LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Farisa Adelia **NIM. 2110817120010**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2024

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE I
MODUL 4

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

Nama Praktikan : Farisa Adelia
NIM : 2110817120010

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

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

A. Source Code

Tabel 1. Source Code MainActivity.kt

```
package com.example.modul4viewmodelanddebugging

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.NavController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.example.modul4viewmodelanddebugging.model.MovieItem
import
com.example.modul4viewmodelanddebugging.ui.theme.Modul3ScrollableListTheme
import
com.example.modul4viewmodelanddebugging.viewmodel.MovieViewModel
import
com.example.modul4viewmodelanddebugging.viewmodel.MovieViewModelFactory

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContent {
            Modul3ScrollableListTheme {
                Surface(modifier = Modifier.fillMaxSize(),
color = MaterialTheme.colorScheme.background) {
                    val viewModel: MovieViewModel =
viewModel(factory = MovieViewModelFactory())
                    MainScreen(viewModel)
                }
            }
        }
    }
}

@Composable
fun MainScreen(viewModel: MovieViewModel) {
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination =
"list") {
        composable("list") {
            MovieListScreen(navController, viewModel)
        }
        composable("detail") {
            val movie by
viewModel.selectedMovie.collectAsState()
            movie?.let {
                DetailScreen(it)
            }
        }
    }
}

@Composable
fun MovieListScreen(navController: NavController, viewModel:
MovieViewModel) {
    val context = LocalContext.current
    val movieList by viewModel.movies.collectAsState()

    LazyColumn(modifier = Modifier.padding(8.dp)) {
        items(movieList) { movie ->
            Card(
                shape = RoundedCornerShape(12.dp),
                modifier = Modifier
                    .padding(vertical = 8.dp)
                    .fillMaxWidth()
            ) {
                Column(modifier = Modifier.padding(12.dp)) {
                    Image(
                        painter = painterResource(id =
movie.imageResId),
                        contentDescription = movie.title,
                        contentScale = ContentScale.Crop,
                        modifier = Modifier
```

```
                                .fillMaxWidth()
                                .height(180.dp)
                                .clip(RoundedCornerShape(8.dp))
                        )
                        Spacer(modifier = Modifier.height(8.dp))
                        Row(Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.SpaceBetween) {
                            Text(movie.title, fontWeight =
FontWeight.Bold)
                            Text(movie.year)
                        }
                        Spacer(modifier = Modifier.height(4.dp))
                        Text("Plot: ${movie.plot}", maxLines = 3)
                        Spacer(modifier = Modifier.height(8.dp))
                        Row(Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.SpaceEvenly) {
                            Button(onClick = {
                                viewModel.onImdbClick(movie)
                                val intent =
Intent(Intent.ACTION_VIEW, Uri.parse(movie.imdbLink))
                                context.startActivity(intent)
                            }) {
                                Text("IMDB")
                            }
                            Button(onClick = {
                                viewModel.selectMovie(movie)
                                viewModel.onDetailClick(movie)
                                navController.navigate("detail")
                            }) {
                                Text("Detail")
                            }
                        }
                    }
                }
            }
        }
    }
}

@Composable
fun DetailScreen(movie: MovieItem) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Image(
            painter = painterResource(id = movie.imageResId),
            contentDescription = movie.title,
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxWidth()
                .height(250.dp)
```

```
                .clip(RoundedCornerShape(12.dp))
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(movie.title, style =
MaterialTheme.typography.titleLarge)
        Text(movie.year, style =
MaterialTheme.typography.labelMedium)
        Spacer(modifier = Modifier.height(8.dp))
        Text("Plot:", fontWeight = FontWeight.Bold)
        Text(movie.plot)
    }
}
```

Tabel 2. Source Code AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"

android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Modul3ScrollableList"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.Modul3ScrollableList">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Tabel 3. Source Code MovieItem.kt

```
package com.example.modul4viewmodelanddebugging.model

data class MovieItem(
    val id: Int,
    val title: String,
    val year: String,
    val plot: String,
    val imageResId: Int,
    val imdbLink: String
)
```

