LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Natalie Grace Katiandagho NIM. 2310817120003

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 Prakitkum ini dikerjakan oleh:

Nama Praktikan : Natalie Grace Katiandagho
NIM : 2310817120003

Menyetujui,	Mengetahui,
Asisten Praktikum	Dosen Penanggung Jawab Praktikum
Natalie Grace Katiandagho	Muti`a Maulida S.Kom M.T.I
NIM. 2310817120003	NIP. 19881027 201903 20 13

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR GAMBAR

SOAL 1

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 bebas, contoh API gratis 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 (misalnya offline-first app, pengaturan dark/light mode, fitur favorite, dll)

f. Gunakan caching strategy pada Room..

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

Tabel 1. MovieDao.kt Modul 5

```
01  package com.example.movielist.data.local.dao
2
3   import androidx.room.Dao
4   import androidx.room.Insert
5   import androidx.room.OnConflictStrategy
6   import androidx.room.Query
7   import
8   com.example.movielist.data.local.entities.MovieEntity
9
10  @Dao
11  interface MovieDao {
12      @Insert(onConflict = OnConflictStrategy.REPLACE)
13      suspend fun insertAllMovies(movies:
    List<MovieEntity>)
14
15      @Query("SELECT * FROM movies ORDER BY popularity
    DESC")
16      suspend fun getAllMovies(): List<MovieEntity>
```

```
17      @Query("DELETE FROM movies")
18      suspend fun clearAllMovies()
19  }
```

Tabel 2. AppDatabase.kt Modul 5

```
01  package com.example.movielist.data.local.database

2   import androidx.room.Database
3   import androidx.room.RoomDatabase
4   import com.example.movielist.data.local.dao.MovieDao
5   import
6   com.example.movielist.data.local.entities.MovieEntity
7
8   @Database(entities = [MovieEntity::class], version =
    1, exportSchema = false)
9   abstract class AppDatabase : RoomDatabase() {
10      abstract fun movieDao(): MovieDao
11
12      companion object {
13          const val DATABASE_NAME = "tmdb_app_db"
14      }
15  }
```

Tabel 3. MovieAppPreferences.kt Modul 5

```
01  package com.example.movielist.data.local
2
3   import android.content.Context
4   import android.content.SharedPreferences
5
6   class MovieAppPreferences(context: Context) {
7
8       private val sharedPreferences:
    SharedPreferences =
9
10  context.getSharedPreferences("tmdb_app_prefs",
    Context.MODE_PRIVATE)
11
12      companion object {
13          private const val KEY_API_KEY = "api_key"
14          private const val KEY_DARK_MODE =
    "dark_mode"
15      }
```

```
    fun saveApiKey(apiKey: String) {

sharedPreferences.edit().putString(KEY_API_KEY,
apiKey).apply()
    }

    fun getApiKey(): String? {
        return
sharedPreferences.getString(KEY_API_KEY, null)
    }

    fun saveDarkModeState(isDarkMode: Boolean) {

sharedPreferences.edit().putBoolean(KEY_DARK_MODE,
isDarkMode).apply()
    }

    fun getDarkModeState(): Boolean {
        return
sharedPreferences.getBoolean(KEY_DARK_MODE, false)
    }
}
```

Tabel 4. RetrofitClient.kt Modul 5

```
package com.example.movielist.data.remote.api


import
com.jakewharton.retrofit2.converter.kotlinx.serializatio
n.asConverterFactory
import kotlinx.serialization.json.Json
import okhttp3.MediaType.Companion.toMediaType
import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import java.util.concurrent.TimeUnit

object RetrofitClient {

    private const val BASE_URL =
"https://api.themoviedb.org/3/"

    private val json = Json {
        ignoreUnknownKeys = true
        prettyPrint = true
```

```
17      }
18
19      private val okHttpClient: OkHttpClient by lazy {
20          val logging = HttpLoggingInterceptor()
21
22  logging.setLevel(HttpLoggingInterceptor.Level.BODY)
23
24          OkHttpClient.Builder()
25              .addInterceptor(logging)
26              .connectTimeout(30, TimeUnit.SECONDS)
27              .readTimeout(30, TimeUnit.SECONDS)
28              .writeTimeout(30, TimeUnit.SECONDS)
29              .build()
30      }
31
32      val tmdbApiService: TmdbApiService by lazy {
33          Retrofit.Builder()
34              .baseUrl(BASE_URL)
35              .client(okHttpClient)
36
37  .addConverterFactory(json.asConverterFactory("applicatio
    n/json".toMediaType()))
38              .build()
                .create(TmdbApiService::class.java)
39      }
40  }
```

Tabel 5. TmdbApiService.kt Modul 5

```
01  package com.example.movielist.data.remote.api
-
2
3   import
    com.example.movielist.data.remote.models.MovieListRespo
    nse
4   import retrofit2.Response
5   import retrofit2.http.GET
6   import retrofit2.http.Query
7
8   interface TmdbApiService {
9
10      @GET("movie/popular")
11      suspend fun getPopularMovies(
12          @Query("api_key") apiKey: String,
13          @Query("language") language: String = "en-US",
14          @Query("page") page: Int = 1
```

```
    ): Response<MovieListResponse>
}
```

Tabel 6.MovieDto.kt Modul 5

```
package com.example.movielist.data.remote.models

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

@Serializable
data class MovieDto(
    val adult: Boolean,
    @SerialName("backdrop_path")
    val backdropPath: String?,
    @SerialName("genre_ids")
    val genreIds: List<Int>,
    val id: Int,
    @SerialName("original_language")
    val originalLanguage: String,
    @SerialName("original_title")
    val originalTitle: String,
    val overview: String,
    val popularity: Double,
    @SerialName("poster_path")
    val posterPath: String?,
    @SerialName("release_date")
    val releaseDate: String,
    val title: String,
    val video: Boolean,
    @SerialName("vote_average")
    val voteAverage: Double,
    @SerialName("vote_count")
    val voteCount: Int
)
```

Tabel 7.MovieDtoExtension.kt

```
package com.example.movielist.data.remote.models

import com.example.movielist.domain.model.Movie
import
com.example.movielist.data.local.entities.MovieEntity

fun MovieDto.toDomainMovie(): Movie {
    return Movie(
        id = id,
```

```
9           title = title,
10          overview = overview,
11          posterPath = posterPath,
12          releaseDate = releaseDate,
13          voteAverage = voteAverage
14      )
15  }
16
17  fun MovieDto.toMovieEntity(): MovieEntity {
18      return MovieEntity(
19          id = id,
20          title = title,
21          overview = overview,
22          posterPath = posterPath,
23          releaseDate = releaseDate,
24          voteAverage = voteAverage,
25          popularity = popularity
26      )
27  }
```

Tabel 8. MovieListResponse.kt Modul 5

```
1   package com.example.movielist.data.remote.models
2
3   import kotlinx.serialization.SerialName
4   import kotlinx.serialization.Serializable
5
6   @Serializable
7   data class MovieListResponse(
8       val page: Int,
9       val results: List<MovieDto>,
10      @SerialName("total_pages")
11      val totalPages: Int,
12      @SerialName("total_results")
13      val totalResults: Int
14  )
```

Tabel 9. MovieRepository.kt Modul 5

```
1   package com.example.movielist.data.repository
2
3   import com.example.movielist.data.local.dao.MovieDao
4   import
    com.example.movielist.data.remote.api.TmdbApiService
5   import
```

```
    com.example.movielist.data.remote.models.toDomainMovie
6   import
    com.example.movielist.data.remote.models.toMovieEntity
7   import com.example.movielist.domain.model.Movie
8   import com.example.movielist.utils.Result
9   import kotlinx.coroutines.flow.Flow
10  import kotlinx.coroutines.flow.flow
11  import retrofit2.HttpException
12  import java.io.IOException
13
14  interface MovieRepository {
15      fun getPopularMovies(): Flow<Result<List<Movie>>>
16  }
17
18  class MovieRepositoryImpl(
19      private val apiService: TmdbApiService,
20      private val movieDao: MovieDao,
21      private val apiKey: String
22  ) : MovieRepository {
23
24      override fun getPopularMovies():
    Flow<Result<List<Movie>>> = flow {
25          emit(Result.Loading)
26
27          val cachedMovies = movieDao.getAllMovies().map
    { it.toDomainMovie() }
28          if (cachedMovies.isNotEmpty()) {
29              emit(Result.Success(cachedMovies))
30          }
31
32          try {
33              val response =
    apiService.getPopularMovies(apiKey = apiKey)
34              if (response.isSuccessful) {
35                  val movieDtos =
    response.body()?.results ?: emptyList()
36                  val domainMovies = movieDtos.map {
    it.toDomainMovie() }
37
38                  movieDao.clearAllMovies()
39                  movieDao.insertAllMovies(movieDtos.map
    { it.toMovieEntity() })
40
41                  emit(Result.Success(domainMovies))
                } else {
42                  emit(Result.Error(Exception("API
    Error: ${response.code()} ${response.message()}")))
```

```
            }
        } catch (e: HttpException) {
            emit(Result.Error(Exception("Network Error
(HTTP ${e.code()}): ${e.message()}")))
        } catch (e: IOException) {
            emit(Result.Error(Exception("No Internet
Connection or API Timeout: ${e.message}")))
        } catch (e: Exception) {
            emit(Result.Error(Exception("An unexpected
error occurred: ${e.localizedMessage}")))
        }
    }
}
```

