LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Rifky Putra Mahardika NIM. 2310817210023

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 : Rifky Putra Mahardika
NIM : 2310817210023

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Zulfa Auliya Akbar NIM. 2210817210026	Muti`a Maulida S.Kom M.T.I NIP. 198810272019032013

DAFTAR ISI

DAFTAR GAMBAR

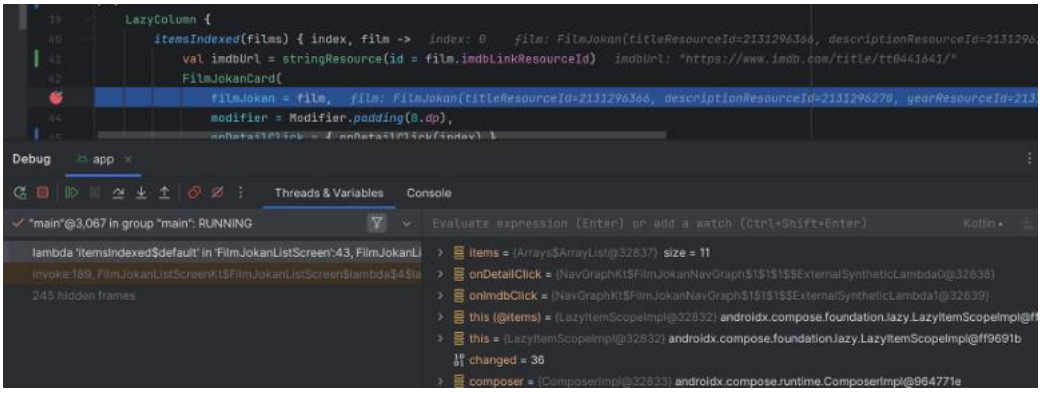
DAFTAR TABEL

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.presca.modul4
2
3  import android.content.Intent
4  import android.net.Uri
5  import android.os.Bundle
6  import androidx.activity.ComponentActivity
7  import androidx.activity.compose.setContent
8  import androidx.activity.enableEdgeToEdge
9  import androidx.activity.viewModels
10 import androidx.compose.runtime.Composable
11 import androidx.compose.ui.platform.LocalContext
12 import androidx.navigation.NavHostController
13 import androidx.navigation.compose.NavHost
14 import androidx.navigation.compose.composable
15 import androidx.navigation.compose.rememberNavController
16 import com.presca.modul4.ui.screens.MusicDetailScreen
17 import com.presca.modul4.ui.screens.MusicListScreen
18 import com.presca.modul4.ui.theme.Modul4Theme
19 import com.presca.modul4.viewmodel.MusicViewModel
20 import com.presca.modul4.viewmodel.MusicViewModelFactory
21
22 class MainActivity : ComponentActivity() {
23     private val viewModel: MusicViewModel by viewModels {
   MusicViewModelFactory() }
24
25     override fun onCreate(savedInstanceState: Bundle?) {
26         super.onCreate(savedInstanceState)
27         enableEdgeToEdge()
28         setContent {
29             Modul4Theme {
30                 val navController = rememberNavController()
31                 AppNavigation(navController, viewModel)
```

```
            }
        }
    }
}

@Composable
fun AppNavigation(navController: NavHostController,
viewModel: MusicViewModel) {
    val context = LocalContext.current
    NavHost(navController, startDestination = "music_list") {
        composable("music_list") {
            MusicListScreen(
                viewModel = viewModel,
                onMusicClick = {
                    viewModel.logDetailClick()
                    viewModel.logSelect(it)
                    navController.navigate("detail/${it.id}")
                },
                onExternalClick = {
                    viewModel.logExternalClick(it)

context.startActivity(Intent(Intent.ACTION_VIEW,
Uri.parse(it)))
                }
            )
        }
        composable("detail/{musicId}") { backStackEntry ->
            val musicId =
backStackEntry.arguments?.getString("musicId")?.toIntOrNull()
            val music = viewModel.musicList.value.find {
it.id == musicId }
            music?.let {
                MusicDetailScreen(it, navController)
            }
```

```
61          }
62      }
63  }
```

Tabel 1. Source Code Jawaban Soal 1 MainActivity.kt

2. MusicData.kt

```
1   package com.presca.modul4.data
2
3   import com.presca.modul4.models.Music
4
5   val twiceMusicList = listOf(
6       Music(
7           id = 1,
8           title = "What is Love?",
9           year = "9 April 2018",
10          imageUrl = "https://i.postimg.cc/5tpLc5DY/What-Is-
    Love-Online-Cover.webp",
11          externalUrl =
    "https://en.wikipedia.org/wiki/What_Is_Love%3F_(Twice_song)",
12          description = "Lagu \"What is Love?\" menceritakan
    tentang rasa penasaran seorang gadis muda yang belum pernah
    merasakan cinta sejati. Ia hanya mengetahui tentang cinta
    dari film, drama, dan buku, sehingga muncul keinginan kuat
    untuk benar-benar memahami dan merasakannya sendiri. Lirik
    lagu ini menggambarkan perasaan ingin tahu dan harapan akan
    datangnya cinta yang indah seperti yang sering digambarkan
    dalam kisah romantis di layar kaca."
13      ),
14      Music(
15          id = 2,
16          title = "Fancy",
17          year = "22 April 2019",
```

