LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Allano Lintang Ertantora **NIM. 2310817210004**

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 : Allano Lintang Ertantora
NIM : 2310817210004

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Zulfa Auliya Akbar NIM. 2210817210026	Muti`a Maulida S.Kom M.T.I NIP. 19881027 201903 20 13

DAFTAR ISI

DAFTAR GAMBAR

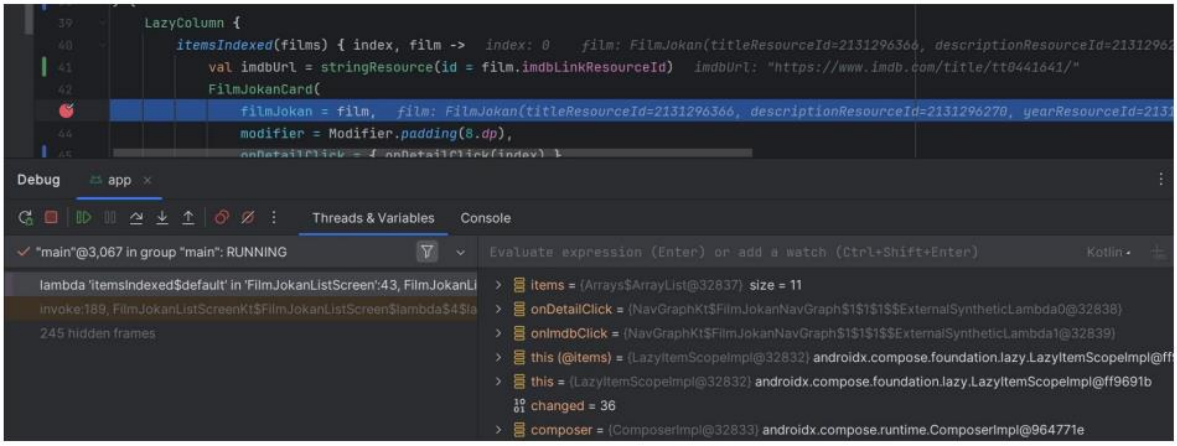
DAFTAR TABEL

SOAL 1

Soal Praktikum:

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

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



A. Source Code

1. MainActivity.kt

Tabel 1. Source Code Jawaban Soal 1

```
1   package com.allano.nongki
2
3   import android.os.Bundle
4   import androidx.appcompat.app.AppCompatActivity
5   import androidx.activity.enableEdgeToEdge
6   import com.allano.nongki.databinding.ActivityMainBinding
7
8   class MainActivity : AppCompatActivity() {
9       private lateinit var binding: ActivityMainBinding
10
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          enableEdgeToEdge()
14          binding =
    ActivityMainBinding.inflate(layoutInflater)
15          setContentView(binding.root)
16      }
17  }
```