Tabel 4. Source Code MovieViewModel.kt

```
package com.example.modul4viewmodelanddebugging.viewmodel

import android.util.Log
import androidx.lifecycle.ViewModel
import com.example.modul4viewmodelanddebugging.R
import com.example.modul4viewmodelanddebugging.model.MovieItem
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow

class MovieViewModel : ViewModel() {

    private val _movies =
MutableStateFlow<List<MovieItem>>(emptyList())
    val movies: StateFlow<List<MovieItem>> = _movies

    private val _selectedMovie =
MutableStateFlow<MovieItem?>(null)
    val selectedMovie: StateFlow<MovieItem?> = _selectedMovie

    init {
        val sample = listOf(
            MovieItem(1, "Pengabdi Setan 2: Communion", "2022",
"When the heavy storm hits...", R.drawable.pengabdi,
"https://www.imdb.com"),
            MovieItem(2, "Siksa Kubur", "2024", "Tells about
the punishment of the grave...", R.drawable.siksa,
"https://www.imdb.com"),
            MovieItem(3, "Pengepungan di Bukit Duri", "2025",
"A special school for troubled children...",
R.drawable.bukitduri, "https://www.imdb.com")
        )

        Log.d("MovieViewModel", "Data masuk ke dalam list:
${sample.size} item")
        _movies.value = sample
    }
```

```
    fun selectMovie(movie: MovieItem) {
        Log.d("MovieViewModel", "Data yang dipilih untuk
detail: $movie")
        _selectedMovie.value = movie
    }

    fun onImdbClick(movie: MovieItem) {
        Log.d("MovieViewModel", "Tombol IMDB ditekan:
${movie.title}")
    }

    fun onDetailClick(movie: MovieItem) {
        Log.d("MovieViewModel", "Tombol Detail ditekan:
${movie.title}")
    }
}
```

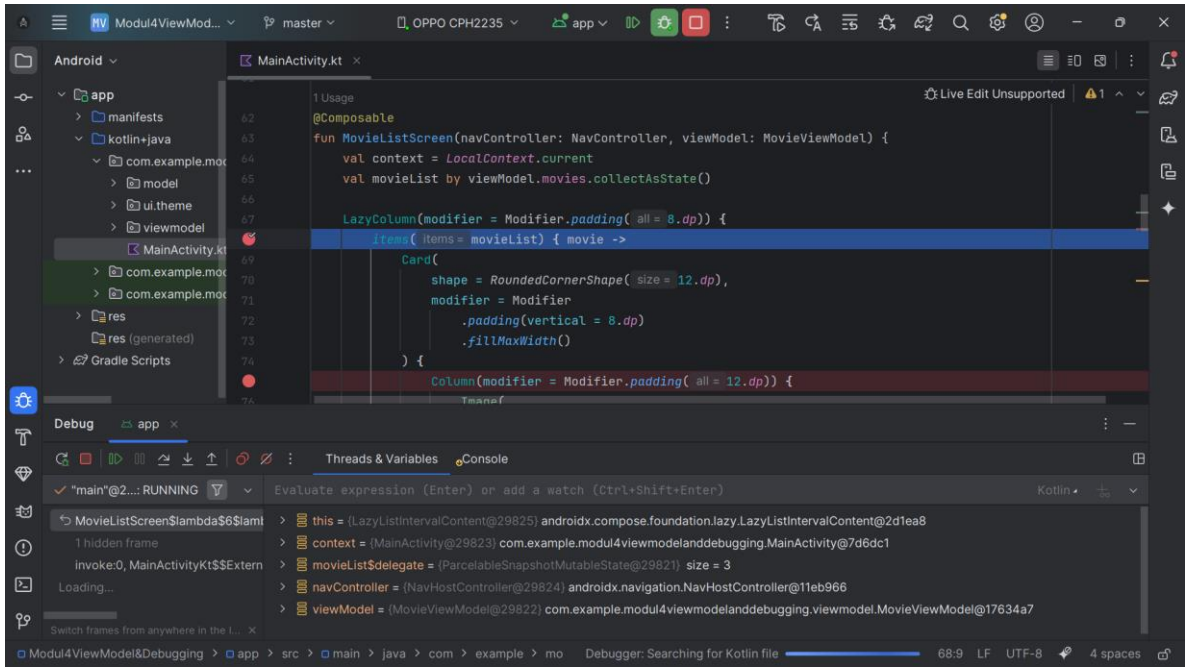
Tabel 5. Source Code MovieViewModelFactory.kt

```
package com.example.modul4viewmodelanddebugging.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider

class MovieViewModelFactory : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>):
T {
        return MovieViewModel() as T
    }
}
```