Tabel 10. Movie.kt Modul 5

```
package com.example.movielist.domain.model

import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@Parcelize
data class Movie(
    val id: Int,
    val title: String,
    val overview: String,
    val posterPath: String?,
    val releaseDate: String,
    val voteAverage: Double
) : Parcelable
```

Tabel 11. GetPopularMovieUseCase.kt Modul 5

```
package com.example.movielist.domain.usecase


import com.example.movielist.domain.model.Movie
import
com.example.movielist.data.repository.MovieRepositoryIm
pl
import com.example.movielist.utils.Result
import kotlinx.coroutines.flow.Flow

class GetPopularMoviesUseCase (
    private val movieRepository: MovieRepositoryImpl
) {
```

```
11      operator fun invoke(): Flow<Result<List<Movie>>> {
12          return movieRepository.getPopularMovies()
13      }
14  }
```

Tabel 12. DetailActivity.kt Modul 5

```
01  package com.example.movielist.presentation.ui.activity
2
3   import android.os.Build
4   import android.os.Bundle
5   import android.view.MenuItem
6   import android.widget.Toast
7   import androidx.appcompat.app.AppCompatActivity
8   import com.bumptech.glide.Glide
9   import
    com.example.movielist.databinding.ActivityDetailBinding
10  import com.example.movielist.domain.model.Movie
11
12  class DetailActivity : AppCompatActivity() {
13
14      private lateinit var binding: ActivityDetailBinding
15
16      companion object {
17          const val EXTRA_MOVIE = "extra_movie"
18      }
19
20      override fun onCreate(savedInstanceState: Bundle?)
21  {
22          super.onCreate(savedInstanceState)
23          binding =
    ActivityDetailBinding.inflate(layoutInflater)
24          setContentView(binding.root)
25
26
27
28  supportActionBar?.setDisplayHomeAsUpEnabled(true)
29
30
31          val movie = if (Build.VERSION.SDK_INT >=
    Build.VERSION_CODES.TIRAMISU) {
32              intent.getParcelableExtra(EXTRA_MOVIE,
    Movie::class.java)
33          } else {
34              @Suppress("DEPRECATION")
35              intent.getParcelableExtra(EXTRA_MOVIE)
36          }
```

```
37
38          movie?.let {
39
40              supportActionBar?.title = it.title
41
42              binding.apply {
43                  tvDetailTitle.text = it.title
44                  tvDetailReleaseDate.text = "Release
    Date: ${it.releaseDate}"
45                  tvDetailVoteAverage.text = "Rating:
    ${String.format("%.1f", it.voteAverage)}"
46                  tvDetailOverview.text = it.overview
47
48                  val imageUrl =
    "https://image.tmdb.org/t/p/w500${it.posterPath}"
49                  Glide.with(this@DetailActivity)
50                      .load(imageUrl)
51                      .centerCrop()
52                      .into(ivDetailPoster)
53              }
54          } ?: run {
55              Toast.makeText(this, "Film tidak
    ditemukan.", Toast.LENGTH_SHORT).show()
56              finish()
57          }
58      }
59
60
61      override fun onOptionsItemSelected(item: MenuItem):
    Boolean {
62          if (item.itemId == android.R.id.home) {
63              onBackPressedDispatcher.onBackPressed()
64              return true
65          }
66          return super.onOptionsItemSelected(item)
67      }
68  }
```

Tabel 13. MainActivity.kt Modul 5

```
001 package com.example.movielist.presentation.ui.activity
-
2
3   import android.content.Intent
4   import android.os.Bundle
5   import android.view.View
```

```
6  import android.widget.Toast
7  import androidx.activity.viewModels
8  import androidx.appcompat.app.AppCompatActivity
9  import androidx.appcompat.app.AppCompatDelegate
10 import androidx.lifecycle.Observer
11 import androidx.recyclerview.widget.LinearLayoutManager
12 import androidx.room.Room
13 import
   com.example.movielist.data.local.MovieAppPreferences
14 import
   com.example.movielist.data.local.database.AppDatabase
15 import
   com.example.movielist.data.repository.MovieRepositoryIm
   pl
16 import
   com.example.movielist.data.remote.api.RetrofitClient
17 import
   com.example.movielist.databinding.ActivityMainBinding
18 import
   com.example.movielist.domain.usecase.GetPopularMoviesUs
   eCase
19 import
   com.example.movielist.presentation.ui.adapter.MovieAdap
   ter
20 import
   com.example.movielist.presentation.viewmodel.MovieViewM
   odel
21 import
   com.example.movielist.presentation.viewmodel.ViewModelF
   actory
22 import com.example.movielist.utils.Result
23 import com.example.movielist.R
24 class MainActivity : AppCompatActivity() {
25
26     private lateinit var binding: ActivityMainBinding
27     private lateinit var movieAdapter: MovieAdapter
28     private lateinit var movieAppPreferences:
   MovieAppPreferences
29
30     private val movieViewModel: MovieViewModel by
   viewModels {
31         val apiService = RetrofitClient.tmdbApiService
32         val database = Room.databaseBuilder(
33             applicationContext,
34             AppDatabase::class.java,
35             AppDatabase.DATABASE_NAME
36         ).build()
```

```
37          val movieDao = database.movieDao()
38          val tmdbApiKey =
"71819bfeac768c2a5b9a32b26e50cae1"
39          movieAppPreferences.saveApiKey(tmdbApiKey)
40
41          val movieRepositoryImpl =
MovieRepositoryImpl(apiService, movieDao, tmdbApiKey)
42          val getPopularMoviesUseCase =
GetPopularMoviesUseCase(movieRepositoryImpl)
43          ViewModelFactory(getPopularMoviesUseCase)
44      }
45
46      override fun onCreate(savedInstanceState: Bundle?)
47  {
48          super.onCreate(savedInstanceState)
49          binding =
ActivityMainBinding.inflate(layoutInflater)
50          setContentView(binding.root)
51
52
53  supportActionBar?.setDisplayHomeAsUpEnabled(false)
54          supportActionBar?.title = "Popular Movies"
55
56          movieAppPreferences = MovieAppPreferences(this)
57
58          setupRecyclerView()
59          observeViewModel()
60          setupDarkModeToggle()
61
62          binding.btnRetry.setOnClickListener {
63              movieViewModel.fetchPopularMovies()
64          }
65      }
66
67      private fun setupRecyclerView() {
68          movieAdapter = MovieAdapter()
69          binding.rvMovies.apply {
70              layoutManager =
LinearLayoutManager(this@MainActivity)
71              adapter = movieAdapter
72          }
73
74          movieAdapter.onItemClick = { movie ->
75              val intent = Intent(this,
DetailActivity::class.java).apply {
77                  putExtra(DetailActivity.EXTRA_MOVIE,
movie)
```

```
78              }
79              startActivity(intent)
80          }
81      }
82
83      private fun observeViewModel() {
84          movieViewModel.popularMovies.observe(this,
    Observer { result ->
85              when (result) {
86                  is Result.Loading -> {
87                      binding.progressBar.visibility =
    View.VISIBLE
88                      binding.tvError.visibility =
    View.GONE
89                      binding.btnRetry.visibility =
    View.GONE
90                      binding.rvMovies.visibility =
    View.GONE
91                  }
92                  is Result.Success -> {
93                      binding.progressBar.visibility =
    View.GONE
94                      binding.tvError.visibility =
    View.GONE
95                      binding.btnRetry.visibility =
    View.GONE
96                      binding.rvMovies.visibility =
    View.VISIBLE
97
98  movieAdapter.submitList(result.data)
99                  }
100                 is Result.Error -> {
101                     binding.progressBar.visibility =
    View.GONE
102                     binding.rvMovies.visibility =
    View.GONE
103                     binding.tvError.visibility =
    View.VISIBLE
104                     binding.btnRetry.visibility =
    View.VISIBLE
105                     binding.tvError.text = "Error:
    ${result.exception.message}"
106                     Toast.makeText(this, "Error:
    ${result.exception.message}", Toast.LENGTH_LONG).show()
107                 }
108             }
109         })
```

```
110     }
111
112     private fun setupDarkModeToggle() {
113         val isDarkMode =
    movieAppPreferences.getDarkModeState()
114         applyTheme(isDarkMode)
115         updateToggleIcon(isDarkMode)
116
117         binding.btnDarkModeToggle.setOnClickListener
118             val currentMode =
    movieAppPreferences.getDarkModeState()
119             val newMode = !currentMode
120
    movieAppPreferences.saveDarkModeState(newMode)
121             applyTheme(newMode)
122         }
123     }
124
125     private fun applyTheme(isDarkMode: Boolean) {
126         if (isDarkMode) {
127
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate
128 .MODE_NIGHT_YES)
129         } else {
130
131 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate
132 .MODE_NIGHT_NO)
133         }
134     }
135
136     private fun updateToggleIcon(isDarkMode: Boolean) {
137         if (isDarkMode) {
    binding.btnDarkModeToggle.setImageResource(R.drawable.i
    c_light_bulb_off)
138         } else {
139 binding.btnDarkModeToggle.setImageResource(R.drawable.i
    c_light_bulb_on)
140         }
141     }
142 }
```

