

**LAPORAN PRAKTIKUM  
PEMROGRAMAN MOBILE  
MODUL 5**



**CONNECT TO THE INTERNET**

**Oleh:**

**Devi Hafida Ariyani                    NIM. 2310817220018**

**PROGRAM STUDI TEKNOLOGI INFORMASI  
FAKULTAS TEKNIK  
UNIVERSITAS LAMBUNG MANGKURAT  
JUNI 2025**

**LEMBAR PENGESAHAN**  
**LAPORAN PRAKTIKUM PEMROGRAMAN I**  
**MODUL 5**

Laporan Praktikum Pemrograman Mobile Modul 5: Connect to the Internet ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Devi Hafida Ariyani  
NIM : 2310817220018

Menyetujui,  
Asisten Praktikum

Mengetahui,  
Dosen Penanggung Jawab Praktikum

Muhammad Raka Azwar  
NIM. 2210817210012

Andreyan Rizky Baskara, S.Kom., M.Kom.  
NIP. 19930703 201903 01 011

## **DAFTAR ISI**

<b>LEMBAR PENGESAHAN .....</b>	<b>2</b>
<b>DAFTAR ISI .....</b>	<b>3</b>
<b>DAFTAR GAMBAR .....</b>	<b>4</b>
<b>DAFTAR TABEL .....</b>	<b>5</b>
<b>SOAL 1 .....</b>	<b>6</b>
A. SOURCE CODE .....	6
B. OUTPUT PROGRAM .....	16
C. PEMBAHASAN .....	19
D. TAUTAN GIT .....	21

## **DAFTAR GAMBAR**

Gambar 1. Soal 1 .....	16
Gambar 2. Soal 1.....	16
Gambar 3. Soal 1.....	17
Gambar 4. Soal 1.....	17
Gambar 5. Soal 1.....	18
Gambar 6. Soal 1.....	18

## **DAFTAR TABEL**

Tabel 1. Source Code Jawaban Soal 1.....	6
Tabel 2. Source Code Jawaban Soal 1.....	7
Tabel 3. Source Code Jawaban Soal 1.....	7
Tabel 4. Source Code Jawaban Soal 1.....	8
Tabel 5. Source Code Jawaban Soal 1.....	8
Tabel 6. Source Code Jawaban Soal 1.....	10
Tabel 7. Source Code Jawaban Soal 1.....	11
Tabel 8. Source Code Jawaban Soal 1.....	11
Tabel 9. Source Code Jawaban Soal 1.....	12
Tabel 10. Source Code Jawaban Soal 1 .....	12
Tabel 11. Source Code Jawaban Soal 1 .....	13
Tabel 12. Source Code Jawaban Soal 1 .....	13
Tabel 13. Source Code Jawaban Soal 1 .....	14
Tabel 14. Source Code Jawaban Soal 1 .....	14
Tabel 15. Source Code Jawaban Soal 1 .....	14
Tabel 16. Source Code Jawaban Soal 1 .....	15
Tabel 17. Source Code Jawaban Soal 1 .....	15

## SOAL 1

### Soal Praktikum:

Lanjutkan aplikasi Android yang sudah dibuat pada Modul 4 dengan menambahkan modifikasi sesuai ketentuan berikut:

- a. Gunakan networking library seperti Retrofit atau Ktor agar aplikasi dapat mengambil data dari remote API. Dalam penggunaan networking library, sertakan generic response untuk status dan error handling pada API dan Flow untuk data stream.
- b. Gunakan KotlinX Serialization sebagai library JSON.
- c. Gunakan library seperti Coil atau Glide untuk image loading.
- d. API yang digunakan pada modul ini adalah The Movie Database (TMDB) API yang menampilkan data film. Berikut link dokumentasi API:  
<https://developer.themoviedb.org/docs/getting-started>
- e. Implementasikan konsep data persistence (aplikasi menyimpan data walau pengguna keluar dari aplikasi) dengan SharedPreferences untuk menyimpan data ringan (seperti pengaturan aplikasi) dan Room untuk data relasional.
- f. Gunakan caching strategy pada Room. Dibebaskan untuk memilih caching strategy yang sesuai, dan sertakan penjelasan kenapa menggunakan caching strategy tersebut.
- g. Untuk Modul 5, bebas memilih UI yang ingin digunakan, antara berbasis XML atau Jetpack Compose.