2. HomeFragment.kt

Tabel 2. Source Code Jawaban Soal 1

```
1   package com.allano.nongki
2
3   import android.os.Bundle
4   import android.util.Log
```

```
5  import android.view.LayoutInflater
6  import android.view.View
7  import android.view.ViewGroup
8  import androidx.fragment.app.Fragment
9  import androidx.fragment.app.viewModels
10 import androidx.lifecycle.lifecycleScope
11 import androidx.navigation.fragment.findNavController
12 import androidx.recyclerview.widget.LinearLayoutManager
13 import com.allano.nongki.databinding.FragmentHomeBinding
14 import kotlinx.coroutines.flow.collect
15 import kotlinx.coroutines.launch
16
17 class HomeFragment : Fragment() {
18     private var _binding: FragmentHomeBinding? = null
19     private val binding get() = _binding!!
20     private lateinit var adapter: LocationAdapter
21
22     private val viewModel: LocationViewModel by viewModels
23 {
24
25 LocationViewModel.Factory(requireActivity().application)
26     }
27
28     override fun onCreateView(inflater: LayoutInflater,
   container: ViewGroup?, savedInstanceState: Bundle?): View?
29 {
30         _binding = FragmentHomeBinding.inflate(inflater,
   container, false)
31         return binding.root
32     }
33
34     override fun onViewCreated(view: View,
   savedInstanceState: Bundle?) {
35         super.onViewCreated(view, savedInstanceState)
36
37         adapter = LocationAdapter(emptyList(),
   findNavController(), viewModel)
38         binding.recyclerView.layoutManager =
   LinearLayoutManager(requireContext())
39         binding.recyclerView.adapter = adapter
40
41         lifecycleScope.launch {
42             viewModel.locations.collect { locations ->
43                 adapter.updateData(locations)
44                 Log.d("HomeFragment", "Data list
   diperbarui di UI")
45             }
```

```
46          }
47
48          lifecycleScope.launch {
49              viewModel.selectedLocation.collect { location
    ->
50                  location?.let {
51                      navigateToDetail(it)
52                  }
53              }
54          }
55      }
56
57      private fun navigateToDetail(location: LocationModel)
58  {
59          val bundle = Bundle().apply {
60              putInt("fotoResId", location.fotoResId)
61              putString("nama", location.nama)
62              putString("deskripsi", location.deskripsi)
63          }
64          Log.d("HomeFragment", "Navigasi ke detail dengan
    data: ${location.nama}")
65          findNavController().navigate(R.id.detailFragment,
    bundle)
66      }
67
68      override fun onDestroyView() {
69          super.onDestroyView()
70          _binding = null
71      }
72  }
```

3. LocationViewModel.kt

Tabel 3. Source Code Jawaban Soal 1

```
1   package com.allano.nongki
2
3   import android.app.Application
4   import android.util.Log
5   import androidx.lifecycle.AndroidViewModel
6   import androidx.lifecycle.ViewModel
7   import androidx.lifecycle.ViewModelProvider
8   import androidx.lifecycle.viewModelScope
9   import kotlinx.coroutines.flow.MutableStateFlow
10  import kotlinx.coroutines.flow.StateFlow
11  import kotlinx.coroutines.flow.asStateFlow
```

```
12 import kotlinx.coroutines.launch
13
14 class LocationViewModel(application: Application) :
   AndroidViewModel(application) {
15     private val context =
   getApplication<Application>().applicationContext
16     private val _locations =
   MutableStateFlow<List<LocationModel>>(emptyList())
17     val locations: StateFlow<List<LocationModel>> =
   _locations.asStateFlow()
18
19     private val _selectedLocation =
   MutableStateFlow<LocationModel?>(null)
20     val selectedLocation: StateFlow<LocationModel?> =
   _selectedLocation.asStateFlow()
21
22     init {
23         loadLocations()
24     }
25
26     private fun loadLocations() {
27         viewModelScope.launch {
28             val nama =
   context.resources.getStringArray(R.array.data_name)
29             val deskripsi =
   context.resources.getStringArray(R.array.data_description)
30             val rating =
   context.resources.getStringArray(R.array.data_rating)
31             val link =
   context.resources.getStringArray(R.array.data_link)
32             val fotoResId = listOf(
33                 R.drawable.image1,
34                 R.drawable.image2,
35                 R.drawable.image3,
36                 R.drawable.image4,
37                 R.drawable.image5
38             )
39
40             val locationsList = nama.indices.map { i ->
41                 LocationModel(
42                     nama[i],
43                     fotoResId[i],
44                     deskripsi[i],
45                     rating[i],
46                     link[i]
47                 ).also {
48                     Log.d("LocationViewModel", "Data item
```

```
    ditambahkan: ${it.nama}")
49                  }
50              }
51
52              _locations.value = locationsList
53              Log.d("LocationViewModel", "Data list selesai
    dimuat, total: ${locationsList.size} item")
54          }
55      }
56
57      fun selectLocation(location: LocationModel) {
58          _selectedLocation.value = location
59          Log.d("LocationViewModel", "Item dipilih:
    ${location.nama}")
60      }
61
62      class Factory(private val application: Application) :
    ViewModelProvider.Factory {
63          @Suppress("UNCHECKED_CAST")
64          override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
65              if
    (modelClass.isAssignableFrom(LocationViewModel::class.java))
66  {
67                  return LocationViewModel(application) as T
68              }
69              throw IllegalArgumentException("Unknown
    ViewModel class")
70          }
71      }
72  }
```