Tabel 14. MovieAdapter.kt Modul 5

```
01  package com.example.movielist.presentation.ui.adapter
-
2
3   import android.view.LayoutInflater
4   import android.view.ViewGroup
5   import androidx.recyclerview.widget.DiffUtil
6   import androidx.recyclerview.widget.ListAdapter
7   import androidx.recyclerview.widget.RecyclerView
8   import com.bumptech.glide.Glide
9   import
    com.example.movielist.databinding.ItemMovieBinding
10  import com.example.movielist.domain.model.Movie
11
12  class MovieAdapter : ListAdapter<Movie,
13  MovieAdapter.MovieViewHolder>(MovieDiffCallback()) {
14
15      var onItemClick: ((Movie) -> Unit)? = null
16
17      override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): MovieViewHolder {
18          val binding =
    ItemMovieBinding.inflate(LayoutInflater.from(parent.con
    text), parent, false)
19          return MovieViewHolder(binding)
20      }
21
22      override fun onBindViewHolder(holder:
    MovieViewHolder, position: Int) {
23          val movie = getItem(position)
24          holder.bind(movie)
25      }
26
27      inner class MovieViewHolder(private val binding:
    ItemMovieBinding) :
28          RecyclerView.ViewHolder(binding.root) {
29
30          init {
31              binding.btnDetail.setOnClickListener {
32
    onItemClick?.invoke(getItem(adapterPosition))
33              }
34          }
35
36          fun bind(movie: Movie) {
37              binding.apply {
38                  tvMovieTitle.text = movie.title
39                  tvReleaseDate.text = "Release Date:
```

```
    ${movie.releaseDate}"
40                  tvVoteAverage.text = "Rating:
    ${String.format("%.1f", movie.voteAverage)}"
41                  tvOverview.text = movie.overview
42
43                  val imageUrl =
    "https://image.tmdb.org/t/p/w500${movie.posterPath}"
44
45                  Glide.with(itemView.context)
46                      .load(imageUrl)
47                      .centerCrop()
48                      .into(ivPoster)
49              }
50          }
51      }
52
53      class MovieDiffCallback :
    DiffUtil.ItemCallback<Movie>() {
54          override fun areItemsTheSame(oldItem: Movie,
55  newItem: Movie): Boolean {
                return oldItem.id == newItem.id
56          }
57
58          override fun areContentsTheSame(oldItem: Movie,
    newItem: Movie): Boolean {
59              return oldItem == newItem
            }
60      }
61  }
```

Tabel 15. MovieViewModel.kt

```
01  package com.example.movielist.presentation.viewmodel
-
2
3   import androidx.lifecycle.LiveData
4   import androidx.lifecycle.MutableLiveData
5   import androidx.lifecycle.ViewModel
6   import androidx.lifecycle.viewModelScope
7   import com.example.movielist.domain.model.Movie
8   import
    com.example.movielist.domain.usecase.GetPopularMoviesUs
    eCase
9   import com.example.movielist.utils.Result
10  import kotlinx.coroutines.launch
```

```
11  class MovieViewModel (
12      private val getPopularMoviesUseCase:
    GetPopularMoviesUseCase
13  ) : ViewModel() {
14
15      private val _popularMovies =
    MutableLiveData<Result<List<Movie>>>()
16      val popularMovies: LiveData<Result<List<Movie>>> =
    _popularMovies
17
18      init {
19          fetchPopularMovies()
20      }
21
22      fun fetchPopularMovies() {
23          viewModelScope.launch {
24              getPopularMoviesUseCase().collect { result
25  ->
26                  _popularMovies.value = result
27              }
28          }
29      }
30  }
```

Tabel 16. ViewModelFactory.kt Modul 5

```
01-  package com.example.movielist.presentation.viewmodel
2
3
4    import androidx.lifecycle.ViewModel
5    import androidx.lifecycle.ViewModelProvider
     import
     com.example.movielist.domain.usecase.GetPopularMoviesUs
6    eCase
7
8    class ViewModelFactory(
         private val getPopularMoviesUseCase:
9    GetPopularMoviesUseCase
10   ) : ViewModelProvider.Factory {
11
         override fun <T : ViewModel> create(modelClass:
12   Class<T>): T {
             if
13   (modelClass.isAssignableFrom(MovieViewModel::class.java
14   )) {
15               @Suppress("UNCHECKED_CAST")
```

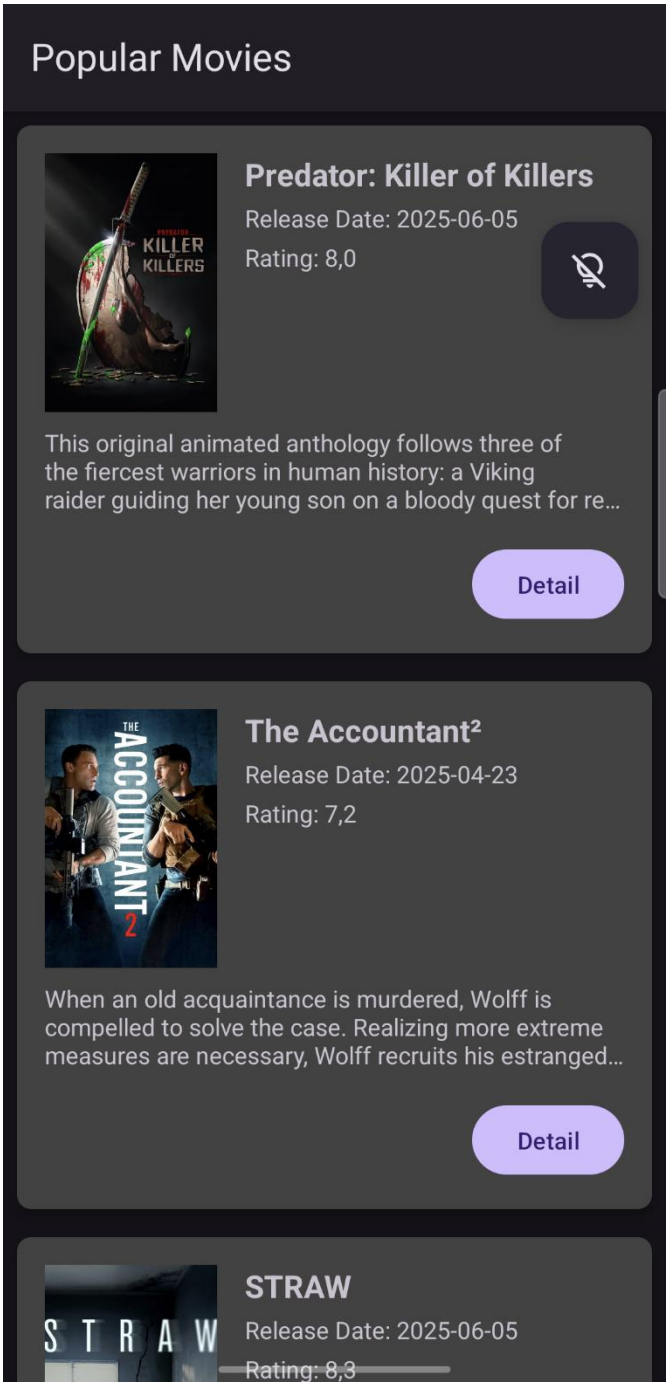
```
            return
16  MovieViewModel(getPopularMoviesUseCase) as T
17          }
            throw IllegalArgumentException("Unknown
18  ViewModel class")
19      }
    }
```