Aplikasi harus mempertahankan fitur-fitur yang dibuat pada modul sebelumnya.

### A. SOURCE CODE

Compose:

MainActivity.kt

Tabel 1. Source Code Jawaban Soal 1

```
package com.example.listfilm

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.compose.rememberNavController
import com.example.listfilm.repository.MovieRepository
import com.example.listfilm.ui.screen.AppNavGraph
import com.example.listfilm.viewmodel.MovieViewModel
import com.example.listfilm.viewmodel.MovieViewModelFactory
import timber.log.Timber

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

        Timber.d("MainActivity onCreate() called")
    }
}
```

```

        setContent {
            val navController = rememberNavController()
            val context = applicationContext
            val repository = MovieRepository(context)
            val viewModel: MovieViewModel = viewModel(
                factory = MovieViewModelFactory(repository)
            )
            AppNavGraph(navController = navController, viewModel =
viewModel)
        }
    }
}

```

## MovieDao.kt

*Tabel 2. Source Code Jawaban Soal 1*

```

package com.example.listfilm.data.local

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface MovieDao {
    @Query("SELECT * FROM movies")
    suspend fun getAllMovies(): List<MovieEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(movies: List<MovieEntity>)
}

```

## MovieDatabase.kt

*Tabel 3. Source Code Jawaban Soal 1*

```

package com.example.listfilm.data.local

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [MovieEntity::class], version = 1)
abstract class MovieDatabase : RoomDatabase() {
    abstract fun movieDao(): MovieDao

    companion object {
        @Volatile private var INSTANCE: MovieDatabase? = null

        fun getDatabase(context: Context): MovieDatabase {
            return INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(

```

```
        context.applicationContext,  
        MovieDatabase::class.java,  
        "movie_database"  
    ).build().also { INSTANCE = it }  
}  
}  
}
```

## MovieEntity.kt

*Tabel 4. Source Code Jawaban Soal 1*

```
package com.example.listfilm.data.local

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "movies")
data class MovieEntity(
    @PrimaryKey val id: Int,
    val title: String,
    val posterPath: String,
    val overview: String
)
```

## FilmListScreen.kt

*Tabel 5. Source Code Jawaban Soal 1*

```
package com.example.listfilm.ui.screen

import android.content.Intent
import android.net.Uri
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
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.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.navigation.NavHostController
import coil.compose.rememberAsyncImagePainter
import com.example.listfilm.model.Movie
import com.example.listfilm.util.Constants
import com.example.listfilm.viewmodel.MovieViewModel

@Composable
fun FilmListScreen(navController: NavHostController, viewModel: MovieViewModel) {
    val context = LocalContext.current
    val movies = viewModel.movies.collectAsState().value

    LazyColumn(contentPadding = PaddingValues(16.dp)) {
        items(movies) { movie: Movie ->
            Card(
                shape = RoundedCornerShape(16.dp),
                modifier = Modifier
                    .padding(bottom = 16.dp)
                    .fillMaxWidth()
            ) {
                Column(Modifier.padding(16.dp)) {
                    // 📸 Gambar online dari TMDB pakai Coil
                    Image(
                        painter =
rememberAsyncImagePainter("${Constants.IMAGE_BASE_URL}${movie.posterPath}"),
                        contentDescription = null,
                        modifier = Modifier
                            .fillMaxWidth()
                            .height(200.dp)
                            .clip(RoundedCornerShape(12.dp)),
                        contentScale = ContentScale.Crop
                    )

                    Spacer(Modifier.height(8.dp))
                    Text(text = movie.title, fontWeight = FontWeight.Bold)
                    Text(text = movie.overview, maxLines = 3)

                    Spacer(Modifier.height(8.dp))
                    Row(
                        modifier = Modifier.fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween
                    ) {
                        Button(onClick = {
                            viewModel.logOpenSiteClicked(movie)
                            val intent = Intent(
                                Intent.ACTION_VIEW,
                                Uri.parse("https://www.themoviedb.org/movie/${movie.id}")
                            )
                            context.startActivity(intent)
                        })
                    }
                }
            }
        }
    }
}