B. Output Program



Gambar 1. Screenshoot Hasil Jawaban Soal 1

Siksa Kubur

2024

Plot:

Tells about the punishment of the grave...

Gambar 2. Screenshoot Hasil Jawaban Soal 1

Gambar 3. Screenshoot Hasil Jawaban Soal 1

C. Pembahasan

- ViewModel untuk Pengolahan Data
  - ViewModel digunakan sebagai komponen arsitektural untuk menyimpan dan mengelola data list item
  - Pendekatan ini mengikuti prinsip MVVM (Model-View-ViewModel) dan memastikan data tetap terjaga meskipun konfigurasi perangkat berubah (misalnya rotasi layar).
  - Data tidak disimpan di Activity/Fragment, melainkan di ViewModel yang bersifat lifecycle-aware.
- ViewModelFactory
  - ViewModel dibuat menggunakan ViewModelProvider.Factory untuk menyuplai dependensi jika dibutuhkan (misalnya repository atau argumen).
  - Hal ini membantu ketika ViewModel memiliki konstruktor non-default.
- StateFlow untuk State Management
  - StateFlow digunakan untuk mengelola event onClick dan list data item dari ViewModel ke Fragment.
  - Dibanding LiveData, StateFlow menawarkan pendekatan reaktif dan mendukung coroutine secara penuh.
  - StateFlow digunakan dengan collectAsState() dalam Jetpack Compose agar UI otomatis merespons perubahan data.
- Logging
  - Logging ditambahkan untuk membantu proses debugging dan pemantauan perilaku aplikasi
  - Saat data item ditambahkan ke list

    ```
    Log.d("MovieViewModel", "Item ditambahkan:
    ${movie.title}")
    ```
  - Saat tombol “Detail” dan “Explicit Intent” ditekan

    ```
    Log.d("MovieListScreen", "Tombol Detail ditekan untuk
    id: ${movie.id}")
    Log.d("MovieListScreen", "Explicit Intent ditekan:
    ${movie.imdbUrl}")
    ```

- Saat berpindah ke halaman Detail

  ```
  Log.d("DetailScreen", "Menampilkan data untuk id: $movieId")
  ```

- Debugging Menggunakan Debugger

  Debugger di Android Studio digunakan untuk menganalisis perilaku aplikasi saat runtime:

  - Breakpoint ditempatkan pada baris pemrosesan intent navigasi untuk memantau data yang diparsing.
  - Step Into: Masuk ke dalam fungsi yang sedang dipanggil, berguna untuk memeriksa logika internal.
  - Step Over: Melewati fungsi tanpa masuk ke dalamnya, melanjutkan eksekusi ke baris berikutnya.
  - Step Out: Keluar dari fungsi saat ini dan kembali ke pemanggil sebelumnya.

SOAL 2

Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

JAWABAN : Application class adalah kelas dasar yang menggambarkan seluruh siklus dari sebuah aplikasi. Class ini hanya dibuat sekali saat aplikasi dijalankan dan bertahan selama aplikasi masih aktif di memori. Fungsi utama dari Application Class adalah sebagai tempat untuk melakukan inisialisasi yang hanya dilakukan sekali, menyediakan akses global ke aplikasi, menjadi entry poin untuk konfigurasi aplikasi, dan dapat memonitoring seperti menangkap error atau mengawasi aplikasi secara menyeluruh.

LINK GITHUB : https://github.com/FarisaAdelia/Pemrograman-Mobile