Tabel 17. Result.kt Modul 5

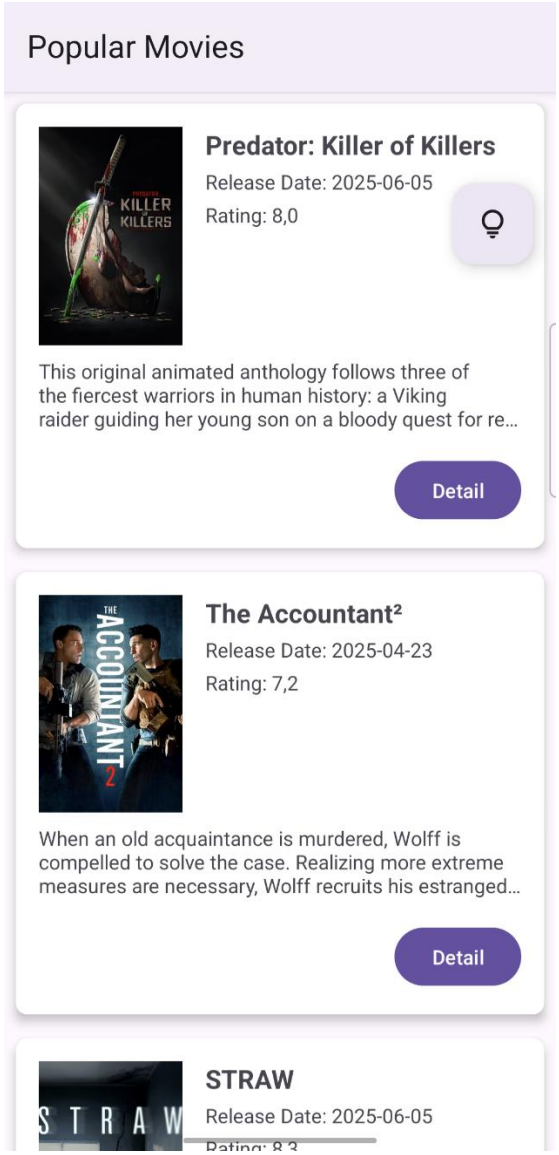
```
1   package com.example.movielist.utils
2
3   sealed class Result<out T> {
4     object Loading : Result<Nothing>()
5     data class Success<out T>(val data: T) : Result<T>()
      data class Error(val exception: Exception) : Result<Nothing>()
6   }

7
```

B. Output Produk

Gambar 1. Ouput Modul 5



Gambar 2. Ouput Modul 5



Gambar 3. Ouput Modul 5

C. Pembahasan

MovieAppPreferences.kt:

Pada baris 1, package com.example.movielist.data.local mendefinisikan *package* file ini di lapisan data lokal. Pada baris 3-4, import class Context dan SharedPreferences yang diperlukan untuk mengakses layanan sistem dan API penyimpanan data ringan.

Pada baris 6, class MovieAppPreferences(context: Context) mendefinisikan class MovieAppPreferences yang bertanggung jawab untuk mengelola akses ke SharedPreferences aplikasi. Konstruktornya menerima objek Context untuk inisialisasi. Pada baris 8-9, private val sharedPreferences: SharedPreferences = context.getSharedPreferences("tmdb_app_prefs", Context.MODE_PRIVATE)

menginisialisasi instance SharedPreferences dengan nama file "tmdb_app_prefs" dan mode MODE_PRIVATE, berarti data hanya dapat diakses oleh aplikasi ini.

Pada baris 11, companion object { ... } mendefinisikan objek pendamping yang berisi konstanta kunci untuk data yang disimpan. Pada baris 12, private const val KEY_API_KEY = "api_key" mendefinisikan konstanta kunci untuk menyimpan API key. Pada baris 13, private const val KEY_DARK_MODE = "dark_mode" mendefinisikan konstanta kunci untuk menyimpan preferensi mode gelap.

Pada baris 15, fun saveApiKey(apiKey: String) adalah fungsi untuk menyimpan API key. Pada baris 16, sharedPreferences.edit().putString(KEY_API_KEY, apiKey).apply() mengambil editor SharedPreferences, menyimpan string API key, dan menerapkan perubahan secara asinkron. Pada baris 19, fun getApiKey(): String? adalah fungsi untuk mengambil API key yang tersimpan, mengembalikan null jika tidak ditemukan.

Pada baris 23, fun saveDarkModeState(isDarkMode: Boolean) adalah fungsi untuk menyimpan status mode gelap. Pada baris 24, sharedPreferences.edit().putBoolean(KEY_DARK_MODE, isDarkMode).apply() menyimpan status boolean mode gelap. Pada baris 27, fun getDarkModeState(): Boolean adalah fungsi untuk mengambil status mode gelap yang tersimpan, mengembalikan false secara default.

MovieDao.kt:

Pada baris 1, package com.example.movielist.data.local.dao mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk Data Access Object (DAO) film. Pada baris 3-7, import anotasi Room (@Dao, @Insert, @OnConflictStrategy, @Query) dan class MovieEntity yang akan berinteraksi dengan DAO ini.

Pada baris 9, @Dao menandai *interface* MovieDao sebagai Data Access Object untuk Room, yang akan secara otomatis diimplementasikan oleh Room. Pada baris 10, interface MovieDao mendeklarasikan antarmuka ini.

Pada baris 11, @Insert(onConflict = OnConflictStrategy.REPLACE) menandai fungsi ini sebagai operasi penyisipan. OnConflictStrategy.REPLACE akan mengganti data yang sudah ada jika ada konflik primary key. Pada baris 12, suspend fun insertAllMovies(movies: List<MovieEntity>) mendefinisikan fungsi suspend untuk menyisipkan daftar MovieEntity ke database. suspend menunjukkan bahwa ini adalah fungsi *coroutine* yang dapat dihentikan sementara.

Pada baris 14, @Query("SELECT * FROM movies ORDER BY popularity DESC") menandai fungsi ini dengan kueri SQL kustom untuk mengambil semua film dari tabel "movies" yang diurutkan berdasarkan popularitas secara descending. Pada baris 15, suspend fun getAllMovies(): List<MovieEntity> mendefinisikan fungsi suspend untuk mengambil semua MovieEntity dari database.

Pada baris 17, @Query("DELETE FROM movies") menandai fungsi ini dengan kueri SQL untuk menghapus semua entri dari tabel "movies". Pada baris 18, suspend fun clearAllMovies() mendefinisikan fungsi suspend untuk menghapus semua data film dari cache.

AppDatabase.kt:

Pada baris 1, package com.example.movielist.data.local.database mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk class database Room. Pada baris 3-7, import anotasi Room (@Database, @RoomDatabase, @TypeConverters), GenreConverter, MovieDao, dan MovieEntity.

Pada baris 9, @Database(entities = [MovieEntity::class], version = 2, exportSchema = false) menandai class AppDatabase sebagai database Room. entities = [MovieEntity::class] mendaftarkan MovieEntity sebagai tabel dalam database. version = 2 menetapkan versi database, yang perlu dinaikkan setiap kali skema database berubah (misalnya, penambahan kolom baru). exportSchema = false menonaktifkan ekspor skema database ke file, yang cocok untuk pengembangan.

Pada baris 10, @TypeConverters(GenreConverter::class) mendaftarkan GenreConverter sebagai *TypeConverter* untuk database ini, memungkinkan Room untuk menyimpan dan mengambil tipe data kompleks seperti List<Int> yang tidak didukung secara *native* oleh SQLite.

Pada baris 11, abstract class AppDatabase : RoomDatabase() mendefinisikan class abstrak AppDatabase yang mewarisi dari RoomDatabase. Room akan mengimplementasikan class ini secara otomatis. Pada baris 12, abstract fun movieDao(): MovieDao mendefinisikan fungsi abstrak untuk mendapatkan instance MovieDao, yang merupakan cara aplikasi berinteraksi dengan database.

Pada baris 14, companion object { ... } mendefinisikan objek pendamping. Pada baris 15, const val DATABASE_NAME = "tmdb_app_db" mendefinisikan konstanta untuk nama file database.