```

NavGraph.kt

*Tabel 6. Source Code Jawaban Soal 1*

```
package com.example.listfilm.ui.screen

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.ui.Modifier
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import com.example.listfilm.viewmodel.MovieViewModel

@Composable
fun AppNavGraph(navController: NavHostController, viewModel: MovieViewModel) {
    val movies = viewModel.movies.collectAsState().value

    NavHost(navController = navController, startDestination = "list")
    {
        composable("list") {
            FilmListScreen(navController = navController, viewModel = viewModel)
        }
        composable("detail/{id}") { backStackEntry ->
            val id = backStackEntry.arguments?.getString("id")?.toIntOrNull()
            val movie = movies.find { it.id == id }
            if (movie != null) {
                MovieDetailScreen(movie = movie, navController = navController)
            } else {
                Text("Movie not found", modifier = Modifier.fillMaxSize())
            }
        }
    }
}
```

```
        }
    }
}
```

## MovieDetailScreen.kt

Tabel 7. Source Code Jawaban Soal 1

```
package com.example.listfilm.ui.screen

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.NavHostController
import com.example.listfilm.model.Movie

@Composable
fun MovieDetailScreen(movie: Movie, navController: NavHostController) {
    Column(modifier = Modifier
        .fillMaxSize()
        .padding(16.dp)) {
        Text(text = movie.title, style =
MaterialTheme.typography.headlineMedium)
        Spacer(modifier = Modifier.height(8.dp))
        Text(text = movie.overview)
        Spacer(modifier = Modifier.height(16.dp))

        androidx.compose.material3.Button(onClick =
navController.popBackStack() ) {
            Text("Back")
        }
    }
}
```

## MovieViewModel.kt

Tabel 8. Source Code Jawaban Soal 1

```
package com.example.listfilm.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.listfilm.model.Movie
import com.example.listfilm.repository.MovieRepository
import com.example.listfilm.util.Constants
```

```

import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.stateIn
import timber.log.Timber

class MovieViewModel(private val repository: MovieRepository) : ViewModel() {
    val movies: StateFlow<List<Movie>> = repository.getPopularMovies(Constants.API_KEY)
        .stateIn(viewModelScope, SharingStarted.Eagerly, emptyList())

    fun logDetailClicked(movie: Movie) {
        Timber.d("Detail clicked: ${movie.title}")
    }

    fun logOpenSiteClicked(movie: Movie) {
        Timber.d("Open site clicked: ${movie.id}")
    }
}

```

## ListFilmApp.kt

Tabel 9. Source Code Jawaban Soal 1

```

package com.example.listfilm

import android.app.Application
import timber.log.Timber

class ListFilmApp : Application() {
    override fun onCreate() {
        super.onCreate()
        Timber.plant(Timber.DebugTree())
    }
}

```

## MovieViewModelFactory.kt

Tabel 10. Source Code Jawaban Soal 1

```

package com.example.listfilm.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import com.example.listfilm.repository.MovieRepository

class MovieViewModelFactory(
    private val repository: MovieRepository
) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(MovieViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return MovieViewModel(repository) as T
        }
    }
}

```

```

        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}

```

## Constants.kt

Tabel 11. Source Code Jawaban Soal 1

```

package com.example.listfilm.util

object Constants {
    const val BASE_URL = "https://api.themoviedb.org/3/"
    const val IMAGE_BASE_URL = "https://image.tmdb.org/t/p/w500"
    const val API_KEY = "8daab9d30b9168bb81cb48d5fcdf302"
}

```

## MovieRepository.kt

Tabel 12. Source Code Jawaban Soal 1

```

package com.example.listfilm.repository

import android.content.Context
import com.example.listfilm.data.local.MovieDatabase
import com.example.listfilm.data.local.MovieEntity
import com.example.listfilm.model.Movie
import com.example.listfilm.network.RetrofitInstance
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import timber.log.Timber

class MovieRepository(context: Context) {
    private val movieDao = MovieDatabase.getDatabase(context).movieDao()

    fun getPopularMovies(apiKey: String): Flow<List<Movie>> = flow {
        try {
            val response = RetrofitInstance.api.getPopularMovies(apiKey)
            Timber.d("Ambil dari API berhasil: ${response.results.size}")
            val movieEntities = response.results.map {
                MovieEntity(it.id, it.title, it.posterPath, it.overview)
            }
            movieDao.insertAll(movieEntities)
            emit(response.results) // from API
        } catch (e: Exception) {
            Timber.e("Gagal ambil dari API: ${e.message}")
            val cached = movieDao.getAllMovies().map {
                Movie(it.id, it.title, it.posterPath, it.overview)
            }
            emit(cached) // fallback to local Room
        }
    }
}

