soal 1 Versi Jetpack Compose

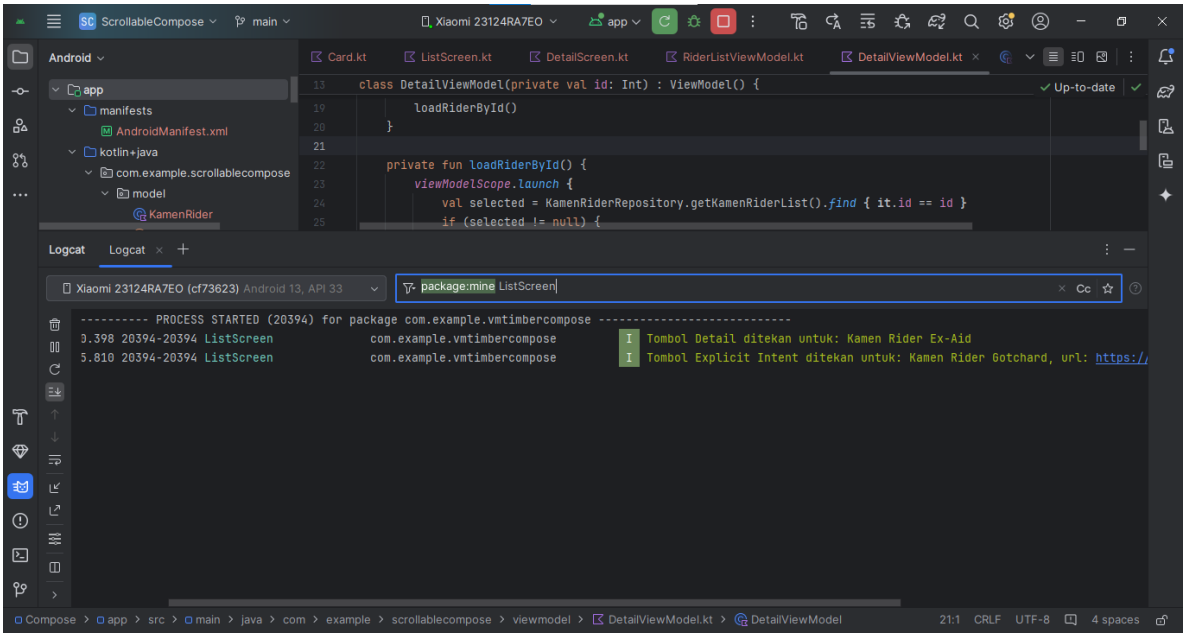


Gambar 4. Screenshot List Screen Soal 1 Versi XML

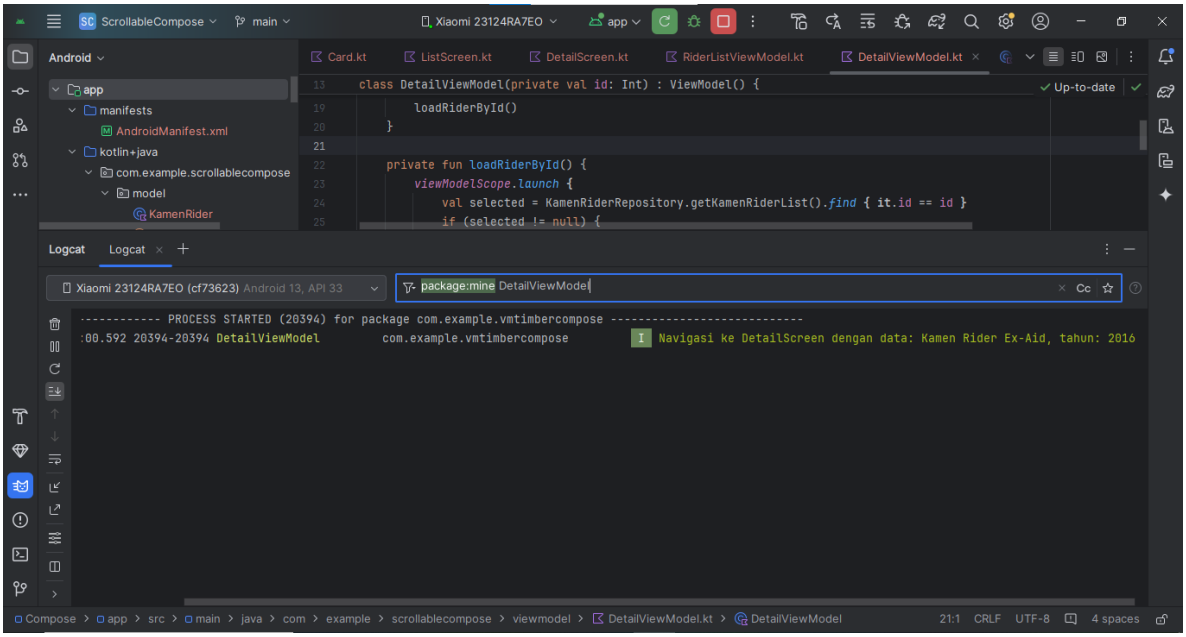
Gambar 5. Screenshot Detail Screen Soal 1 Versi XML