```
18          imageUrl =
    "https://upload.wikimedia.org/wikipedia/id/0/09/Twice_-
    _Fancy_You.png",
19          externalUrl = "https://twice.fandom.com/wiki/Fancy",
20          description = "\"FANCY\" bercerita tentang perasaan
    jatuh cinta yang begitu kuat, penuh keberanian, dan tidak
    takut mengambil risiko. Lagu ini menggambarkan momen ketika
    seseorang naksir atau menyukai orang lain dengan sangat
    intens, bahkan meski terasa berbahaya seperti duri mawar,
    perasaan itu tetap terasa manis dan menyenangkan. TWICE
    mengekspresikan keinginan untuk mengungkapkan cinta secara
    langsung, tanpa ragu, dan berani mengambil langkah pertama.
    Lirik seperti “Aku menyukaimu, aku menyukaimu, menyukaimu”
    dan “Pegang lebih kuat, ambil tanganku. Ini akan sedikit
    berbahaya, bahkan lebih berbahaya lagi, sayang” menegaskan
    keberanian dalam menghadapi cinta yang penuh tantangan."
21      ),
22      Music(
23          id = 3,
24          title = "The Feels",
25          year = "1 Oktober 2021",
26          imageUrl =
    "https://upload.wikimedia.org/wikipedia/en/5/50/Twice_-
    _The_Feels.png",
27          externalUrl =
    "https://twice.fandom.com/wiki/The_Feels",
28          description = "\"The Feels\" menggambarkan perasaan
    jatuh cinta yang kuat namun penuh rasa malu, seperti yang
    sering dialami remaja pada cinta pertama. Lagu ini bercerita
    tentang dua orang yang saling menyukai, namun keduanya masih
    malu-malu untuk mengungkapkan perasaan mereka secara
    langsung. TWICE mengekspresikan kegembiraan, rasa penasaran,
    dan degup jantung yang tak tertahankan saat jatuh cinta,
    sekaligus memberi kode kepada orang yang disukai agar lebih
```

```
    berani mengungkapkan perasaannya."
29      ),
30      Music(
31          id = 4,
32          title = "Likey",
33          year = "30 Oktober 2017",
34          imageUrl =
    "https://i.scdn.co/image/ab67616d0000b2731f21c24e81a9d0d4a30b
    e533",
35          externalUrl = "https://twice.fandom.com/wiki/Likey",
36          description = "\"Likey\" menceritakan tentang
    perasaan suka dan jatuh cinta yang membuat seseorang merasa
    bersemangat sekaligus sedikit canggung. Lagu ini
    menggambarkan bagaimana sang tokoh utama membangun
    kepercayaan diri untuk mengungkapkan perasaannya kepada orang
    yang disukai, meski masih ada rasa malu dan ragu. Selain itu,
    lagu ini juga mengangkat tema tentang bagaimana seseorang
    ingin tampil menarik dan disukai, terutama di era media
    sosial, sehingga ada tekanan untuk selalu menunjukkan sisi
    terbaiknya agar mendapatkan \"like\" atau perhatian dari
    orang lain. Melalui liriknya, TWICE menyampaikan pesan
    tentang keberanian dalam mencintai dan menjadi diri sendiri,
    tanpa harus terlalu terpengaruh oleh penilaian orang lain."
37      ),
38      Music(
39          id = 5,
40          title = "Cheer Up",
41          year = "24 April 2016",
42          imageUrl =
    "https://i.scdn.co/image/ab67616d0000b273acf4830dde5e17d356b8
    0ae8",
43          externalUrl =
    "https://en.wikipedia.org/wiki/Cheer_Up_(song)",
44          description = "Lagu ini mengisahkan tentang seorang
```

```
    perempuan yang sedang dalam tahap pendekatan dengan pria yang
    sangat posesif, di mana ia meminta sedikit ruang agar proses
    pendekatan menjadi lebih menyenangkan dan tidak menekan.
    Melalui liriknya, \"Cheer Up\" menyampaikan pesan untuk tetap
    semangat dan memberikan dukungan kepada orang yang disukai,
    meskipun ada rasa canggung dan ketidakpastian dalam hubungan
    yang sedang berkembang. Lagu ini juga menonjolkan karakter
    ceria dan energik khas TWICE, dengan melodi yang catchy dan
    koreografi yang dinamis."
45      ),
46      Music(
47          id = 6,
48          title = "TT",
49          year = "24 Oktober 2016",
50          imageUrl =
    "https://i.scdn.co/image/ab67616d0000b273387444ab2fc1f08dfe79
    15ab",
51          externalUrl =
    "https://en.wikipedia.org/wiki/TT_(song)",
52          description = "Lagu \"TT\" menggambarkan perasaan
    cinta yang membingungkan dan penuh gejolak. Liriknya
    menceritakan tentang seorang perempuan yang merasa kesal dan
    bingung terhadap perasaan cintanya yang semakin menggebu-gebu
    meskipun ia berusaha menjauh. Perasaan tersebut membuatnya
    merasa seperti \"TT\" - sebuah ekspresi emotikon menangis
    yang juga menjadi simbol khas lagu ini. Lagu ini
    mengungkapkan dilema antara ingin mendekat dan sekaligus
    merasa canggung atau takut dalam menghadapi cinta."
53      )
54  )
```

Tabel 2. Source Code Jawaban Soal 1 MusicData.kt

3. Music.kt

```
1   package com.presca.modul4.models
2
3   data class Music(
4       val id: Int,
5       val title: String,
6       val year: String,
7       val imageUrl: String,
8       val externalUrl: String,
9       val description: String
10  )
```

Tabel 3. Source Code Jawaban Soal 1 Music.kt

4. MusicDetailScreen.kt

```
1   package com.presca.modul4.ui.screens
2
3   import androidx.compose.foundation.layout.*
4   import androidx.compose.foundation.lazy.LazyColumn
5   import androidx.compose.material.icons.Icons
6   import androidx.compose.material.icons.filled.ArrowBack
7   import androidx.compose.material3.*
8   import androidx.compose.runtime.Composable
9   import androidx.compose.ui.Modifier
10  import androidx.compose.ui.draw.clip
11  import androidx.compose.ui.unit.dp
12  import androidx.navigation.NavController
13  import
    com.bumptech.glide.integration.compose.ExperimentalGlideCom
    poseApi
14  import com.bumptech.glide.integration.compose.GlideImage
15  import com.presca.modul4.models.Music
16  import androidx.compose.foundation.shape.RoundedCornerShape
17
```