4. LocationAdapter.kt

Tabel 4. Source Code Jawaban Soal 1

```
1   package com.allano.nongki
2
3   import android.content.Intent
4   import android.util.Log
5   import android.view.LayoutInflater
6   import android.view.ViewGroup
7   import androidx.navigation.NavController
8   import androidx.recyclerview.widget.RecyclerView
9   import com.allano.nongki.databinding.ItemLocationBinding
10  import androidx.core.net.toUri
11
```

```
12 class LocationAdapter(
13     private var items: List<LocationModel>,
14     private val navController: NavController,
15     private val viewModel: LocationViewModel
16 ) : RecyclerView.Adapter<LocationAdapter.LocationViewHolder>() {
17
18     class LocationViewHolder(val binding: ItemLocationBinding) :
   RecyclerView.ViewHolder(binding.root)
19
20     fun updateData(newItems: List<LocationModel>) {
21         items = newItems
22         notifyDataSetChanged()
23     }
24
25     override fun onCreateViewHolder(parent: ViewGroup, viewType:
   Int): LocationViewHolder {
26         val binding =
   ItemLocationBinding.inflate(LayoutInflater.from(parent.context),
   parent, false)
27         return LocationViewHolder(binding)
28     }
29
30     override fun onBindViewHolder(holder: LocationViewHolder,
   position: Int) {
31         val item = items[position]
32         holder.binding.textViewName.text = item.nama
33         holder.binding.rating.text = item.rating
34
   holder.binding.imageView.setImageResource(item.fotoResId)
35
36         holder.binding.buttonBrowser.setOnClickListener {
37             val url = item.link
38             Log.d("LocationAdapter", "Tombol browser ditekan
   untuk: ${item.nama}, URL: $url")
39             val intent = Intent(Intent.ACTION_VIEW, url.toUri())
40             it.context.startActivity(intent)
41         }
42
43         holder.binding.buttonDetail.setOnClickListener {
44             Log.d("LocationAdapter", "Tombol detail ditekan
   untuk: ${item.nama}")
45             viewModel.selectLocation(item)
46         }
47     }
48
49     override fun getItemCount() = items.size
50 }
```

5. DetailFragment.kt

Tabel 5. Source Code Jawaban Soal 1

```
1  package com.allano.nongki
2
3  import android.os.Bundle
4  import android.util.Log
5  import android.view.LayoutInflater
6  import android.view.View
7  import android.view.ViewGroup
8  import androidx.fragment.app.Fragment
9  import com.allano.nongki.databinding.FragmentDetailBinding
10
11 class DetailFragment: Fragment() {
12     private var _binding: FragmentDetailBinding? = null
13     private val binding get() = _binding!!
14
15     override fun onCreateView(inflater: LayoutInflater,
   container: ViewGroup?, savedInstanceState: Bundle?): View?
16 {
17         _binding = FragmentDetailBinding.inflate(inflater,
   container, false)
18         return binding.root
19     }
20
21     override fun onViewCreated(view: View,
   savedInstanceState: Bundle?) {
22         super.onViewCreated(view, savedInstanceState)
23
24         val nama = arguments?.getString("nama") ?: "Tidak
   ada nama"
25         val deskripsi = arguments?.getString("deskripsi")
   ?: "Tidak ada deskripsi"
26         val fotoResId = arguments?.getInt("fotoResId") ?:
   R.drawable.image5
27
28         Log.d("DetailFragment", "Menampilkan detail untuk:
   $nama")
29         Log.d("DetailFragment", "Deskripsi: $deskripsi")
30         Log.d("DetailFragment", "Foto Resource ID:
   $fotoResId")
31
32
33 binding.imageViewDetail.setImageResource(fotoResId)
34         binding.titleViewDetail.text = nama
```