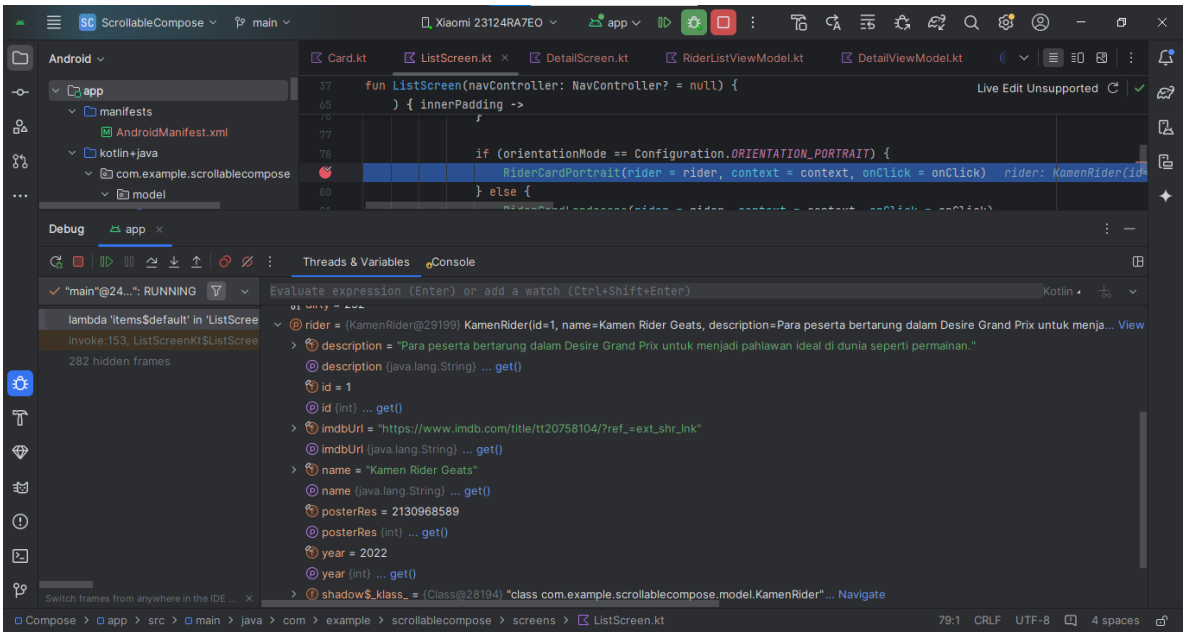
Gambar 6. Screenshot Log Saat Data Item Masuk Ke Dalam List Versi Compose



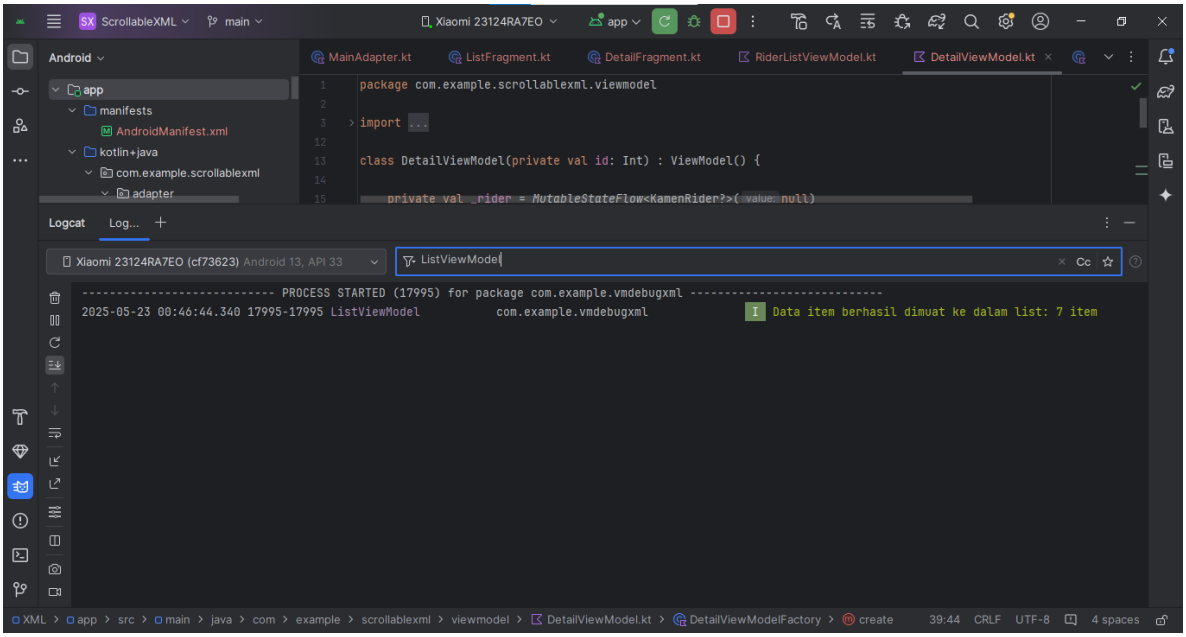
Gambar 7. Screenshot Log Saat Tombol Detail Dan Tombol Explicit Intent Ditekan Versi Compose