```
@OptIn(ExperimentalGlideComposeApi::class,
ExperimentalMaterial3Api::class)
@Composable
fun MusicDetailScreen(music: Music, navController:
NavController) {
    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(music.title, color =
MaterialTheme.colorScheme.onPrimary) },
                navigationIcon = {
                    IconButton(onClick = {
navController.popBackStack() }) {
                        Icon(Icons.Filled.ArrowBack,
contentDescription = "Back")
                    }
                },
                colors =
TopAppBarDefaults.topAppBarColors(containerColor =
MaterialTheme.colorScheme.surface)
            )
        }
    ) { padding ->
        LazyColumn(
            modifier =
Modifier.padding(padding).padding(16.dp),
            verticalArrangement =
Arrangement.spacedBy(12.dp)
        ) {
            item {
                GlideImage(
                    model = music.imageUrl,
                    contentDescription = music.title,
                    modifier =
```

```
    Modifier.fillMaxWidth().height(400.dp).clip(RoundedCornerSh
    ape(16.dp))
43                  )
44              }
45              item {
46                  Text("Judul: ${music.title}", style =
    MaterialTheme.typography.titleLarge)
47              }
48              item {
49                  Text("Tanggal Rilis: ${music.year}")
50              }
51              item {
52                  Text("Tentang Lagu Ini:", style =
    MaterialTheme.typography.titleMedium)
53              }
54              item {
55                  Text(music.description, style =
    MaterialTheme.typography.bodyMedium)
56              }
57          }
58      }
59  }
60
```

Tabel 4. Source Code Jawaban Soal 1 MusicDetailScreen.kt

5. MusicListScreen.kt

```
1   package com.presca.modul4.ui.screens
2
3   import androidx.compose.foundation.layout.*
4   import androidx.compose.foundation.lazy.LazyColumn
5   import androidx.compose.foundation.lazy.items
6   import androidx.compose.foundation.shape.RoundedCornerShape
7   import androidx.compose.material3.*
```

```
8  import androidx.compose.runtime.Composable
9  import androidx.compose.runtime.collectAsState
10 import androidx.compose.ui.Modifier
11 import androidx.compose.ui.draw.clip
12 import androidx.compose.ui.text.font.FontWeight
13 import androidx.compose.ui.unit.dp
14 import androidx.compose.ui.unit.sp
15 import
   com.bumptech.glide.integration.compose.ExperimentalGlideCom
   poseApi
16 import com.bumptech.glide.integration.compose.GlideImage
17 import com.presca.modul4.models.Music
18 import com.presca.modul4.viewmodel.MusicViewModel
19
20 @OptIn(ExperimentalGlideComposeApi::class,
   ExperimentalMaterial3Api::class)
21 @Composable
22 fun MusicListScreen(
23     viewModel: MusicViewModel,
24     onMusicClick: (Music) -> Unit,
25     onExternalClick: (String) -> Unit
26 ) {
27     val musicList =
   viewModel.musicList.collectAsState().value
28
29     Scaffold(
30         topBar = { TopAppBar(title = { Text("List of
   Twice's Best Songs") }) }
31     ) { padding ->
32         LazyColumn(
33             contentPadding = padding,
34             modifier = Modifier.fillMaxSize().padding(8.dp)
35         ) {
36             items(musicList) { music ->
```

```
37                 Card(
38                     modifier =
    Modifier.fillMaxWidth().padding(vertical = 10.dp),
39                     shape = RoundedCornerShape(16.dp),
40                     elevation =
    CardDefaults.cardElevation(6.dp)
41                 ) {
42                     Row(modifier = Modifier.padding(12.dp))
    {
43                         GlideImage(
44                             model = music.imageUrl,
45                             contentDescription =
    music.title,
46                             modifier =
    Modifier.size(120.dp).clip(RoundedCornerShape(12.dp))
47                         )
48                         Spacer(Modifier.width(16.dp))
49                         Column(modifier =
    Modifier.weight(1f)) {
50                             Text(music.title, fontSize =
    18.sp, style = MaterialTheme.typography.titleMedium)
51                             Text("Tanggal Rilis:
    ${music.year}", fontSize = 14.sp)
52                             Spacer(Modifier.height(8.dp))
53                             Text("Tentang Lagu Ini:",
    fontSize = 12.sp, fontWeight = FontWeight.Bold)
54                             Text(music.description,
    fontSize = 12.sp, maxLines = 3)
55                             Spacer(Modifier.height(12.dp))
56                             Row(
57                                 modifier =
    Modifier.fillMaxWidth(),
58                                 horizontalArrangement =
    Arrangement.spacedBy(8.dp)
```

```
59                                ) {
60                                    Button(onClick = {
    onExternalClick(music.externalUrl) }, modifier =
    Modifier.weight(1f)) {
61                                        Text("Info")
62                                    }
63                                    Button(onClick = {
    onMusicClick(music) }, modifier = Modifier.weight(1f)) {
64                                        Text("Detail")
65                                    }
66                                }
67                            }
68                        }
69                    }
70                }
71            }
72        }
73    }
```

Tabel 5. Source Code Jawaban Soal 1 MusicListScreen.kt

6. MusicViewModel.kt

```
1   package com.presca.modul4.viewmodel
2
3   import android.util.Log
4   import androidx.lifecycle.ViewModel
5   import com.presca.modul4.data.twiceMusicList
6   import com.presca.modul4.models.Music
7   import kotlinx.coroutines.flow.MutableStateFlow
8   import kotlinx.coroutines.flow.StateFlow
9
10  class MusicViewModel : ViewModel() {
11      private val _musicList =
```