MovieEntity.kt:

Pada baris 1, package com.example.movielist.data.local.entities mendefinisikan *package* ini sebagai bagian dari lapisan data lokal untuk entitas Room. Pada baris 3-5, import anotasi Room (@Entity, @PrimaryKey) dan class Movie dari lapisan domain.

Pada baris 7, @Entity(tableName = "movies") menandai data class MovieEntity sebagai tabel dalam database Room dengan nama "movies". Pada baris 8, data class MovieEntity(...) mendefinisikan data class MovieEntity, yang merupakan representasi satu baris dalam tabel movies. Properti-propertinya (id, title, overview, dll.) menjadi kolom-kolom tabel.

Pada baris 9, @PrimaryKey val id: Int menandai id sebagai primary key, memastikan setiap film memiliki identifikasi unik di database. Pada baris 16, val genreIds: List<Int> adalah kolom baru yang menyimpan daftar ID genre; ini memerlukan TypeConverter karena List<Int> bukan tipe data yang didukung secara *native* oleh SQLite.

Pada baris 19, fun toDomainMovie(): Movie mendefinisikan fungsi di dalam MovieEntity yang mengonversi objek MovieEntity dari database menjadi Movie model domain. Ini adalah bagian penting dari pemetaan antar lapisan.

Pada baris 29, companion object { ... } mendefinisikan objek pendamping. Pada baris 30, fun fromDomainMovie(movie: Movie, popularity: Double, genreIds: List<Int>): MovieEntity adalah fungsi *factory* di companion object yang digunakan untuk membuat MovieEntity dari Movie domain model dan properti tambahan seperti popularity dan genreIds yang hanya ada di MovieEntity untuk tujuan penyimpanan.

RetrofitClient.kt:

Pada baris 1, package com.example.movielist.data.remote.api mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk klien API. Pada baris 3-9, import berbagai class dan fungsi yang diperlukan untuk konfigurasi Retrofit, KotlinX Serialization, OkHttp, dan penanganan waktu.

Pada baris 11, object RetrofitClient mendeklarasikan objek *singleton* RetrofitClient, yang berarti hanya ada satu instance dari objek ini di seluruh aplikasi. Ini adalah praktik umum untuk klien HTTP.

Pada baris 13, private const val BASE_URL = "https://api.themoviedb.org/3/" mendefinisikan konstanta URL dasar untuk semua permintaan ke TMDB API.

Pada baris 15-18, private val json = Json { ... } menginisialisasi instance Json dari KotlinX Serialization dengan konfigurasi kustom. ignoreUnknownKeys = true mengonfigurasi parser untuk mengabaikan kunci JSON yang tidak ada di model data Kotlin Anda, mencegah *crash* jika ada perubahan di API. prettyPrint = true digunakan untuk memformat output JSON agar mudah dibaca (berguna untuk *debugging*).

Pada baris 20-29, private val okHttpClient: OkHttpClient by lazy { ... } mendeklarasikan instance OkHttpClient secara *lazy* (akan dibuat saat pertama kali diakses). Di dalamnya, HttpLoggingInterceptor ditambahkan dengan level BODY untuk menampilkan detail permintaan dan respons HTTP di Logcat (berguna untuk *debugging*). Berbagai *timeout* (koneksi, baca, tulis) juga dikonfigurasi untuk mencegah permintaan menggantung terlalu lama.

Pada baris 32-38, val tmdbApiService: TmdbApiService by lazy { ... } mendeklarasikan instance TmdbApiService secara *lazy*. Retrofit.Builder() digunakan untuk membangun instance Retrofit dengan URL dasar, OkHttpClient yang sudah dikonfigurasi, dan KotlinX Serialization Converter Factory (json.asConverterFactory(...)) untuk

mengonversi JSON menjadi objek Kotlin. Terakhir, .create(TmdbApiService::class.java) membuat implementasi TmdbApiService dari antarmuka yang didefinisikan.

TmdbApiService.kt:

Pada baris 1, package com.example.movielist.data.remote.api mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk antarmuka API. Pada baris 3-6, import class model respons MovieListResponse dan anotasi Retrofit (@GET, @Query, Response).

Pada baris 8, interface TmdbApiService mendeklarasikan antarmuka TmdbApiService. Ini mendefinisikan kontrak untuk berinteraksi dengan API TMDB.

Pada baris 10, @GET("movie/popular") menandai fungsi getPopularMovies() untuk melakukan permintaan HTTP GET ke endpoint "movie/popular" relatif terhadap URL dasar yang dikonfigurasi di RetrofitClient. Pada baris 11, suspend fun getPopularMovies(...) mendefinisikan fungsi suspend untuk mengambil daftar film populer. Kata kunci suspend berarti fungsi ini dapat dipanggil dari *coroutine*.

Pada baris 12, @Query("api_key") apiKey: String mendeklarasikan parameter kueri URL "api_key" yang wajib diisi. Pada baris 13, @Query("language") language: String = "en-US" menambahkan parameter kueri "language" dengan nilai default "en-US". Pada baris 14, @Query("page") page: Int = 1 menambahkan parameter kueri "page" untuk pagination dengan nilai default 1.

Pada baris 15, : Response<MovieListResponse> menentukan bahwa fungsi ini akan mengembalikan objek Response dari Retrofit yang membungkus MovieListResponse, yang berisi daftar film.

MovieDto.kt:

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk model data. Pada baris 3-4, import anotasi SerialName dan Serializable dari KotlinX Serialization.

Pada baris 6, @Serializable menandai data class MovieDto agar dapat diubah menjadi/dari format JSON oleh KotlinX Serialization. Pada baris 7, data class MovieDto(...) mendefinisikan data class MovieDto, yang berfungsi sebagai Data Transfer Object (DTO) untuk film. Struktur propertinya secara langsung memetakan struktur JSON yang diterima dari TMDB API.

Pada baris 8-28, setiap properti seperti adult, backdropPath, genreIds, id, originalLanguage, originalTitle, overview, popularity, posterPath, releaseDate, title, video, voteAverage, dan voteCount didefinisikan. Anotasi @SerialName("nama_json") digunakan untuk properti di mana nama Kotlin berbeda dari nama kunci di JSON (misalnya, backdropPath untuk backdrop_path). Tanda ? setelah tipe data (misalnya String?) menunjukkan bahwa properti tersebut bisa bernilai null.

MovieListResponse.kt:

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini sebagai bagian dari lapisan data remote untuk model data. Pada baris 3-4, import anotasi SerialName dan Serializable dari KotlinX Serialization.

Pada baris 6, @Serializable menandai data class MovieListResponse agar dapat diubah menjadi/dari format JSON. Pada baris 7, data class MovieListResponse(...) mendefinisikan data class MovieListResponse, yang merepresentasikan struktur respons keseluruhan ketika meminta daftar film populer dari TMDB API.

Pada baris 8, val page: Int mendefinisikan properti untuk nomor halaman saat ini. Pada baris 9, val results: List<MovieDto> mendefinisikan properti results, yang merupakan daftar aktual dari objek MovieDto (daftar film).

Pada baris 10, @SerialName("total_pages") val totalPages: Int memetakan kunci JSON "total_pages" ke properti totalPages (jumlah total halaman hasil). Pada baris 12, @SerialName("total_results") val totalResults: Int memetakan kunci JSON "total_results" ke properti totalResults (jumlah total film yang ditemukan).

MovieDtoExtension.kt:

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan *package* ini untuk file ekstensi model data. Pada baris 3-4, import class Movie dari domain dan MovieEntity dari lapisan data lokal.

Pada baris 6, fun MovieDto.toDomainMovie(): Movie mendefinisikan fungsi ekstensi untuk MovieDto. Fungsi ini mengonversi sebuah objek MovieDto (yang berasal dari API) menjadi Movie model domain yang bersih. Ini adalah bagian penting dari pemetaan antar lapisan data dan domain.

Pada baris 15, fun MovieDto.toMovieEntity(): MovieEntity mendefinisikan fungsi ekstensi lain untuk MovieDto. Fungsi ini mengonversi objek MovieDto dari API menjadi MovieEntity yang dapat disimpan di Room Database. Fungsi ini menggunakan MovieEntity.fromDomainMovie() untuk melakukan konversi, meneruskan properti yang relevan termasuk popularity dan genreIds yang spesifik untuk MovieEntity.

MovieRepository.kt:

Pada baris 1, package com.example.movielist.data.repository mendefinisikan *package* untuk repository, bagian dari lapisan data. Pada baris 3-10, import berbagai class dan interface yang dibutuhkan untuk fungsionalitas repository (DAO, API service, model, Flow, Result).