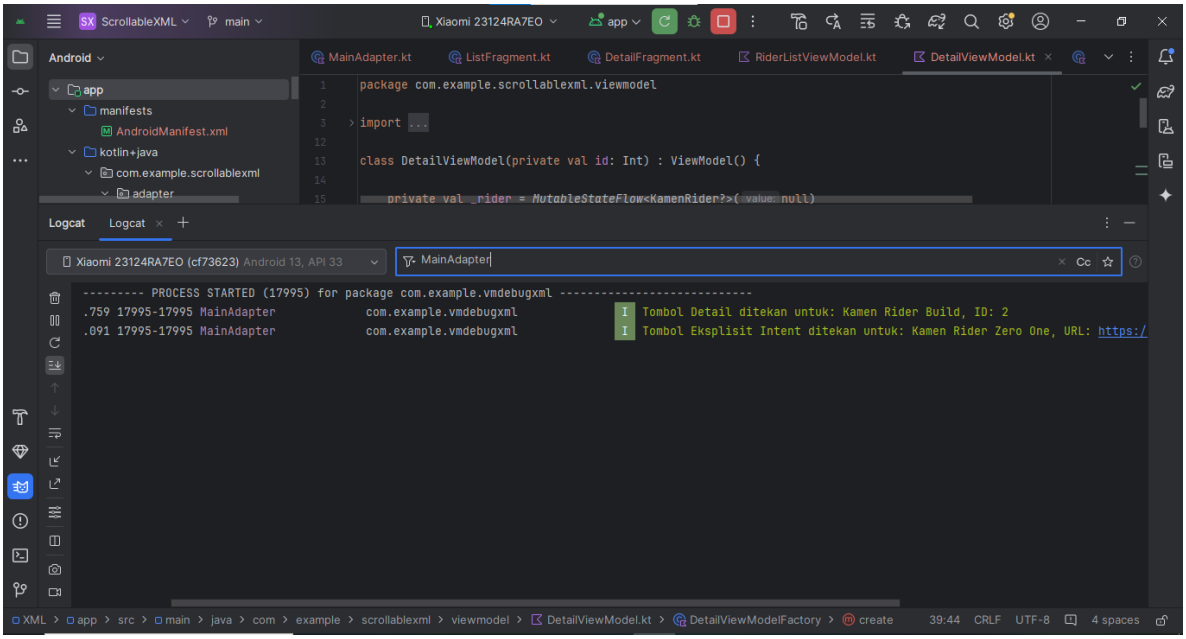
Gambar 8. Screenshot Log Data Dari List Yang Dipilih Ketika Berpindah Ke Halaman Detail Versi Compose



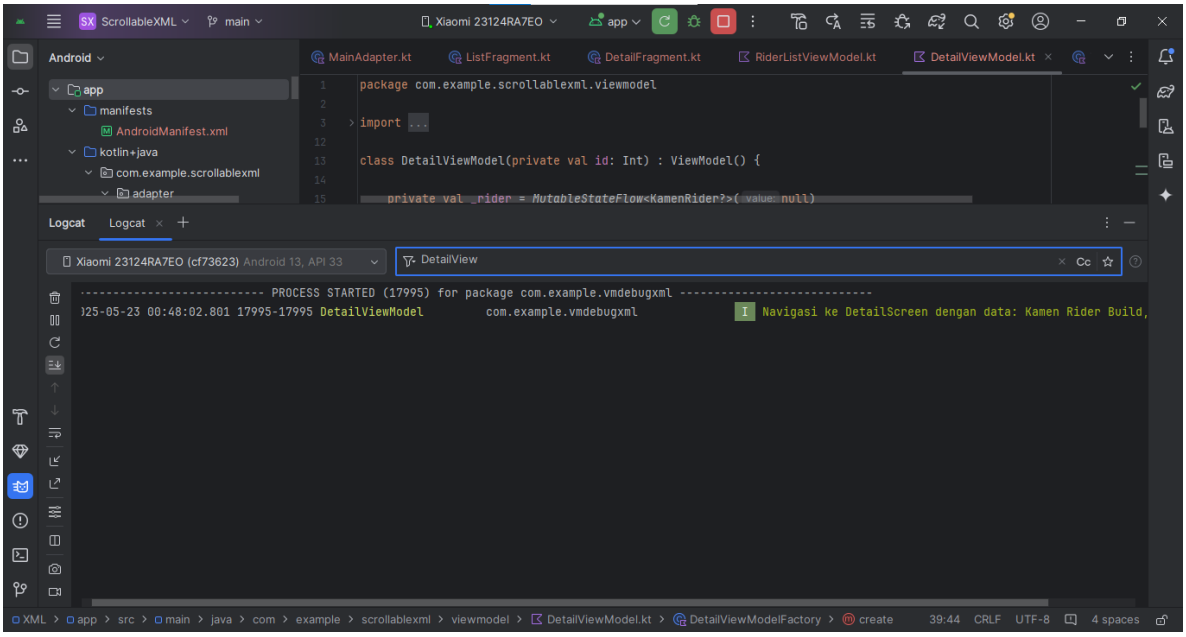
Gambar 9. Screenshot Debugging dengan Tool Debugger di Android Studio Versi Compose