```
    MutableStateFlow(twiceMusicList)
12      val musicList: StateFlow<List<Music>> = _musicList
13
14      fun logSelect(music: Music) {
15          Log.d("MusicViewModel", "Musik: ${music.title}")
16      }
17
18      fun logDetailClick() {
19          Log.d("MusicViewModel", "Tombol detail ditekan")
20      }
21
22      fun logExternalClick(url: String) {
23          Log.d("MusicViewModel", "link website informasi
    musik ditekan: $url")
24      }
25
26      init {
27          Log.d("MusicViewModel", " Musik list berisi
    ${_musicList.value.size} items")
28      }
29  }
30
```

Tabel 6. Source Code Jawaban Soal 1 MusicViewModel.kt

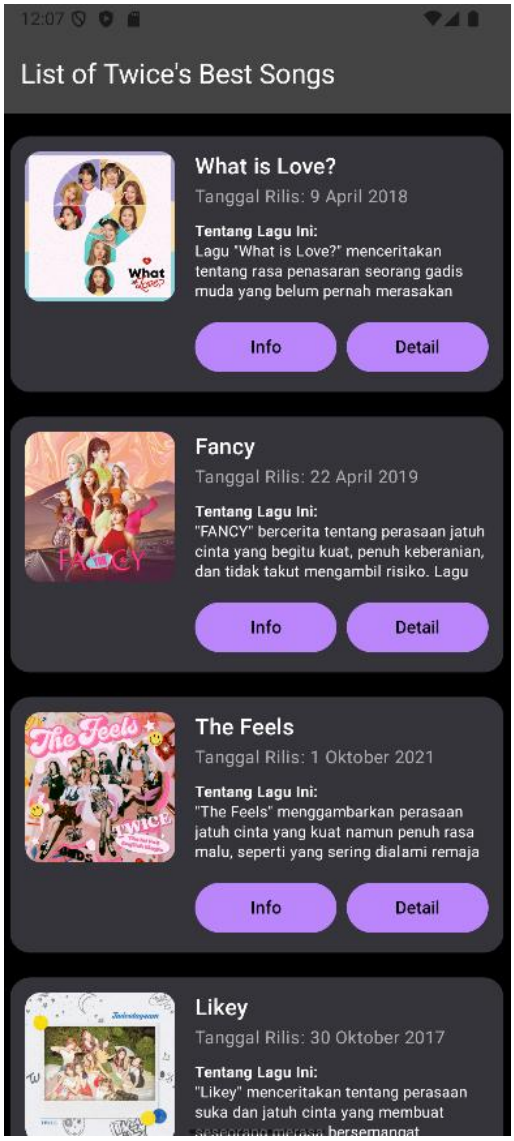
1. MusicViewModelFactory.kt

```
1   package com.presca.modul4.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5
6   class MusicViewModelFactory : ViewModelProvider.Factory {
7       override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
```