Pada baris 12, interface MovieRepository mendeklarasikan antarmuka MovieRepository. Ini mendefinisikan kontrak tentang bagaimana data film akan disediakan, tanpa mengungkapkan detail implementasinya. Pada baris 13, fun getPopularMovies(): Flow<Result<List<Movie>>> adalah satu-satunya fungsi dalam antarmuka, yang akan mengembalikan Flow yang membungkus Result dari daftar Movie.

Pada baris 16, class MovieRepositoryImpl(...) : MovieRepository mendefinisikan class MovieRepositoryImpl, yang merupakan implementasi konkret dari antarmuka MovieRepository. Konstruktornya menerima dependensi TmdbApiService (untuk jaringan) dan MovieDao (untuk database lokal).

Pada baris 20, override fun getPopularMovies(): Flow<Result<List<Movie>>> = flow { ... } mengimplementasikan fungsi dari antarmuka. Ini adalah inti dari strategi *caching* dan pengambilan data. Pada baris 21, emit(Result.Loading) segera memancarkan status Loading ke Flow, memberi tahu UI bahwa proses pengambilan data telah dimulai.

Pada baris 23, val cachedMovies = movieDao.getAllMovies().map { it.toDomainMovie() } mencoba mengambil data film yang sudah ada di cache Room Database. Data ini kemudian dipetakan ke domain model Movie. Pada baris 24-26, jika ada data di cache, data tersebut segera dipancarkan sebagai Result.Success, memastikan aplikasi dapat menampilkan data dengan cepat.

Pada baris 28-47, blok try { ... } catch (...) { ... } menangani pengambilan data dari jaringan dan berbagai jenis error. Pada baris 29, val response = apiService.getPopularMovies(apiKey = apiKey) melakukan panggilan API ke TMDB. Pada baris 30-32, jika panggilan API berhasil, DTO film diambil dan dipetakan ke domain model.

Pada baris 34, movieDao.clearAllMovies() menghapus data lama dari cache Room. Pada baris 35, movieDao.insertAllMovies(movieDtos.map { it.toMovieEntity() })

menyisipkan data film terbaru dari API ke dalam cache Room. Pada baris 37, emit(Result.Success(domainMovies)) memancarkan data terbaru ke Flow untuk diperbarui di UI. Baris 39-47 adalah blok catch yang menangani HttpException (kesalahan API), IOException (masalah koneksi/timeout), dan Exception umum, memancarkan Result.Error yang sesuai.

Movie.kt:

Pada baris 1, package com.example.movielist.domain.model mendefinisikan *package* ini sebagai bagian dari lapisan domain untuk model data. Pada baris 3-4, import antarmuka Parcelable dari Android dan anotasi @Parcelize dari plugin Kotlin Parcelize.

Pada baris 6, @Parcelize adalah anotasi yang secara otomatis menghasilkan implementasi kode *boilerplate* Parcelable untuk data class Movie, sehingga memungkinkan objek ini untuk dikirim antar komponen Android (seperti antar Activity) secara efisien tanpa harus menulis kode manual.

Pada baris 7, data class Movie(...) : Parcelable mendefinisikan data class Movie. Ini adalah model domain yang bersih, yang berarti ia tidak bergantung pada detail implementasi API (DTO) atau database (Entity). Ia hanya berisi data yang relevan untuk logika bisnis dan presentasi. : Parcelable menunjukkan bahwa class ini mengimplementasikan antarmuka Parcelable.

Pada baris 8-13, properti-properti seperti id, title, overview, posterPath, releaseDate, dan voteAverage adalah atribut-atribut film yang relevan di lapisan domain aplikasi.

GetPopularMoviesUseCase.kt:

Pada baris 1, package com.example.movielist.domain.usecase mendefinisikan *package* ini sebagai bagian dari lapisan domain untuk *use case*. Pada baris 3-6, import class yang dibutuhkan (Movie model domain, MovieRepositoryImpl implementasi repository, Result, Flow).

Pada baris 8, class GetPopularMoviesUseCase(...) mendefinisikan *use case* GetPopularMoviesUseCase. *Use case* ini mengkapsulasi logika bisnis spesifik untuk "mendapatkan daftar film populer".

Pada baris 9, private val movieRepository: MovieRepositoryImpl mendeklarasikan dependensi pada implementasi repository (MovieRepositoryImpl). Dalam arsitektur Clean Architecture yang lebih ketat, *use case* seharusnya bergantung pada antarmuka repository (yang berada di lapisan domain), tetapi di sini disesuaikan dengan keputusan untuk menggabungkan antarmuka dan implementasi repository.

Pada baris 11, operator fun invoke(): Flow<Result<List<Movie>>> adalah fungsi operator invoke. Ini memungkinkan instance dari GetPopularMoviesUseCase dipanggil sebagai fungsi (misalnya getPopularMoviesUseCase()) alih-alih getPopularMoviesUseCase.invoke(). Fungsi ini mengembalikan Flow yang membungkus Result dari daftar Movie. Pada baris 12, return movieRepository.getPopularMovies() memanggil fungsi getPopularMovies() dari repository untuk mendapatkan data, dan mengembalikan Flow hasilnya. *Use case* ini sendiri tidak memiliki logika kompleks lain selain mendelegasikan tugas ke repository.

MainActivity.kt:

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk Activity utama. Pada baris 3-21, import berbagai class dan interface yang dibutuhkan untuk fungsionalitas Activity.

Pada baris 23, class MainActivity : AppCompatActivity() mendefinisikan class MainActivity sebagai titik masuk utama aplikasi. Pada baris 26-28, private lateinit var binding: ActivityMainBinding, private lateinit var movieAdapter: MovieAdapter, dan private lateinit var movieAppPreferences: MovieAppPreferences mendeklarasikan variabel untuk View Binding, adapter RecyclerView, dan preferensi aplikasi.

Pada baris 30-42, private val movieViewModel: MovieViewModel by viewModels { ... } mendeklarasikan dan menginisialisasi MovieViewModel. Di blok ini, semua dependensi ViewModel (API service, database Room, DAO, API key, repository, use case) diinjeksi secara manual ke ViewModelFactory.

Pada baris 44, override fun onCreate(savedInstanceState: Bundle?) adalah metode *lifecycle* yang dipanggil saat Activity pertama kali dibuat. Pada baris 46-47, binding = ActivityMainBinding.inflate(layoutInflater) dan setContentView(binding.root)

menginisialisasi View Binding dan mengatur layout Activity. Pada baris 49, supportActionBar?.setDisplayHomeAsUpEnabled(false) menonaktifkan tombol kembali di ActionBar (karena ini adalah layar utama). Pada baris 50, supportActionBar?.title = "Popular Movies" mengatur judul ActionBar. Pada baris 52, movieAppPreferences = MovieAppPreferences(this) menginisialisasi MovieAppPreferences.

Pada baris 54-56, setupRecyclerView(), observeViewModel(), dan setupDarkModeToggle() dipanggil untuk menyiapkan UI, mengamati data, dan mengelola mode gelap. Pada baris 58-60, binding.btnRetry.setOnClickListener { ... } mengatur *listener* klik untuk tombol "Retry" yang akan memanggil movieViewModel.fetchPopularMovies() untuk memuat ulang data.

Pada baris 63, private fun setupRecyclerView() menyiapkan RecyclerView dengan LinearLayoutManager dan MovieAdapter. Pada baris 70-74, movieAdapter.onItemClick = { movie -> ... } mengatur *callback* untuk item adapter, yang akan meluncurkan DetailActivity dan meneruskan objek Movie yang dipilih.

Pada baris 77, private fun observeViewModel() mengamati popularMovies LiveData dari movieViewModel. Pada baris 78-95, blok when (result) { ... } memperbarui UI berdasarkan Result state (Loading, Success, Error): menampilkan ProgressBar saat loading, menampilkan RecyclerView dengan data saat sukses, dan menampilkan pesan error serta tombol retry saat terjadi error.

Pada baris 97, private fun setupDarkModeToggle() mengelola fungsionalitas mode gelap. Pada baris 99, val isDarkMode = movieAppPreferences.getDarkModeState() membaca status mode gelap dari preferensi. Pada baris 100, applyTheme(isDarkMode) menerapkan tema yang sesuai, dan updateToggleIcon(isDarkMode) memperbarui ikon tombol. Pada baris 103-107, binding.btnDarkModeToggle.setOnClickListener { ... } menangani klik pada tombol mode gelap, membalik status mode, menyimpan ke preferensi, dan menerapkan tema baru.