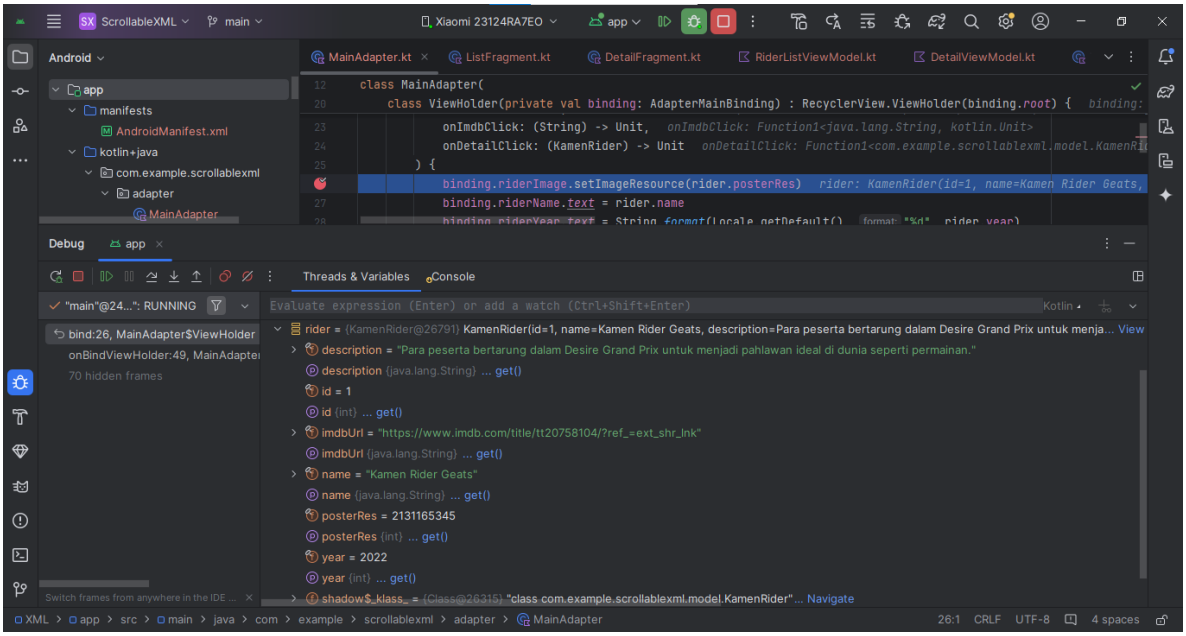
Gambar 10. Screenshot Log Saat Data Item Masuk Ke Dalam List Versi XML



Gambar 11. Screenshot Log Saat Tombol Detail Dan Tombol Explicit Intent Ditekan Versi XML



Gambar 12. Screenshot Log Data Dari List Yang Dipilih Ketika Berpindah Ke Halaman Detail Versi XML



Gambar 13. Screenshot Debugging dengan Tool Debugger di Android Studio Versi XML

C. Pembahasan

1) Versi Jetpack Compose

Card.kt

Pada Card.kt, ditampilkan dua komponen UI utama dalam bentuk fungsi Composable, yaitu RiderCardPortrait dan RiderCardLandscape, yang masing-masing menampilkan kartu informasi Kamen Rider dalam orientasi potret dan lanskap. Kedua fungsi menerima objek KamenRider, Context, dan onClick lambda sebagai parameter untuk menangani aksi pengguna. Di dalamnya, kartu dibuat menggunakan Card dari Material 3, dengan desain responsif, bayangan (elevation), serta gaya warna yang konsisten dengan tema aplikasi. Gambar poster ditampilkan dengan Image, menggunakan painterResource untuk memuat resource lokal, dipadukan dengan RoundedCornerShape dan ContentScale.FillBounds agar tampilan tetap proporsional.

Teks judul (name), tahun (year), dan deskripsi ditampilkan secara terstruktur dengan Text, serta disesuaikan ukurannya tergantung orientasi. Dua tombol berada di bagian bawah: tombol “IMDB” menggunakan explicit intent yang diarahkan ke aplikasi Chrome dan membuka URL dari properti imdbUrl, sedangkan tombol “Detail” menjalankan callback onClick, biasanya digunakan untuk navigasi ke layar detail. Kedua tombol dilengkapi dengan Timber log yang mencatat saat tombol ditekan, membantu debugging dan pelacakan interaksi pengguna.