```
8           if
        (modelClass.isAssignableFrom(MusicViewModel::class.java)) {
9               return MusicViewModel() as T
10          }
11          throw IllegalArgumentException("Unknown ViewModel
        class")
12      }
13  }
```

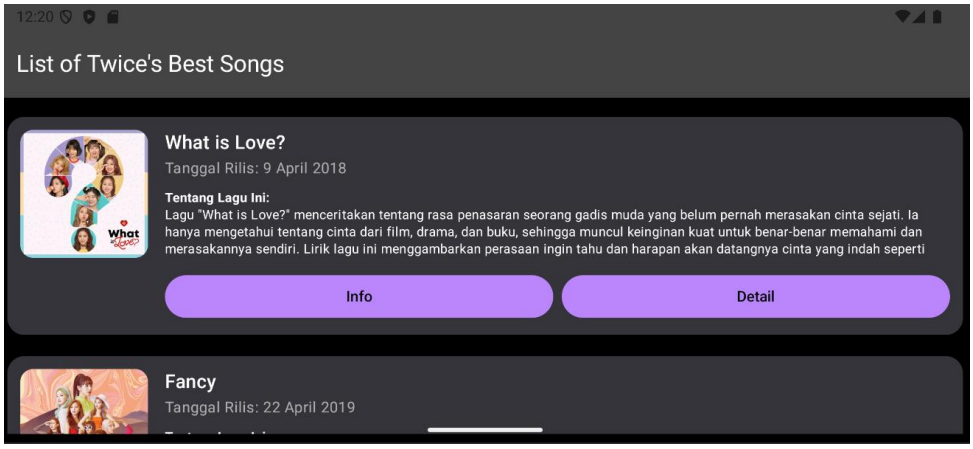
Tabel 7. Source Code Jawaban Soal 1 MusicViewModelFactory.kt

B. Output Program

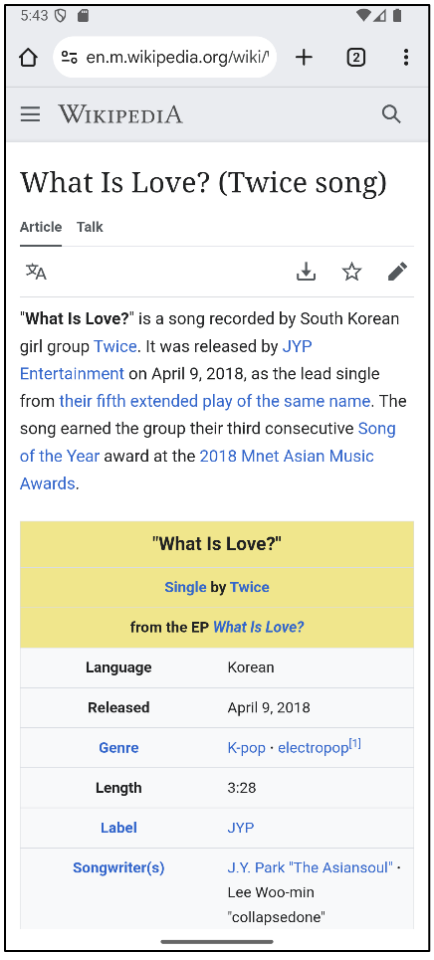


Gambar 2. Screenshot Hasil Jawaban Soal 1 Tampilan List

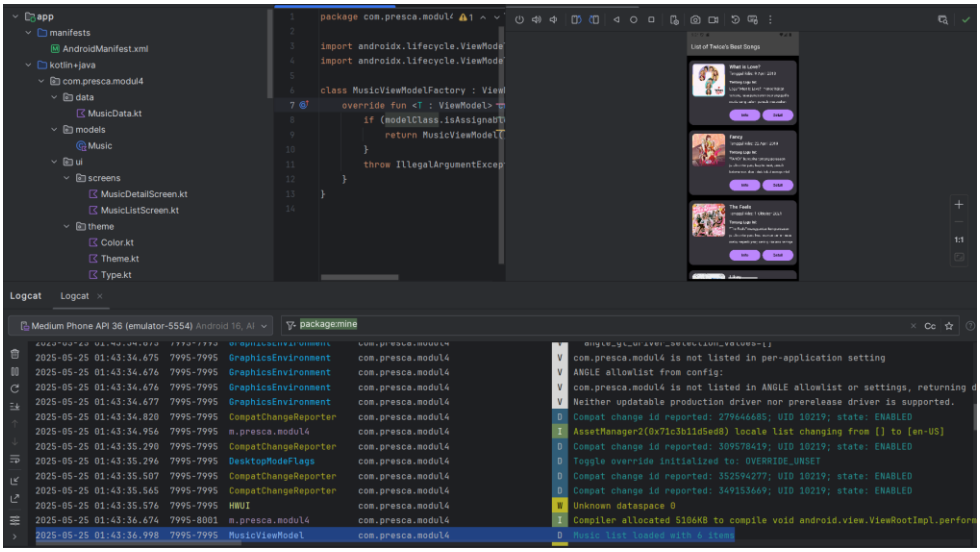
Gambar 3. Screenshot Hasil Jawaban Soal 1 Tampilan Detail



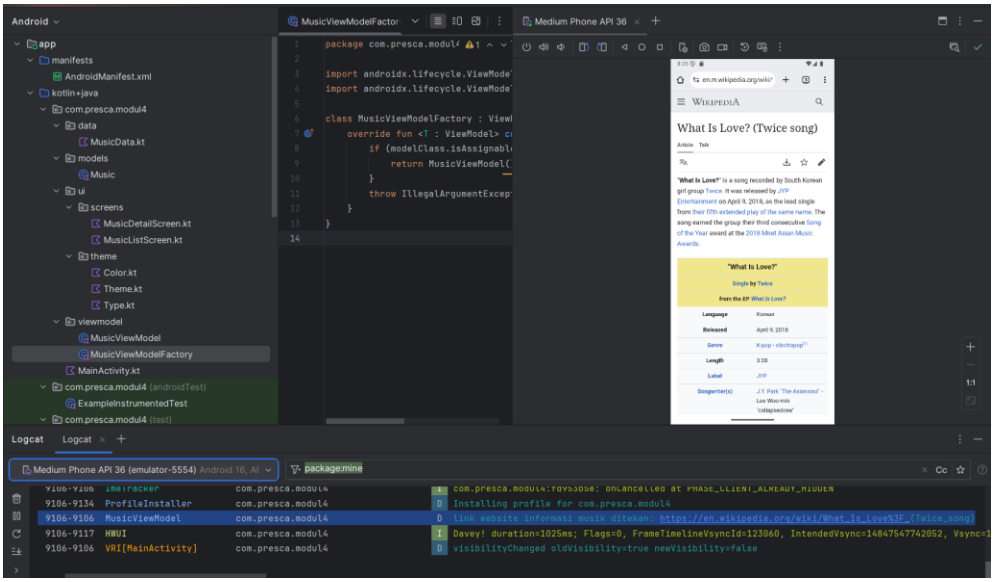
Gambar 4. Screenshot Hasil Jawaban Soal 1 Mode Landscape



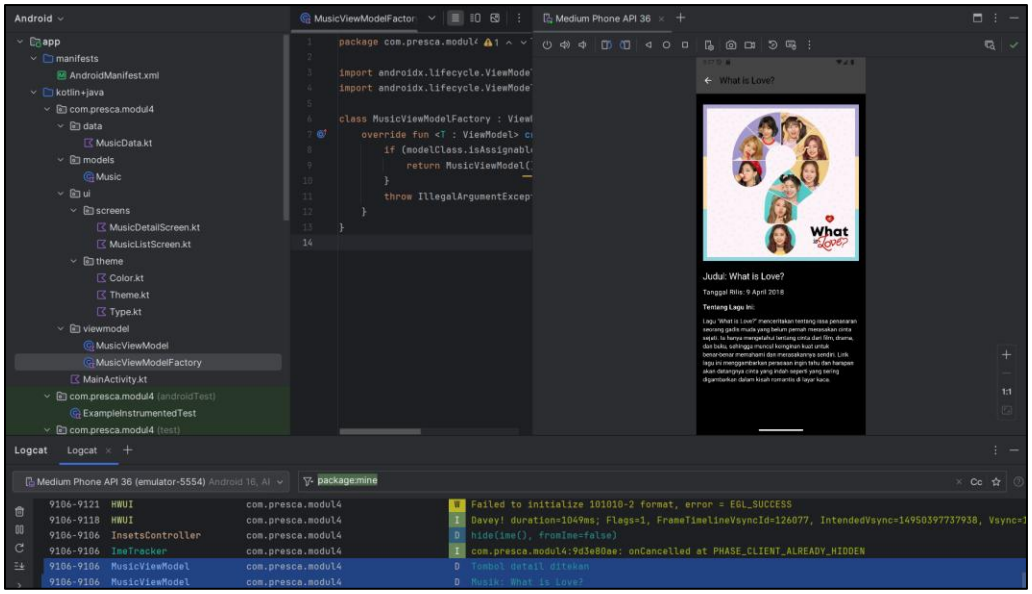
Gambar 5. Screenshot Hasil Jawaban Soal 1 Hasil Tombol Info



Gambar 6. Screenshot Hasil Jawaban Soal 1 Pada Saat Memuat List Musik



Gambar 7. Screenshot Hasil Jawaban Soal 1 Pada Saat Memuat Info Eksternal Musik



Gambar 8. Screenshot Hasil Jawaban Soal 1 Pada Saat Memuat Info Detail Musik

C. Pembahasan

1) Berikut adalah penjelasan untuk soal nomor 1:

1. MainActivity.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `com.presca.modul4`.
- Pada baris [3] hingga [20], `import` adalah perintah yang digunakan untuk mengimpor kelas, fungsi, atau objek dari package lain tanpa harus menyebutkan path lengkapnya.
- Pada baris [22], `class MainActivity : ComponentActivity()` ini digunakan sebagai titik awal aplikasi yang akan mengatur tampilan aplikasi.
- Pada baris [23], `private val viewModel: MusicViewModel by viewModels { MusicViewModelFactory() }` ini menginisiasikan `MusicViewModel` menggunakan delegasi `by viewModels` dengan `MusicViewModelFactory`. `ViewModel` digunakan untuk logika bisnis dan data.
- Pada baris [25], `onCreate()` merupakan siklus hidup (lifecycle) dari sebuah activity, yang dimana fungsi ini dipanggil pertama kali saat Activity dibuat.

- Pada baris [26], `super.onCreate(savedInstanceState)` ini memanggil `onCreate()` agar proses inisialisasi standar tetap berjalan.
- Pada baris [27], `enableEdgeToEdge()` digunakan untuk mengaktifkan rendering hingga ke edge layar.
- Pada baris [28], `setContent{…}` digunakan untuk menetapkan tampilan UI dari aplikasi, dan diterapkan tampilan dari `Modul4Theme`.
- Pada baris [29], diterapkan tema khusus aplikasi dari `Modul4Theme`.
- Pada baris [30], `rememberNavController()` digunakan untuk membuat controller navigasi yang mengatur perpindahan antar-screen.
- Pada baris [28], `AppNavigation(navController, viewModel)` berisi logika navigasi aplikasi dan menggunakan viewModel untuk menyimpan logika terkait dengan navigasi.
- Pada baris [38], `AppNavigation` adalah fungsi compose yang digunakan untuk menangani semua screen dan navigasi antar-screen.
- Pada baris [39], `LocalContext.current` ini berisi akses Context Android dari dalam Compose.
- Pada baris [40], `NavHost` merupakan container dari navigasi. `startDestination` ini digunakan untuk mengatur screen awal adalah "music_list".
- Pada baris [41], didefinisikan “music_list” untuk menampilkan daftar musik.
- Pada baris [42], `MusicListScreen(...)` merupakan fungsi `@Composable` yang bertanggung jawab menampilkan daftar musik.
- Pada baris [43], `viewModel = viewModel` digunakan untuk mendapatkan data musik dan menjalankan fungsi logika.
- Pada baris [44], fungsi `onMusicClick` ini akan terpanggil ketika pengguna menekan button “Detail” dan akan menampilkan detail musik.
- Pada baris [45], Dipanggil fungsi `logDetailClick()` dari `viewModel`.
- Pada baris [46], memanggil `logSelect(it)` untuk mencatat lagu apa yang diklik.

- Pada baris [47], `navController.navigate("detail/${it.id}")` ini merupakan navigasi ke halaman dari detail lagu berdasarkan ID.
- Pada baris [49] hingga [51], fungsi `onExternalClick` ini akan terpanggil ketika pengguna menekan button “Info” dan halaman akan berpindah membuka URL di browser.