```

```
    }  
}
```

## PreferenceManager.kt

Tabel 13. Source Code Jawaban Soal 1

```
package com.example.listfilm.data.preferences  
  
import android.content.Context  
import android.content.SharedPreferences  
  
class PreferenceManager(context: Context) {  
    private val prefs: SharedPreferences =  
        context.getSharedPreferences("user_prefs", Context.MODE_PRIVATE)  
  
    fun setDarkMode(enabled: Boolean) {  
        prefs.edit().putBoolean("dark_mode", enabled).apply()  
    }  
  
    fun isDarkMode(): Boolean = prefs.getBoolean("dark_mode", false)  
}
```

## Movie.kt

Tabel 14. Source Code Jawaban Soal 1

```
package com.example.listfilm.model  
  
import kotlinx.serialization.SerializedName  
import kotlinx.serialization.Serializable  
  
@Serializable  
data class Movie(  
    val id: Int,  
    val title: String,  
    @SerializedName("poster_path") val posterPath: String,  
    val overview: String  
)
```

## MovieResponse.kt

Tabel 15. Source Code Jawaban Soal 1

```
package com.example.listfilm.model
```

```

import kotlinx.serialization.Serializable

@Serializable
data class MovieResponse(
    val page: Int,
    val results: List<Movie>
)

```

## RetrofitInstance.kt

*Tabel 16. Source Code Jawaban Soal 1*

```

package com.example.listfilm.network

import
com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterFactory
import kotlinx.serialization.json.Json
import okhttp3.MediaType.Companion.toMediaType
import retrofit2.Retrofit

object RetrofitInstance {
    private val json = Json { ignoreUnknownKeys = true }
    private val contentType = "application/json".toMediaType()

    val api: Tmdb ApiService by lazy {
        Retrofit.Builder()
            .baseUrl("https://api.themoviedb.org/3/")
            .addConverterFactory(json.asConverterFactory(contentType))
            .build()
            .create(Tmdb ApiService::class.java)
    }
}

```

## Tmdb ApiService.kt

*Tabel 17. Source Code Jawaban Soal 1*

```

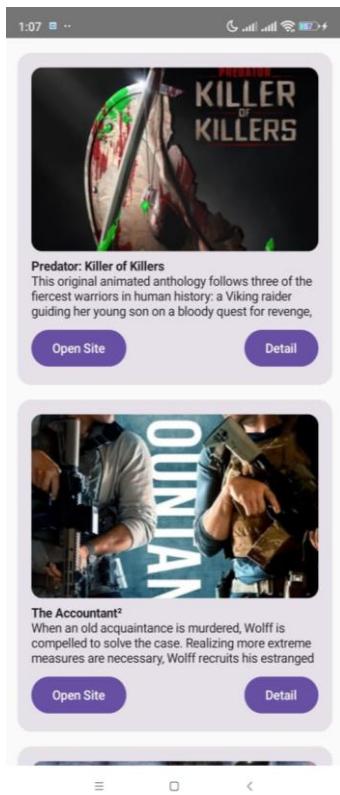
package com.example.listfilm.network

import com.example.listfilm.model.MovieResponse
import retrofit2.http.GET
import retrofit2.http.Query

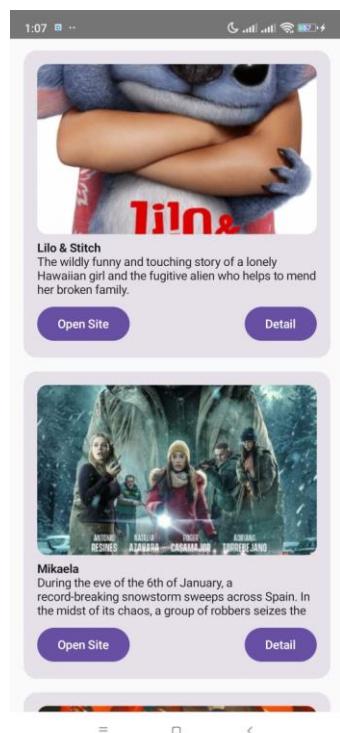
interface Tmdb ApiService {
    @GET("movie/popular")
    suspend fun getPopularMovies(
        @Query("api_key") apiKey: String
    ): MovieResponse
}