ListScreen.kt

Pada ListScreen.kt, ditampilkan antarmuka utama yang menampilkan daftar Kamen Rider menggunakan LazyColumn dalam arsitektur Jetpack Compose. Fungsi ListScreen dideklarasikan sebagai @Composable dan memanfaatkan Scaffold untuk menyusun tata letak dengan TopAppBar yang menampilkan judul aplikasi. ViewModel diinisialisasi menggunakan RiderListViewModelFactory, di mana datanya dikumpulkan secara reaktif melalui collectAsState() dari dua StateFlow: riderList untuk daftar Kamen Rider, dan onClickRider untuk menangani interaksi pengguna.

Mode orientasi perangkat diambil dari LocalConfiguration, digunakan untuk menentukan apakah RiderCardPortrait atau RiderCardLandscape yang akan ditampilkan. Komponen LaunchedEffect memantau perubahan pada clickedRider, dan jika pengguna memilih salah satu Rider, akan dilakukan navigasi ke DetailScreen berdasarkan ID Rider melalui NavController, lalu klik akan di-reset menggunakan clearRiderClick(). Masing-masing item daftar memiliki tombol "Detail" yang ketika ditekan, akan memicu onRiderClick() pada ViewModel, sekaligus dicatat dalam log menggunakan Timber.

File ini juga menyertakan @Preview untuk menampilkan pratinjau layar dalam mode potret (ListScreenPortPreview) dengan tema ScrollableComposeTheme, memudahkan proses desain antarmuka di Android Studio. Potongan kode pratinjau untuk mode lanskap juga tersedia, namun masih dikomentari.

DetailScreen.kt

Pada DetailScreen.kt, ditampilkan layar detail dari objek KamenRider tertentu berdasarkan ID yang diterima sebagai parameter. Komponen utama DetailScreen dibangun dengan Scaffold dan menampilkan TopAppBar yang berisi tombol navigasi kembali dan judul. ViewModel diinisialisasi menggunakan DetailViewModelFactory, dan data rider diambil secara reaktif menggunakan collectAsState() dari flow yang disediakan oleh ViewModel. Jika data Rider tersedia, tampilan akan menyesuaikan berdasarkan orientasi layar—mode potret akan memanggil DetailPortrait, sedangkan mode lanskap menggunakan DetailLandscape.

Tampilan potret (DetailPortrait) menyusun gambar dan teks dalam kolom vertikal yang dapat digulir, sementara tampilan lanskap (DetailLandscape) menyusunnya dalam baris horizontal untuk memanfaatkan ruang lebar layar. Komponen RiderImage bertugas menampilkan poster Rider dengan ukuran besar dan sudut membulat. Sementara itu, RiderTextInfo menampilkan informasi teks seperti nama, tahun, dan deskripsi Rider, dengan format penataan teks yang menyesuaikan orientasi (rata tengah untuk potret dan rata kiri untuk lanskap).

Selain itu, file ini menyediakan pratinjau layar dengan anotasi @Preview, memungkinkan pengembang melihat tampilan antarmuka dalam mode potret melalui DetailScreenPortraitPreview, sedangkan pratinjau lanskap tersedia namun masih dikomentari. File ini mencerminkan pemisahan logika dan tampilan dengan baik, serta mengutamakan responsivitas terhadap orientasi perangkat untuk pengalaman pengguna yang optimal.

RiderListViewModel.kt

File RiderListViewModel.kt berfungsi sebagai bagian dari arsitektur MVVM, menangani logika data untuk menampilkan daftar Kamen Rider di UI. RiderListViewModel merupakan subclass dari ViewModel yang bertanggung jawab untuk memuat data dari KamenRiderRepository secara asinkron menggunakan coroutine di dalam viewModelScope. Data daftar Rider disimpan dalam _riderList, sebuah MutableStateFlow yang terekspos ke UI sebagai StateFlow agar bisa diobservasi secara reaktif. Selain itu, ViewModel ini juga menyimpan informasi tentang Rider yang diklik pengguna melalui _onClickRider, yang juga terekspos sebagai StateFlow.

Ketika RiderListViewModel diinisialisasi, ia langsung memanggil fungsi loadKamenRider() untuk mengisi data dari repository. Fungsi ini mengambil semua data Rider dan mengisi StateFlow dengan daftar tersebut, serta mencatat jumlah item yang dimuat menggunakan Timber untuk keperluan logging dan debugging. Fungsi onRiderClick() digunakan untuk menetapkan Rider yang diklik, yang kemudian dapat

digunakan di layar UI untuk navigasi ke layar detail. Setelah navigasi terjadi, clearRiderClick() dipanggil untuk mereset nilai klik agar tidak terjadi navigasi berulang.

Terakhir, RiderListViewModelFactory disediakan untuk membuat instance RiderListViewModel secara dinamis, khususnya ketika ViewModel memerlukan parameter tambahan (title). Kelas ini mengimplementasikan ViewModelProvider.Factory dan memeriksa apakah tipe ViewModel yang diminta cocok sebelum mengembalikan instance-nya. File ini mencerminkan praktik yang baik dalam mengelola data dan interaksi pengguna dalam aplikasi berbasis Jetpack Compose.