- Pada baris [55], `composable("detail/{musicId}")` ini digunakan untuk mendefinisikan route untuk tampilan detail musik.
- Pada baris [56], `val musicId = backStackEntry.arguments?.getString("musicId")?.toIntOrNull()` ini akan mengambil nilai parameter `musicId` dari rute yang akan dikirim pada saat navigasi.
- Pada baris [57], `val music = viewModel.musicList.value.find { it.id == musicId }` digunakan untuk mencaari objek di dalam `musicList` dari `viewModel` berdasarkan ID.
- Pada baris [58] dan [59], Jika music ditemukan (tidak null), maka tampilkan `MusicDetailScreen`. Informasi ini ditampilkan sesuai dengan ID.

2. MusicData.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `com.presca.modul4.data`
- Pada baris [3], `import com.presca.modul4.models.Music` digunakan untuk mengimpor kelas `Music` yang memiliki data class yang berisi struktur dari lagu.
- Pada baris [5], `twiceMusicList` ini digunakan untuk membuat list dari musik yang nantinya akan ditampilkan pada `MusicListScreen`.
- Pada baris [7] hingga [12], diisi dari list musik pertama dengan format `id` (sesuai urutan lagu), `title` (judul lagu), `year` (tanggal rilis), `imageUrl` (gambar album), `externalUrl` (link yang dapat dibuka untuk informasi lebih jelas di browser, dan `description` (untuk penjelasan lebih detailnya).

- Pada baris [14] hingga [54], sesuaikan struktur sebelumnya untuk mengisi sisa lagu yang ingin dibuat ke dalam list.

3. Music.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `com.presca.modul4.models`
- Pada baris [3], didefinisikan data dengan nama `Music`.
- Pada baris [4], didefinisikan juga `id` dalam bentuk integer.
- Pada baris [5] hingga [9], didefinisikan `title`, `year`, `imageUrl`, `externalUrl`, dan `description` dalam bentuk string.

4. MusicDetailScreen.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `com.presca.modul4.ui.screens`
- Pada baris [3] hingga [16], `import` adalah perintah yang digunakan untuk mengimpor kelas, fungsi, atau objek dari package lain tanpa harus menyebutkan path lengkapnya.
- Pada baris [18], `@OptIn(…)` ini digunakan untuk mengindikasikan jika kita menggunakan API eksperimental (Glide Compose dan Material3).
- Pada baris [20], `MusicDetailScreen` dibuat dengan parameter `music` untuk data musik, dan `navController` untuk navigasi antar screen.
- Pada baris [21], `Scaffold` digunakan untuk membuat struktur layout utama.
- Pada baris [22], `topBar` digunakan untuk mendefinisikan bar di bagian atas.
- Pada baris [24], `title` nantinya pada bagian ini digunakan untuk menampilkan judul lagu pada bar atas, dan `text` nantinya digunakan untuk menampilkan teks pada judul.
- Pada baris [25], bagian `navigationIcon` ini digunakan untuk mengatur tombol navigasi (seperti tombol kembali).

- Pada baris [26], `onClick` pada bagian ini, `navController.popBackStack()` digunakan untuk kembali ke screen sebelumnya dalam stack navigasi
- Pada baris [38], pada bagian `Icon(Icons.Filled.ArrowBack, contentDescription = "Back")` ini untuk menampilkan ikon panah kembali sebagai ikon dari tombol kembali, dengan deskripsi “Back”.
- Pada baris [30], `colors` pada bagian ini digunakan untuk mengatur warna pada top bar. `containerColor` digunakan untuk mengatur warna latar belakang pada scaffold.
- Pada baris [34], `LazyColumn` merupakan sebuah composable yang digunakan untuk menampilkan daftar item yang dapat digulir secara vertikal.
- Pada baris [35], `modifier` pada bagian ini digunakan untuk mengatur layout dari `LazyColumn`.
- Pada baris [36], `Arrangement.spacedBy(12.dp)` ini digunakan untuk memberi jarak 12dp secara vertikal antar item.
- Pada baris [38], `item` pada bagian ini digunakan untuk mengatur ukuran dan bentuk dari cover musik.
- Pada baris [39], `GlideImage` ini merupakan komponen yang digunakan untuk menampilkan gambar dari URL menggunakan library Glide, serta mengatur ukuran dan bentuk dari gambar tersebut.