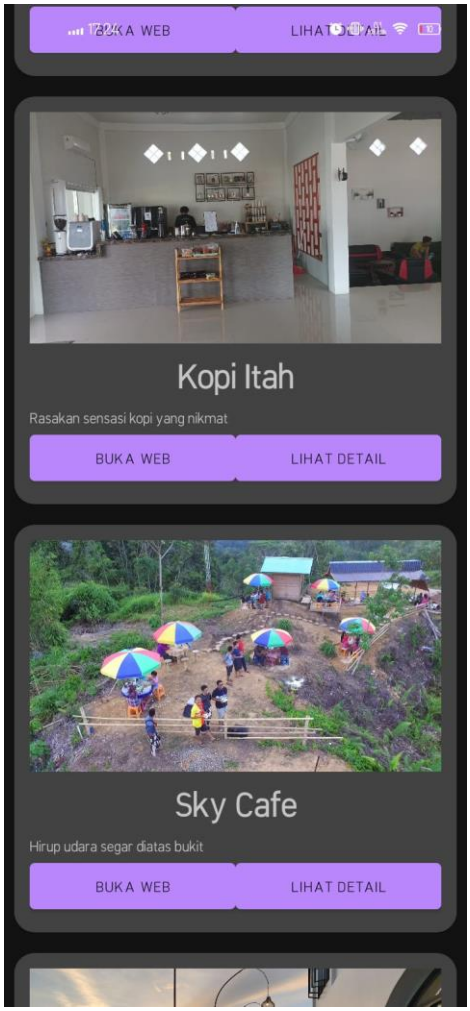
```
35          binding.textViewDetail.text = deskripsi
36      }
37
38      override fun onDestroyView() {
39          super.onDestroyView()
40          _binding = null
41      }
42  }
```

6. LocationModel.kt

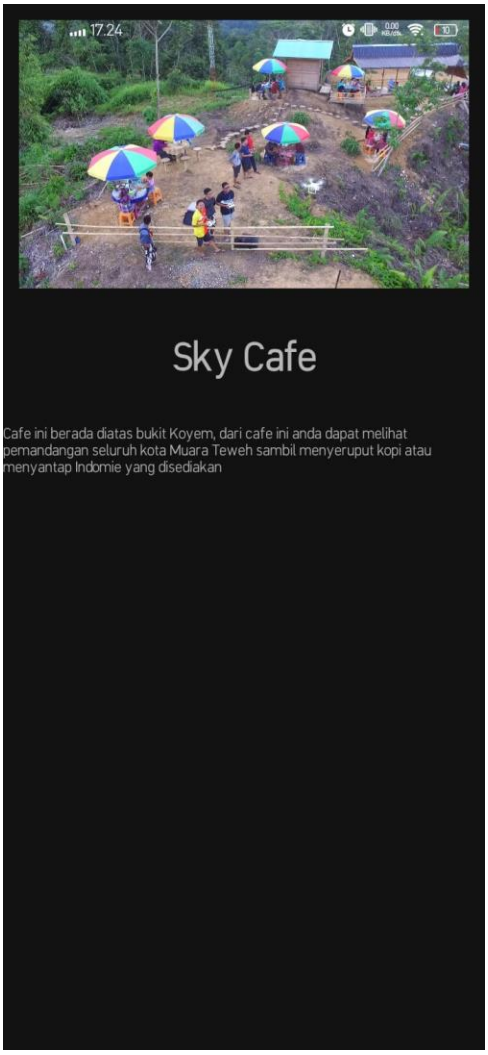
Tabel 6. Source Code Jawaban Soal 1

```
1   package com.allano.nongki
2
3   data class LocationModel (
4       val nama: String,
5       val fotoResId: Int,
6       val deskripsi: String,
7       val rating: String,
8       val link: String
9   )
```