```

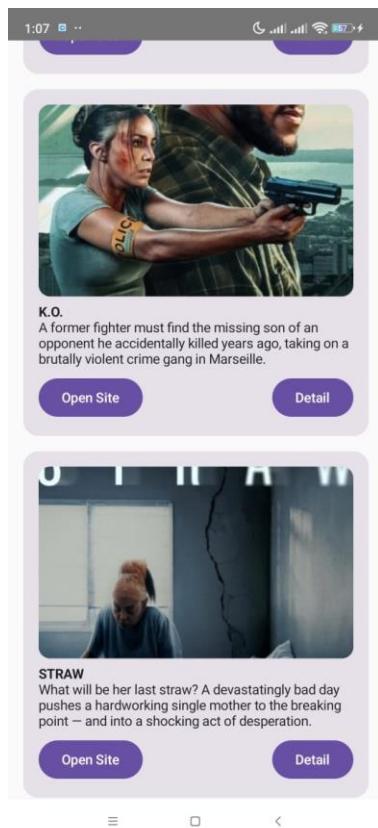
## B. OUTPUT PROGRAM



Gambar 1. Soal 1



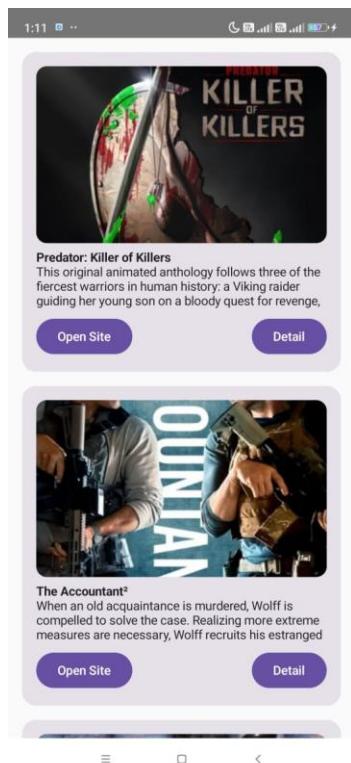
Gambar 2. Soal 1



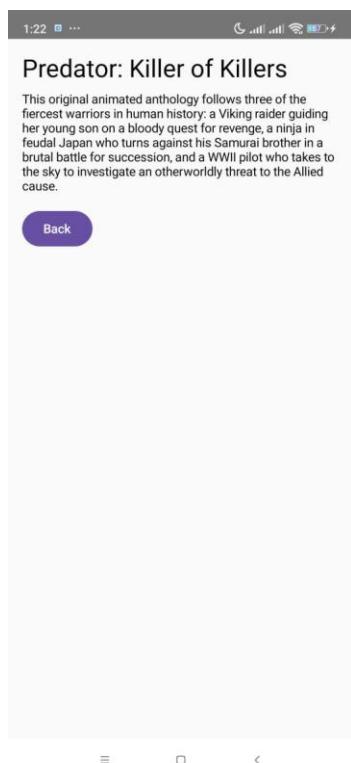
Gambar 3. Soal 1



Gambar 4. Soal 1



Gambar 5. Soal 1



Gambar 6. Soal 1

## C. PEMBAHASAN

### 1. MainActivity.kt

File ini adalah titik awal eksekusi aplikasi Android. Di dalamnya, MovieViewModel dibuat menggunakan MovieViewModelFactory, dan AppNavGraph dipanggil untuk menampilkan tampilan berbasis navigasi Compose. Timber digunakan untuk logging, memudahkan debugging.

### 2. MovieDao.kt

Interface DAO (Data Access Object) untuk Room Database. Berisi dua fungsi utama: getAllMovies() untuk mengambil semua data film dari lokal database, dan insertAll() untuk menyimpan daftar film ke database secara massal dengan strategi konflik REPLACE.

### 3. MovieDatabase.kt

Mengatur konfigurasi database Room. Menggunakan singleton pattern untuk memastikan hanya satu instance database yang dibuat selama siklus hidup aplikasi. Berisi movieDao() yang digunakan untuk akses data lokal.

### 4. MovieEntity.kt

Model entity untuk tabel database lokal dengan nama movies. Menyimpan informasi penting seperti id, title, posterPath, dan overview. Tabel ini mencerminkan data film yang akan disimpan secara lokal untuk caching.

### 5. FilmListScreen.kt

Composable yang menampilkan daftar film dalam bentuk LazyColumn. Menggunakan Coil untuk memuat gambar poster dari internet, serta menyediakan dua tombol: Open Site (membuka detail film di browser) dan Detail (navigasi ke halaman detail).