Pada baris 110, private fun applyTheme(isDarkMode: Boolean) adalah fungsi untuk menerapkan tema. Pada baris 111-115, AppCompatDelegate.setDefaultNightMode()

digunakan untuk mengalihkan tema aplikasi antara mode siang dan malam. Pemanggilan ini akan menyebabkan Activity dibuat ulang.

Pada baris 117, private fun updateToggleIcon(isDarkMode: Boolean) adalah fungsi untuk memperbarui ikon pada tombol mode gelap (ImageButton) berdasarkan status isDarkMode (lampu mati untuk gelap, lampu nyala untuk terang).

DetailActivity.kt:

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk Activity detail. Pada baris 3-9, import berbagai class yang dibutuhkan untuk fungsionalitas Activity.

Pada baris 11, class DetailActivity : AppCompatActivity() mendefinisikan class DetailActivity, yang bertanggung jawab untuk menampilkan detail film. Pada baris 14, private lateinit var binding: ActivityDetailBinding mendeklarasikan variabel binding untuk View Binding.

Pada baris 16, companion object { ... } mendefinisikan objek pendamping. Pada baris 17, const val EXTRA_MOVIE = "extra_movie" mendefinisikan konstanta kunci yang digunakan untuk meneruskan objek Movie melalui Intent.

Pada baris 20, override fun onCreate(savedInstanceState: Bundle?) adalah metode *lifecycle* yang dipanggil saat Activity pertama kali dibuat. Pada baris 22-23, binding = ActivityDetailBinding.inflate(layoutInflater) dan setContentView(binding.root) menginisialisasi View Binding dan mengatur layout Activity. Pada baris 25, supportActionBar?.setDisplayHomeAsUpEnabled(true) mengaktifkan tombol kembali (panah ke kiri) di ActionBar Activity ini.

Pada baris 27-31, val movie = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) { ... } else { ... } mengambil objek Movie yang diteruskan dari MainActivity melalui Intent.getParcelableExtra(), dengan penanganan untuk API level yang berbeda.

Pada baris 33-49, blok movie?.let { ... } akan dieksekusi hanya jika objek movie berhasil diterima dan tidak null. Di dalamnya, pada baris 34, supportActionBar?.title = it.title

mengatur judul ActionBar dengan judul film. Pada baris 36-40, data film (title, releaseDate, voteAverage, overview) ditampilkan ke TextView yang sesuai menggunakan binding. Pada baris 42, val imageUrl = "https://image.tmdb.org/t/p/w500${it.posterPath}" membangun URL lengkap untuk gambar poster film. Pada baris 43-46, Glide.with(this@DetailActivity).load(imageUrl).centerCrop().into(ivDetailPoster) menggunakan Glide untuk memuat gambar poster. Jika movie null, pada baris 49-51, Toast akan ditampilkan dan Activity akan ditutup.

Pada baris 54, override fun onOptionsItemSelected(item: MenuItem): Boolean adalah metode *callback* yang dipanggil saat item di ActionBar diklik. Pada baris 55-58, if (item.itemId == android.R.id.home) memeriksa apakah item yang diklik adalah tombol kembali (android.R.id.home), dan jika ya, onBackPressedDispatcher.onBackPressed() dipanggil untuk mensimulasikan penekanan tombol kembali dan mengakhiri Activity.

MovieAdapter.kt:

Pada baris 1, package com.example.movielist.presentation.ui.adapter mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk adapter RecyclerView. Pada baris 3-8, import berbagai class yang dibutuhkan untuk fungsionalitas adapter (LayoutInflater, ViewGroup, DiffUtil, ListAdapter, RecyclerView, Glide, View Binding, Movie model domain).

Pada baris 10, class MovieAdapter : ListAdapter<Movie, MovieAdapter.MovieViewHolder>(MovieDiffCallback()) mendefinisikan MovieAdapter. Ini adalah turunan dari ListAdapter, sebuah jenis adapter RecyclerView yang sangat efisien dalam memperbarui daftar item karena menggunakan DiffUtil untuk menghitung perbedaan antar daftar. Ia dikonfigurasi untuk menampilkan objek Movie dan menggunakan MovieAdapter.MovieViewHolder. MovieDiffCallback() adalah *callback* yang digunakan oleh DiffUtil.

Pada baris 12, var onItemClick: ((Movie) -> Unit)? = null mendeklarasikan properti *lambda* nullable bernama onItemClick. Properti ini berfungsi sebagai *callback* yang dapat diatur dari MainActivity untuk merespons klik pada tombol "Detail" di setiap item daftar, meneruskan objek Movie yang diklik.

Pada baris 14, override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder adalah metode *callback* yang dipanggil ketika RecyclerView membutuhkan ViewHolder baru. Di dalamnya, ItemMovieBinding.inflate() digunakan untuk meng-*inflate* layout item_movie.xml dan membuat MovieViewHolder baru.

Pada baris 19, override fun onBindViewHolder(holder: MovieViewHolder, position: Int) adalah metode yang dipanggil untuk menampilkan data pada posisi tertentu. Ia mengambil objek Movie dari daftar menggunakan getItem(position) dan memanggil holder.bind(movie) untuk mengisi tampilan.

Pada baris 24, inner class MovieViewHolder(private val binding: ItemMovieBinding) : RecyclerView.ViewHolder(binding.root) mendefinisikan *inner class* MovieViewHolder yang berfungsi sebagai *container* untuk tampilan setiap item daftar. binding menyediakan akses mudah ke elemen UI dari item_movie.xml.

Pada baris 27, init { binding.btnDetail.setOnClickListener { ... } } adalah blok inisialisasi untuk MovieViewHolder. Di sinilah OnClickListener untuk btnDetail diatur. Ketika tombol diklik, onItemClick?.invoke(getItem(adapterPosition)) dipanggil, yang memicu *callback* di MainActivity dengan objek Movie yang sesuai.

Pada baris 32, fun bind(movie: Movie) adalah fungsi di dalam MovieViewHolder yang bertanggung jawab untuk mengisi elemen-elemen UI dengan data dari objek Movie. Pada baris 33-37, binding.apply { ... } digunakan untuk mengatur teks tvMovieTitle, tvReleaseDate, tvVoteAverage, dan tvOverview dari properti objek movie. Pada baris 39, val imageUrl = "https://image.tmdb.org/t/p/w500${movie.posterPath}" membangun URL lengkap untuk gambar poster. Pada baris 40-43, Glide.with(itemView.context).load(imageUrl).centerCrop().into(ivPoster) menggunakan Glide untuk memuat gambar poster ke ImageView.

Pada baris 46, class MovieDiffCallback : DiffUtil.ItemCallback<Movie>() mendefinisikan *custom* DiffUtil.ItemCallback. Pada baris 47, override fun areItemsTheSame(oldItem: Movie, newItem: Movie): Boolean membandingkan dua item untuk melihat apakah mereka adalah objek yang sama (berdasarkan id). Pada baris 50,

override fun areContentsTheSame(oldItem: Movie, newItem: Movie): Boolean membandingkan konten dari dua item yang sama persis untuk mendeteksi perubahan data.

MovieViewModel.kt:

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk ViewModel. Pada baris 3-9, import berbagai class dan interface yang dibutuhkan (LiveData, ViewModel, viewModelScope, Movie model domain, GetPopularMoviesUseCase, Result, *coroutine* launch).

Pada baris 11, class MovieViewModel(...) : ViewModel() mendefinisikan MovieViewModel, yang merupakan turunan dari androidx.lifecycle.ViewModel. ViewModel bertanggung jawab untuk menyiapkan dan mengelola data yang terkait dengan UI, memastikan data tetap ada saat konfigurasi perangkat berubah (misalnya, rotasi layar) dan melepaskan sumber daya saat tidak lagi dibutuhkan.

Pada baris 12, private val getPopularMoviesUseCase: GetPopularMoviesUseCase mendeklarasikan dependensi pada GetPopularMoviesUseCase. MovieViewModel tidak berinteraksi langsung dengan repository atau API, tetapi mendelegasikan semua logika bisnis ke *use case*.

Pada baris 15, private val _popularMovies = MutableLiveData<Result<List<Movie>>>() mendeklarasikan MutableLiveData private. Ini adalah LiveData yang dapat diubah nilainya dan akan menampung hasil pengambilan data film (dibungkus dalam Result yang berisi daftar Movie). Pada baris 16, val popularMovies: LiveData<Result<List<Movie>>> = _popularMovies mengekspos versi LiveData yang tidak dapat diubah (immutable) ke UI, yang akan mengamatinya untuk mendapatkan pembaruan data.