DetailViewModel.kt

File DetailViewModel.kt merupakan bagian penting dari arsitektur MVVM dalam aplikasi, yang berfungsi untuk menangani logika bisnis dan pengambilan data secara spesifik untuk satu entitas Kamen Rider berdasarkan id. DetailViewModel adalah subclass dari ViewModel, yang menerima parameter id saat inisialisasi. ID ini digunakan untuk mencari satu data Rider dari repository melalui fungsi loadRiderById(), yang dijalankan di dalam viewModelScope menggunakan coroutine agar proses asinkron tidak memblokir UI.

Data hasil pencarian Rider disimpan dalam properti _rider, sebuah MutableStateFlow yang terekspos sebagai StateFlow agar bisa diobservasi oleh komponen UI secara reaktif. Jika Rider ditemukan, data tersebut disimpan ke dalam _rider dan sebuah log informasi dicatat menggunakan Timber, menampilkan nama dan tahun Rider yang dipilih. Jika tidak ditemukan, pesan peringatan juga dicatat melalui Timber, memberikan visibilitas terhadap potensi kesalahan ID.

Untuk pembuatan instance DetailViewModel yang memerlukan parameter id, kelas DetailViewModelFactory disediakan. Factory ini mengimplementasikan ViewModelProvider.Factory dan memastikan bahwa hanya DetailViewModel yang bisa dibuat dengan parameter yang sesuai. File ini menunjukkan pendekatan yang terstruktur dan efisien dalam mengelola data entitas tunggal dan mendukung navigasi detail yang dinamis dalam aplikasi Android berbasis Jetpack Compose.

TimberApp.kt

File TimberApp.kt merupakan kelas turunan dari Application, yang digunakan untuk menginisialisasi konfigurasi global saat aplikasi pertama kali dijalankan. Di sini, tujuan utamanya adalah untuk mengaktifkan Timber, sebuah library logging yang lebih fleksibel dan powerful dibandingkan Log bawaan Android. Di dalam method onCreate(), terdapat pengecekan terhadap BuildConfig.DEBUG. Jika aplikasi berjalan dalam mode debug

(misalnya saat development), maka Timber.plant(Timber.DebugTree()) akan dipanggil untuk menanamkan DebugTree, yaitu implementasi default Timber yang mencetak log ke Logcat. Dengan cara ini, developer dapat menggunakan Timber.tag("TAG").d("Pesan") atau log lainnya tanpa perlu menulis tag dan level log secara manual. Konfigurasi ini memastikan bahwa logging hanya aktif saat debugging dan tidak ikut ditanamkan ke build produksi (release), sehingga menjaga performa dan keamanan data saat rilis.

2) Versi XML

MainAdapter.kt

Pada MainAdapter.kt, didefinisikan sebuah adapter untuk RecyclerView yang bertugas menampilkan daftar objek KamenRider menggunakan ViewBinding dari layout AdapterMainBinding. Kelas MainAdapter menerima tiga parameter: daftar awal initialList, lambda onImdbClick yang dipicu ketika tombol IMDb ditekan, dan onDetailClick yang dipicu ketika tombol detail ditekan. Adapter menyimpan daftar KamenRider dalam riderList, yang bisa diperbarui melalui fungsi updateData().

Di dalam kelas ViewHolder, fungsi bind() akan mengatur setiap item UI berdasarkan data KamenRider yang diberikan, seperti menampilkan gambar, nama, tahun, dan deskripsi. Dua tombol diatur dengan listener: satu untuk membuka URL IMDb menggunakan onImdbClick, dan satu lagi untuk menavigasi ke detail rider dengan onDetailClick. Setiap aksi tombol juga dicatat menggunakan Timber untuk tujuan debugging. Adapter ini mengimplementasikan metode standar RecyclerView.Adapter: onCreateViewHolder, onBindViewHolder, dan getItemCount, serta menambahkan metode updateData() yang memperbarui seluruh daftar dan memanggil notifyDataSetChanged() untuk me-refresh tampilan.

ListFragment.kt

Pada ListFragment.kt, didefinisikan sebuah fragment yang bertugas menampilkan daftar Kamen Rider menggunakan RecyclerView dan mengatur navigasi serta aksi pengguna. Fragment ini menggunakan ViewBinding melalui FragmentListBinding untuk mengakses elemen UI dengan aman dan efisien. ViewModel RiderListViewModel diinisialisasi menggunakan delegasi by viewModels dengan sebuah factory yang menyuplai judul list.

Di dalam onCreateView(), adapter MainAdapter dibuat dengan dua callback: satu untuk membuka link IMDb melalui Intent eksplisit menggunakan aplikasi Chrome, dan satu lagi untuk memicu navigasi ke layar detail dengan memanggil

viewModel.onRiderClick(). Adapter kemudian dihubungkan ke RecyclerView melalui binding.recyclerView.adapter.