- Pada baris [45] hingga [55], `item` pada bagian tersebut mengatur kembali untuk tampilan teks dari judul, tanggal rilis, tentang lagu ini, dan teks deskripsi.

5. MusicListScreen.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `com.presca.modul4.ui.screens`
- Pada baris [3] hingga [18], `import` adalah perintah yang digunakan untuk mengimpor kelas, fungsi, atau objek dari package lain tanpa harus menyebutkan path lengkapnya.

- Pada baris [20], `@OptIn(…)` ini digunakan untuk mengindikasikan jika kita menggunakan API eksperimental (Glide Compose dan Material3).
- Pada baris [22] hingga [25], `MusicListScreen` dibuat dengan parameter callback `onMusicClick` dan `onExternalClick` untuk interaksi pengguna. `viewModel` digunakan untuk mengambil data lagu.
- Pada baris [27], val musicList = viewModel.musicList.collectAsState().value digunakan untuk mengumpulkan data `musicList` dari `MusicViewModel` menggunakan `State`.
- Pada baris [29], `Scaffold` digunakan untuk membuat struktur layout utama.
- Pada baris [30], `topBar` digunakan untuk mendefinisikan bar di bagian atas. `title` nantinya pada bagian ini digunakan untuk menampilkan judul lagu pada bar atas, dan `text` nantinya digunakan untuk menampilkan teks pada bar atas.
- Pada baris [32], `LazyColumn` merupakan sebuah composable yang digunakan untuk menampilkan daftar item yang dapat digulir secara vertikal.
- Pada baris [34], `modifier` pada bagian ini digunakan untuk mengatur layout dari `LazyColumn`.
- Pada baris [36], `items(musicList)` ini merupakan loop melalui daftar musik untuk membuat item.
- Pada baris [37], `Card()` digunakan sebagai komponen Material Design yang digunakan untuk menampilkan konten terkelompok.
- Pada baris [38], `fillMaxWidth()` ini digunakan agar lebar mengisi parent.
- Pada baris [39], `RoundedCornerShape(16.dp)` ini digunakan untuk mengatur sudut kelengkungan dengan radius 16dp
- Pada baris [40], `cardElevation(6.dp)` digunakan untuk mengatur efek bayangan dengan elevasi 6dp.
- Pada baris [42], `Row` digunakan untuk layout horizontal untuk gambar dan juga teks informasi.
- Pada baris [43], `GlideImage` digunakan untuk menampilkan gambar cover dari URL, serta mengatur ukuran dan bentuk dari gambar tersebut.

- Pada baris [48], `Spacer` digunakan untuk memberi jarak horizontal antara gambar dan juga teks, yang dimana pada baris ini diberi jarak 16dp.
- Pada baris [49], `Column` digunakan untuk menyusun teks dan tombol secara vertikal.
- Pada baris [50], `Text` pada bagian ini akan mengatur tampilan dari judul.
- Pada baris [51] hingga [55], `Text` pada bagian tersebut mengatur kembali untuk tampilan teks dari tanggal rilis, tentang lagu ini, dan teks deskripsi.
- Pada baris [56], `Row` pada bagian ini digunakan untuk menyusun dan mengatur dua tombol (tombol info dan detail) secara horizontal.
- Pada baris [60] hingga [64], bagian ini digunakan untuk mengatur tampilan dari `button` "Info" dan "Detail" serta fungsi dari button tersebut pada saat diklik.

6. MusicViewModel.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `package com.presca.modul4.viewmodel`
- Pada baris [3] hingga [8], `import` adalah perintah yang digunakan untuk mengimpor kelas, fungsi, atau objek dari package lain tanpa harus menyebutkan path lengkapnya.
- Pada baris [10], `class MusicViewModel : ViewModel()` ini mendefinisikan `MusicViewModel` yang mewarisi `ViewModel` dan ViewModel digunakan untuk menyimpan dan menglola data lagu yang akan ditampilkan.
- Pada baris [11], `_musicList` ini digunakan untuk menyimpan daftar music dari `twiceMusicList` dengan menggunakan `MutableStateFlow`. `MutableStateFlow()` digunakan untuk membungkus suatu data yang dapat berubah dan diamati (observable).
- Pada baris [12], bagian `musicList` ini akan diekspos ke luar sebagai `StateFlow` atau hanya baca.