### 6. NavGraph.kt

File ini menangani navigasi layar dengan NavHost. Terdapat dua rute: "list" untuk menampilkan daftar film (FilmListScreen) dan "detail/{id}" untuk detail film (MovieDetailScreen). Data di-passing melalui parameter id.

### 7. MovieDetailScreen.kt

Menampilkan detail dari film yang dipilih. Ditampilkan judul dan overview-nya. Terdapat tombol "Back" untuk kembali ke daftar. Ini adalah UI untuk route "detail/{id}".

### 8. MovieViewModel.kt

ViewModel yang bertugas mengelola state daftar film dengan StateFlow. Data diambil dari repository dengan fallback ke Room jika gagal memuat dari API. Logging disediakan untuk mencatat aksi pengguna seperti klik detail dan open site.

### 9. ListFilmApp.kt

Subclass dari Application yang digunakan untuk inisialisasi Timber sebagai tool logging. File ini penting untuk memantau log aplikasi di seluruh lifecycle-nya.

### 10. MovieViewModelFactory.kt

Fungsi factory yang membuat instance MovieViewModel, diperlukan karena MovieViewModel butuh parameter (MovieRepository). Factory ini digunakan dalam MainActivity.

#### 11. Constants.kt

File utilitas berisi konstanta global seperti BASE\_URL, IMAGE\_BASE\_URL, dan API\_KEY. Dipakai untuk keperluan API dan gambar film.

#### 12. MovieRepository.kt

Repositori yang menjadi penghubung antara network (API) dan local database (Room). Saat getPopularMovies() dipanggil:

- Coba ambil data dari API.
- Jika berhasil: simpan ke Room (cache), dan tampilkan ke UI.
- Jika gagal: ambil data dari Room sebagai fallback (strategi caching).

#### 13. PreferenceManager.kt

Menyediakan cara menyimpan dan membaca preferensi pengguna dengan SharedPreferences, contohnya: status mode gelap. Bukan bagian dari fitur utama film, tapi berguna untuk personalisasi.

#### 14. Movie.kt

Model data utama yang digunakan di UI dan networking. Menggunakan anotasi @Serializable agar bisa di-serialize menggunakan KotlinX Serialization, terutama saat parsing JSON dari API.

#### 15. MovieResponse.kt

Model data respon dari API yang mengandung daftar film (results) dan halaman (page). Ini adalah struktur JSON dari endpoint /movie/popular.

#### 16. RetrofitInstance.kt

Konfigurasi Retrofit untuk koneksi ke TMDB API. Menggunakan kotlinx.serialization sebagai converter JSON, dan disiapkan secara singleton untuk efisiensi.

#### 17. Tmdb ApiService.kt

Interface untuk mendefinisikan endpoint API. Dalam hal ini, hanya satu fungsi getPopularMovies() yang menggunakan metode GET dan menerima parameter api\_key.

Aplikasi ini menerapkan strategi caching "Network First, Fallback to Cache" dengan menggunakan Room sebagai local database. Pada strategi ini, aplikasi akan mengambil data terlebih dahulu dari jaringan (API TMDB) setiap kali pengguna membuka atau memuat data. Jika permintaan jaringan berhasil, data film yang diterima dari API akan langsung disimpan ke Room Database melalui movieDao.insertAll(...). Dengan demikian, data yang ditampilkan ke pengguna selalu up-to-date saat ada koneksi internet.

Namun, jika terjadi kegagalan saat mengambil data dari API—misalnya karena perangkat tidak memiliki koneksi internet—maka aplikasi akan mengambil data dari database lokal (Room) sebagai cadangan. Proses ini dilakukan melalui movieDao.getAllMovies(), yang mengembalikan daftar film yang sebelumnya telah disimpan ketika jaringan masih tersedia.

Strategi ini dipilih karena menawarkan keseimbangan antara kesegaran data (freshness) dan ketersediaan data secara offline. Dengan cara ini, pengguna akan selalu mendapatkan data terbaru selama mereka terhubung ke internet. Namun, ketika sedang offline, aplikasi tetap dapat menampilkan informasi film yang tersimpan secara lokal, memberikan pengalaman pengguna yang mulus tanpa pesan kesalahan atau layar kosong.

#### **D. TAUTAN GIT**

Berikut adalah tautan untuk source code yang telah dibuat.

<https://github.com/DeviHafida/Mobile/tree/master>