Fungsi observeViewModel() bertugas untuk mengamati dua aliran StateFlow dari ViewModel. Pertama, riderList dipantau agar daftar dalam adapter diperbarui saat data berubah. Kedua, onClickRider diamati untuk menangani klik pada item. Ketika rider dipilih, fragment akan menavigasi ke DetailFragment dengan membawa ID rider menggunakan Safe Args, dan kemudian memanggil clearRiderClick() untuk menghindari navigasi ulang.

Terakhir, pada onDestroyView(), binding di-set ke null untuk mencegah memory leak, sesuai praktik terbaik dalam penggunaan ViewBinding di Fragment.

DetailFragment.kt

Pada DetailFragment.kt, didefinisikan sebuah fragment yang bertugas menampilkan detail dari item Kamen Rider yang dipilih oleh pengguna. Fragment ini menggunakan ViewBinding melalui FragmentDetailBinding untuk mengakses komponen UI secara langsung dan aman. Fragment menerima argumen riderId menggunakan delegasi navArgs() dari Navigation Component, yang memungkinkan pengiriman data antar fragment dengan aman.

ViewModel DetailViewModel diinisialisasi secara lazy menggunakan ViewModelProvider dengan DetailViewModelFactory, yang menerima riderId sebagai parameter untuk memuat data rider yang sesuai dari repository.

Pada onCreateView(), binding diinisialisasi, dan topAppBar diberi listener untuk menangani navigasi kembali menggunakan onBackPressedDispatcher, yang memungkinkan pengguna kembali ke tampilan sebelumnya.

Fungsi observeRider() digunakan untuk mengamati aliran data dari rider (tipe StateFlow) di dalam ViewModel. Ketika data rider tersedia, UI akan diperbarui dengan gambar, nama, tahun, dan deskripsi dari Kamen Rider yang bersangkutan.

Terakhir, pada onDestroyView(), binding di-set ke null untuk mencegah terjadinya memory leak — sebuah praktik standar dalam siklus hidup Fragment ketika menggunakan ViewBinding.

RiderListViewModel.kt

File RiderListViewModel.kt berfungsi sebagai bagian dari arsitektur MVVM, menangani logika data untuk menampilkan daftar Kamen Rider di UI. RiderListViewModel merupakan subclass dari ViewModel yang bertanggung jawab untuk memuat data dari

KamenRiderRepository secara asinkron menggunakan coroutine di dalam viewModelScope. Data daftar Rider disimpan dalam _riderList, sebuah MutableStateFlow yang terekspos ke UI sebagai StateFlow agar bisa diobservasi secara reaktif. Selain itu, ViewModel ini juga menyimpan informasi tentang Rider yang diklik pengguna melalui _onClickRider, yang juga terekspos sebagai StateFlow.

Ketika RiderListViewModel diinisialisasi, ia langsung memanggil fungsi loadKamenRider() untuk mengisi data dari repository. Fungsi ini mengambil semua data Rider dan mengisi StateFlow dengan daftar tersebut, serta mencatat jumlah item yang dimuat menggunakan Timber untuk keperluan logging dan debugging. Fungsi onRiderClick() digunakan untuk menetapkan Rider yang diklik, yang kemudian dapat digunakan di layar UI untuk navigasi ke layar detail. Setelah navigasi terjadi, clearRiderClick() dipanggil untuk mereset nilai klik agar tidak terjadi navigasi berulang.

Terakhir, RiderListViewModelFactory disediakan untuk membuat instance RiderListViewModel secara dinamis, khususnya ketika ViewModel memerlukan parameter tambahan (title). Kelas ini mengimplementasikan ViewModelProvider.Factory dan memeriksa apakah tipe ViewModel yang diminta cocok sebelum mengembalikan instance-nya. File ini mencerminkan praktik yang baik dalam mengelola data dan interaksi pengguna dalam aplikasi berbasis Jetpack Compose.

DetailViewModel.kt

File DetailViewModel.kt merupakan bagian penting dari arsitektur MVVM dalam aplikasi, yang berfungsi untuk menangani logika bisnis dan pengambilan data secara spesifik untuk satu entitas Kamen Rider berdasarkan id. DetailViewModel adalah subclass dari ViewModel, yang menerima parameter id saat inisialisasi. ID ini digunakan untuk mencari satu data Rider dari repository melalui fungsi loadRiderById(), yang dijalankan di dalam viewModelScope menggunakan coroutine agar proses asinkron tidak memblokir UI.

Data hasil pencarian Rider disimpan dalam properti _rider, sebuah MutableStateFlow yang terekspos sebagai StateFlow agar bisa diobservasi oleh komponen UI secara reaktif. Jika Rider ditemukan, data tersebut disimpan ke dalam _rider dan sebuah log informasi dicatat menggunakan Timber, menampilkan nama dan tahun Rider yang dipilih. Jika tidak ditemukan, pesan peringatan juga dicatat melalui Timber, memberikan visibilitas terhadap potensi kesalahan ID.