- Pada baris [14] dan [15], `logSelect(music: Music)` ini digunakan untuk mencatat ketika pengguna memiliki music tertentu dan menampilkan judul dari music yang diklik atau dipilih.
- Pada baris [18] dan [19], logDetailClick() ini digunakan untuk mencatat jika tombol Detail ditekan dan mendeteksi interaksi pengguna ketika tombol Detail ini ditekan.
- Pada baris [22] dan [23], logExternalClick(url: String) ini digunakan untuk mencatat jika tombol Info ditekan dan mendeteksi interaksi pengguna ketika tombol Info ini ditekan.
- Pada baris [26] dan [27], blok `init` ini digunakan ketika `ViewModel` pertama kali dibuat dan mencatat jumlah item musik yang dimuat.

7. MusicViewModelFactory.kt:

- Pada baris [1], dideklarasikan nama package file Kotlin yang dikelompokkan file ini ke dalam package `package com.presca.modul4.viewmodel`
- Pada baris [3] dan [4], `import` adalah perintah yang digunakan untuk mengimpor kelas, fungsi, atau objek dari package lain tanpa harus menyebutkan path lengkapnya.
- Pada baris [6], `class MusicViewModelFactory : ViewModelProvider.Factory` digunakan untuk mendefinisikan `MusicViewModelFactory` yang menggunakan intervace `ViewModelProvider.Factory` yang bertujuan untuk membuat `ViewModel` secara custom.
- Pada baris [7], `override fun <T : ViewModel> create(modelClass: Class<T>): T` merupakan implementasi dari `ViewModelProvider.Factory` yang digunakan untuk membuat dan mengembalikan instance dari `ViewModel`.

- Pada baris [8] dan [9], digunakan untuk mengecek apakah `modelClass` adalah `MusicViewModel`, jika benar maka buat instance baru dari `MusicViewModel`.
- Pada baris [11], `throw IllegalArgumentException("Unknown ViewModel class")` digunakan jika class yang diminta bukan `MusicViewModel`, maka dilempar exception pada baris ini.

Jawaban poin e:

- **Pengertian debugger**

  Debugger merupakan alat bantu yang digunakan untuk membantu menemukan, menganalisis, dan juga memperbaiki kesalahan (bug) pada kode program.
- **Fungsi debugger**
  - Digunakan untuk menemukan bug.
  - Memeriksa dan memahami alur dari eksekusi program.
  - Memeriksa perubahan nilai variabel saat runtime.
  - Menguji bagian kode tertentu, apakah dapat berjalan sesuai ekspetasi.
- **Cara menggunakan debugger**
  1. Memasang Breakpoint dengan cara klik pada bagian sisi kiri nomor baris kode, hingga warna merah muncul di baris yang ingin diperiksa.
  2. Jalankan aplikasi dalam mode Debug.
  3. Aplikasi akan berhenti sementara dan pengguna dapat melakukan debugging.
- **Penjelasan fitur Step Into, Step Over, dan Step Out**
  - Step Into: digunakan untuk masuk ke dalam fungsi atau metode yang sedang dipanggil tadi.
  - Step Over: digunakan untuk melanjutkan ke baris berikutnya tanpa masuk ke dalam fungsi yang dipanggil.
  - Step Out: Keluar dari fungsi pada saat ini dan kembali ke pemanggilnya.

2) Application Class merupakan kelas dasar Android yang mewakili aplikasi dan lifecycle secara keseluruhan dan dibuat sebelum komponen lain (seperti Activity dan Service). Kelas ini hanya dibuat sekali selama aplikasi berjalan.

- **Fungsi Application Class**
  - Digunakan untuk menginisialisasi global yang dimana memuat layanan yang harus tersedia di seluruh aplikasi, seperti Room, Firebase, dan lainnya,
  - Penyimpanan state aplikasi, yang dimana menyimpan data yang perlu untuk diakses di seluruh Activity, contohnya seperti ViewModel global.
  - Digunakan untuk memanajemen lifecycle aplikasi, yang dimana Application class memberikan developer kendali penuh terhadap perilaku aplikasi pada berbagai tahap keberadaannya. Berbeda dengan lifecycle komponen individual seperti Activity atau Fragment yang hanya mengelola bagian tertentu dari UI, Application class ini memiliki kemampuan untuk memantau dan merespons perubahan status aplikasi secara global.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

https://github.com/Prescaa/Kuliah/tree/master/Praktikum%20Mobile/MODUL%204