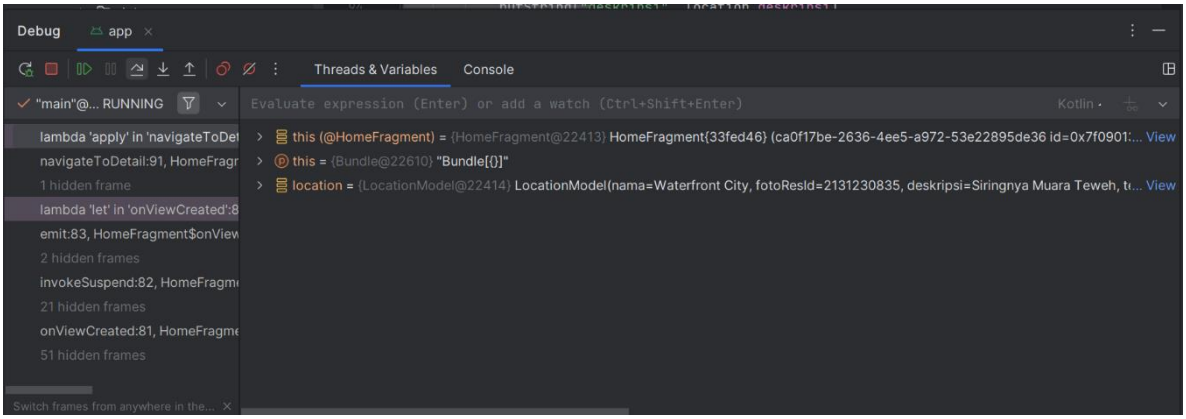
B. Output Program



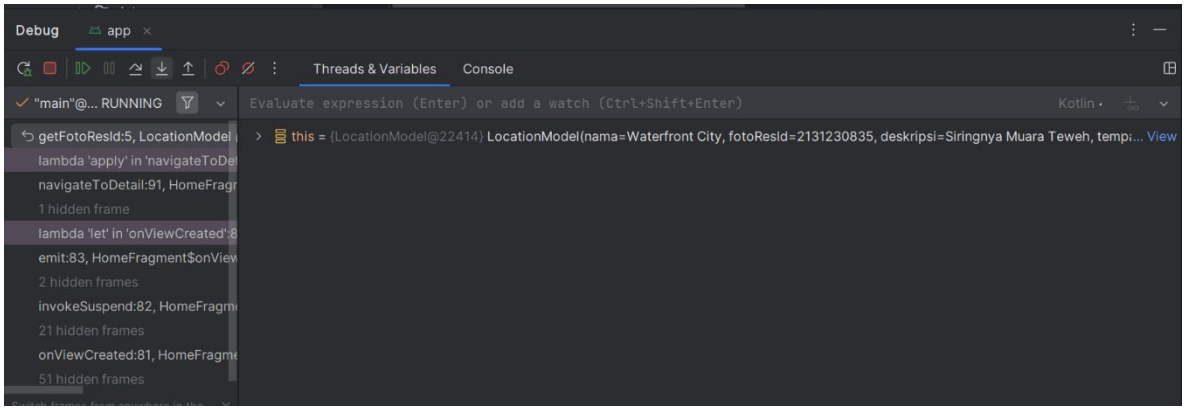
Gambar 1. Screenshot Hasil Jawaban Soal 1



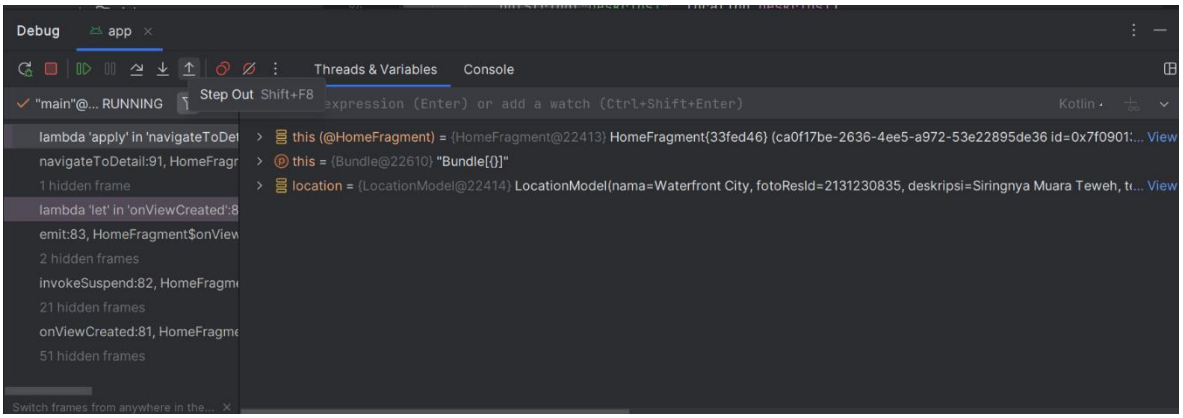
Gambar 2. Screenshot Hasil Jawaban Soal 1