Untuk pembuatan instance DetailViewModel yang memerlukan parameter id, kelas DetailViewModelFactory disediakan. Factory ini mengimplementasikan ViewModelProvider.Factory dan memastikan bahwa hanya DetailViewModel yang bisa dibuat dengan parameter yang sesuai. File ini menunjukkan pendekatan yang terstruktur

dan efisien dalam mengelola data entitas tunggal dan mendukung navigasi detail yang dinamis dalam aplikasi Android berbasis Jetpack Compose.

TimberApp.kt

File TimberApp.kt merupakan kelas turunan dari Application, yang digunakan untuk menginisialisasi konfigurasi global saat aplikasi pertama kali dijalankan. Di sini, tujuan utamanya adalah untuk mengaktifkan Timber, sebuah library logging yang lebih fleksibel dan powerful dibandingkan Log bawaan Android. Di dalam method onCreate(), terdapat pengecekan terhadap BuildConfig.DEBUG. Jika aplikasi berjalan dalam mode debug (misalnya saat development), maka Timber.plant(Timber.DebugTree()) akan dipanggil untuk menanamkan DebugTree, yaitu implementasi default Timber yang mencetak log ke Logcat. Dengan cara ini, developer dapat menggunakan Timber.tag("TAG").d("Pesan") atau log lainnya tanpa perlu menulis tag dan level log secara manual. Konfigurasi ini memastikan bahwa logging hanya aktif saat debugging dan tidak ikut ditanamkan ke build produksi (release), sehingga menjaga performa dan keamanan data saat rilis.

3) Penjelasan Debugger

Debugger adalah alat bantu dalam IDE (Integrated Development Environment) seperti Android Studio yang digunakan untuk menjalankan aplikasi secara step-by-step guna menemukan dan memperbaiki bug atau kesalahan logika pada kode. Dengan debugger, pengembang dapat menghentikan eksekusi program pada titik tertentu yang disebut breakpoint, kemudian memeriksa nilai variabel, kondisi objek, dan alur logika secara langsung saat program berjalan.

Untuk menggunakan debugger, pertama-tama pengguna harus menempatkan breakpoint pada baris kode yang ingin dianalisis, biasanya dengan mengklik di sisi kiri editor kode. Setelah itu, aplikasi dijalankan dalam debug mode, dan ketika eksekusi mencapai breakpoint, aplikasi akan berhenti sementara sehingga pengguna bisa memeriksa status aplikasi melalui jendela debug.

Fitur Step Into digunakan untuk masuk ke dalam fungsi yang sedang dipanggil agar dapat melihat bagaimana fungsi tersebut bekerja baris per baris. Step Over digunakan untuk melangkahi eksekusi fungsi tanpa masuk ke dalamnya — berguna jika pengguna yakin bahwa fungsi tersebut berjalan dengan benar dan ingin melanjutkan ke baris berikutnya di luar fungsi. Sementara itu, Step Out digunakan untuk keluar dari fungsi saat ini dan kembali ke fungsi pemanggil, berguna ketika pengguna telah selesai menelusuri bagian dalam suatu fungsi dan ingin kembali ke konteks sebelumnya. Ketiga fitur ini sangat penting untuk memahami alur eksekusi kode secara mendalam dan menyeluruh.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

Versi Jetpack Compose:
https://github.com/raihan2030/Praktikum-Pemrograman-Mobile/tree/main/Praktikum-4/Compose

Versi XML:
https://github.com/raihan2030/Praktikum-Pemrograman-Mobile/tree/main/Praktikum-4/XML

SOAL 2

Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya!

Jawab:

Kelas Application dalam arsitektur aplikasi Android merupakan komponen global yang pertama kali diinisialisasi saat aplikasi dijalankan, sebelum Activity, Service, atau BroadcastReceiver apa pun. Kelas ini berfungsi sebagai titik masuk utama dan wadah untuk mengelola konfigurasi atau inisialisasi global yang dibutuhkan sepanjang siklus hidup aplikasi.

Umumnya, Application digunakan untuk menginisialisasi dependency injection (seperti Hilt atau Dagger), konfigurasi database lokal (seperti Room), pengaturan library logging (seperti Timber), integrasi Firebase, atau crash reporting tools seperti Crashlytics. Untuk menggunakannya, pengembang membuat subclass dari Application, menambahkan logika pada metode onCreate(), dan mendeklarasikannya di AndroidManifest.xml.

Meskipun memiliki akses global, Application tidak dianjurkan untuk menyimpan state UI atau Context dari komponen seperti Activity atau Fragment, karena berisiko menyebabkan memory leak. Sebaliknya, penyimpanan state sebaiknya dikelola melalui ViewModel atau Repository sesuai prinsip arsitektur MVVM atau Clean Architecture.

Dengan peran strategisnya, kelas Application menjadi fondasi penting untuk memastikan konsistensi, efisiensi, dan skalabilitas aplikasi Android.