Gambar 3. Debugging dengan Fitur Step Into



Gambar 4. Debugging dengan Fitur Step Over



Gambar 5. Debugging dengan Fitur Step Out

C. Pembahasan

1. MainActivity.kt:

Pada baris [1], package `com.allano.nongki` dideklarasikan sebagai namespace dari aplikasi. Selanjutnya, pada baris [2] hingga [4], berbagai library penting diimpor, termasuk komponen Android dan view binding yang diperlukan untuk mengelola tampilan. Pada baris [5], dideklarasikan kelas `MainActivity` yang merupakan turunan dari `AppCompatActivity`, yang berfungsi sebagai aktivitas utama aplikasi. Pada baris [6], variabel `binding` dideklarasikan dengan tipe `ActivityMainBinding`, yang memungkinkan akses langsung ke elemen-elemen layout XML melalui view binding.

Pada baris [8-11], metode `onCreate()` diimplementasikan sebagai titik awal saat aktivitas dijalankan. Pada baris [9], `super.onCreate(savedInstanceState)` dipanggil

untuk menjalankan inisialisasi dasar dari superclass. Di baris [10], `enableEdgeToEdge()` digunakan untuk mengaktifkan tampilan layar penuh (edge-to-edge display), memberikan pengalaman visual yang lebih modern. Kemudian, pada baris [11], layout di-inflate menggunakan `ActivityMainBinding.inflate(layoutInflater)` agar binding dapat digunakan. Akhirnya, pada baris [12], `setContentView(binding.root)` dipanggil untuk menampilkan root view dari layout sebagai tampilan utama aktivitas.

2. HomeFragment.kt

Baris [1] mendeklarasikan `package`, dan baris [3]–[12] mengimpor library yang diperlukan seperti `ViewModel`, `LiveData`, dan `RecyclerView`. Class `HomeFragment` dideklarasikan pada baris [14] sebagai turunan dari `Fragment`.

Pada baris [15]–[18] dideklarasikan properti untuk `view binding` dan `adapter`, sedangkan `ViewModel` diinisialisasi dengan delegate `by viewModels()` (baris [20]–[22]).

Method `onCreateView()` (baris [24]–[27]) digunakan untuk *inflate* layout fragment. Method `onViewCreated()` (baris [29]–[47]) menyiapkan `RecyclerView` dan mengamati data dari `ViewModel` untuk ditampilkan secara dinamis. Method `navigateToDetail()` (baris [49]–[57]) menangani navigasi ke fragment detail dengan membawa data item terpilih melalui `Bundle`. Terakhir, method `onDestroyView()` (baris [59]–[62]) digunakan untuk membersihkan objek binding agar tidak terjadi memory leak.

3. LocationViewModel.kt

Pada baris [1] dideklarasikan `package com.allano.nongki`, sedangkan baris [3]–[11] mengimpor class yang dibutuhkan, seperti `AndroidViewModel`, `StateFlow`, dan komponen coroutine.

Class `LocationViewModel` didefinisikan pada baris [13] sebagai turunan dari `AndroidViewModel`. Baris [14]–[15] menyimpan context aplikasi. Pada baris [16]–[19], dibuat dua `StateFlow`: satu untuk daftar lokasi dan satu lagi untuk lokasi yang dipilih.

Method `loadLocations()` (baris [25]–[47]) mengambil data lokasi dari `string-array` di resources, lalu mengubahnya menjadi list `LocationModel`. Method `selectLocation()` (baris [49]–[52]) digunakan untuk mengubah nilai lokasi yang dipilih. Terakhir, inner class `Factory` (baris [54]–[63]) digunakan untuk membuat instance `ViewModel` dengan context aplikasi.

4. LocationAdapter.kt

Baris [1] mendeklarasikan package, dan baris [3]–[9] mengimpor library yang diperlukan seperti RecyclerView dan ViewBinding. Class `LocationAdapter` didefinisikan pada baris [11] sebagai turunan dari `RecyclerView.Adapter`, yang digunakan untuk menampilkan daftar lokasi.

ViewHolder didefinisikan pada baris [18]–[20] untuk mengelola binding item layout. Method `updateData()` (baris [22]–[25]) berfungsi memperbarui data dan me-refresh tampilan. `onCreateViewHolder()` (baris [27]–[30]) membuat instance ViewHolder. `onBindViewHolder()` (baris [32]–[47]) mengikat data ke tampilan dan menangani klik. Method `getItemCount()` (baris [49]) mengembalikan jumlah item dalam list.

5. DetailFragment.kt

Baris [1] mendeklarasikan package, dan baris [3]–[7] mengimpor library yang diperlukan seperti Fragment, Bundle, dan ViewBinding. Class `DetailFragment` dideklarasikan pada baris [9] sebagai turunan dari `Fragment`. View binding dideklarasikan di baris [10]–[11] untuk mengakses elemen layout dengan aman.

Method `onCreateView()` (baris [13]–[16]) meng-inflate layout fragment. Pada `onViewCreated()` (baris [18]–[30]), data dari arguments diambil dan ditampilkan ke tampilan, seperti nama dan deskripsi lokasi. Logging digunakan untuk memastikan data diterima dengan benar. Method `onDestroyView()` (baris [32]–[35]) membersihkan binding untuk mencegah memory leak.

6. LocationModel.kt

Pada baris [1], package `com.allano.nongki` dideklarasikan sebagai namespace untuk file ini.
Pada baris [3], `data class LocationModel` didefinisikan sebagai kelas data yang digunakan untuk merepresentasikan informasi lokasi nongki.
Pada baris [4–8], constructor utama dari `LocationModel` dideklarasikan dengan lima parameter:

- `nama` bertipe `String`, menyimpan nama tempat
- `fotoResId` bertipe `Int`, menyimpan ID resource gambar
- `deskripsi` bertipe `String`, menyimpan deskripsi tempat
- `rating` bertipe `String`, menyimpan nilai rating tempat
- `link` bertipe `String`, menyimpan URL lokasi atau informasi tambahan

7. Debugger

Debugger berfungsi untuk memeriksa dan melacak jalannya program dengan cara menghentikan eksekusi aplikasi pada titik tertentu menggunakan breakpoint, melihat isi variabel, dan memantau kondisi aplikasi secara real time. Cara menggunakan debugger adalah dengan menekan ikon bergambar serangga atau tekan `shift+f9`.

Terdapat beberapa fitur penting pada debugger, antara lain:

- **Step Into (F7):** untuk masuk ke dalam method yang sedang dipanggil.
- **Step Over (F8):** untuk melewati method tanpa masuk ke dalamnya.
- **Step Out (Shift+F8):** untuk keluar dari method saat ini dan Kembali ke method pemanggil

SOAL 2

2. Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

A. Pembahasan

Application class dalam arsitektur aplikasi Android merupakan komponen inti yang merepresentasikan keseluruhan siklus hidup aplikasi. Kelas ini digunakan sebagai titik awal ketika aplikasi pertama kali dijalankan, sebelum aktivitas atau komponen lain dipanggil. Fungsinya antara lain untuk menginisialisasi konfigurasi global seperti library eksternal, logging, dependency injection, atau setup awal lainnya yang diperlukan oleh berbagai bagian aplikasi. Selain itu, Application class juga dapat digunakan untuk menyimpan data atau state global yang dapat diakses dari berbagai komponen, meskipun penggunaan ini perlu hati-hati agar tidak menimbulkan kebocoran memori. Dengan memanfaatkan Application class, pengembang dapat memastikan bahwa inisialisasi penting dilakukan hanya sekali dan berlaku secara konsisten selama aplikasi berjalan.

Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

https://github.com/AllanoLintang/pemro-mobile