Pada baris 18, init { fetchPopularMovies() } adalah blok inisialisasi yang akan dipanggil saat instance MovieViewModel pertama kali dibuat. Ini secara otomatis memicu proses pengambilan data film.

Pada baris 21, fun fetchPopularMovies() mendefinisikan fungsi untuk memicu pengambilan data film. Pada baris 22, viewModelScope.launch { ... } meluncurkan *coroutine*

dalam cakupan viewModelScope. viewModelScope memastikan bahwa *coroutine* ini akan secara otomatis dibatalkan ketika ViewModel dihancurkan, mencegah kebocoran memori. Pada baris 23, getPopularMoviesUseCase().collect { result -> ... } memanggil invoke() operator dari *use case* dan mengumpulkan nilai-nilai yang dipancarkan oleh Flow yang dikembalikan oleh *use case*. Setiap kali *use case* memancarkan Result baru (Loading, Success, atau Error), blok collect akan menerimanya. Pada baris 24, _popularMovies.value = result memperbarui nilai MutableLiveData, yang secara otomatis akan memberitahu Observer di UI (MainActivity) untuk memperbarui tampilannya.

ViewModelFactory.kt:

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan *package* ini sebagai bagian dari lapisan presentasi untuk ViewModel Factory. Pada baris 3-5, import class ViewModel, ViewModelProvider, dan GetPopularMoviesUseCase.

Pada baris 7, class ViewModelFactory(...) : ViewModelProvider.Factory mendefinisikan ViewModelFactory kustom yang mengimplementasikan ViewModelProvider.Factory. Factory ini bertanggung jawab untuk membuat instance ViewModel dengan dependensi yang diperlukan. Ini adalah cara manual untuk melakukan *Dependency Injection* untuk ViewModel, karena ViewModel tidak dapat memiliki konstruktor dengan parameter secara langsung oleh sistem Android.

Pada baris 8, private val getPopularMoviesUseCase: GetPopularMoviesUseCase adalah dependensi yang dibutuhkan oleh MovieViewModel. Factory ini menerimanya melalui konstruktor.

Pada baris 11, override fun <T : ViewModel> create(modelClass: Class<T>): T adalah metode yang harus diimplementasikan dari ViewModelProvider.Factory. Metode ini bertanggung jawab untuk membuat instance ViewModel yang diminta.

Pada baris 12, if (modelClass.isAssignableFrom(MovieViewModel::class.java)) memeriksa apakah modelClass yang diminta adalah MovieViewModel. Jika ya, pada baris 14, return MovieViewModel(getPopularMoviesUseCase) as T membuat instance baru MovieViewModel dengan dependensi getPopularMoviesUseCase yang disuntikkan. as T

adalah *unsafe cast* yang di-*suppress*. Pada baris 17, throw IllegalArgumentException("Unknown ViewModel class") melempar pengecualian jika modelClass yang diminta tidak dikenali oleh factory ini.

Result.kt:

Pada baris 1, package com.example.movielist.utils mendefinisikan *package* ini untuk utilitas umum. Pada baris 3, sealed class Result<out T> mendefinisikan sealed class bernama Result. Sealed class adalah class abstrak yang nilai-nilainya terbatas pada satu set subclass yang didefinisikan dalam class itu sendiri. Ini sangat berguna untuk merepresentasikan *state* yang berbeda dari sebuah operasi (seperti Loading, Success, Error) dengan cara yang aman dan *type-safe* (out T menunjukkan kovarian tipe).

Pada baris 4, object Loading : Result<Nothing>() adalah objek *singleton* yang merepresentasikan status data sedang dimuat. Nothing menunjukkan bahwa tidak ada data yang terkait dengan *state* ini.

Pada baris 5, data class Success<out T>(val data: T) : Result<T>() adalah *data class* yang merepresentasikan status data berhasil dimuat. Ia membungkus data aktual (val data: T).

Pada baris 6, data class Error(val exception: Exception) : Result<Nothing>() adalah *data class* yang merepresentasikan status terjadi kesalahan. Ia membungkus objek Exception yang menjelaskan kesalahan tersebut.

activity_main.xml:

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah tag root layout, menggunakan ConstraintLayout yang fleksibel untuk memposisikan dan mengukur tampilan. Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan *namespace* untuk atribut layout. Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat layout mengisi seluruh lebar dan tinggi layar. Pada baris 8, tools:context=".presentation.ui.activity.MainActivity" adalah atribut *tooling* untuk Android Studio.

Pada baris 11-18, <TextView android:id="@+id/tv_title_placeholder" ... /> adalah TextView *placeholder* yang disembunyikan (visibility="gone") karena judul "Popular Movies" diatur oleh ActionBar di MainActivity.kt.

Pada baris 21-26, <ProgressBar android:id="@+id/progress_bar" ... /> adalah ProgressBar yang awalnya disembunyikan (visibility="gone") dan diposisikan di tengah layar, berfungsi sebagai indikator loading data.

Pada baris 29-38, <TextView android:id="@+id/tv_error" ... /> adalah TextView untuk menampilkan pesan error. Awalnya disembunyikan, akan muncul saat terjadi kesalahan. app:layout_constraintVertical_chainStyle="packed" dan *constraint* terkait digunakan untuk memusatkan TextView ini bersama dengan tombol "Retry" secara vertikal.

Pada baris 41-48, <Button android:id="@+id/btn_retry" ... /> adalah tombol "Retry". Awalnya disembunyikan, akan muncul di bawah pesan error saat terjadi kesalahan, memungkinkan pengguna untuk mencoba memuat ulang data.

Pada baris 51-58, <androidx.recyclerview.widget.RecyclerView android:id="@+id/rv_movies" ... /> adalah RecyclerView yang digunakan untuk menampilkan daftar film. Ia dikonfigurasi untuk mengisi seluruh ruang match_parent dan dimulai dari bagian atas (constraintTopToTopOf="parent"), yang berarti ia akan berada di bawah ActionBar dan FloatingActionButton dark mode akan menimpanya. tools:listitem digunakan untuk pratinjau layout di Android Studio.

Pada baris 61-73, <com.google.android.material.floatingactionbutton.FloatingActionButton android:id="@+id/btn_dark_mode_toggle" ... /> adalah FloatingActionButton untuk mengalihkan mode gelap. Ia diposisikan di pojok kanan atas (layout_marginEnd, layout_marginTop="?attr/actionBarSize") agar terlihat di bawah ActionBar dan sedikit menimpa RecyclerView. Atribut clickable, focusable, contentDescription diatur. app:srcCompat menentukan ikon awal. app:fabSize="mini" membuatnya berukuran kecil. app:tint dan app:backgroundTint digunakan untuk mewarnai ikon dan *background* tombol agar beradaptasi secara otomatis dengan tema (?attr/colorOnSurface dan ?attr/colorSurface)

item_movie.xml:

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <androidx.cardview.widget.CardView ...> adalah tag root layout untuk item daftar film. CardView digunakan untuk memberikan tampilan item dengan sudut membulat (cardCornerRadius) dan elevasi (cardElevation), yang umum di Material Design.

Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="wrap_content" membuat CardView mengisi lebar penuh dan tingginya sesuai konten. Pada baris 8, android:layout_margin="8dp" menambahkan margin di semua sisi CardView untuk jarak antar item.

Pada baris 12, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam CardView, digunakan untuk mengatur posisi elemen-elemen detail film di dalam kartu. Pada baris 13, android:padding="16dp" menambahkan padding di dalam ConstraintLayout.

Pada baris 15-20, <ImageView android:id="@+id/iv_poster" ... /> mendeklarasikan ImageView untuk menampilkan poster film. Ini dikonfigurasi dengan lebar dan tinggi tetap, skala centerCrop, dan diposisikan di pojok kiri atas.

Pada baris 22-29, <TextView android:id="@+id/tv_movie_title" ... /> adalah TextView untuk menampilkan judul film, diposisikan di sebelah kanan poster dengan gaya teks tebal dan ukuran yang lebih besar.

Pada baris 31-38, <TextView android:id="@+id/tv_release_date" ... /> adalah TextView untuk menampilkan tanggal rilis, diposisikan di bawah judul film.

Pada baris 40-47, <TextView android:id="@+id/tv_vote_average" ... /> adalah TextView untuk menampilkan rata-rata voting/rating film, diposisikan di bawah tanggal rilis.

Pada baris 49-56, <TextView android:id="@+id/tv_overview" ... /> adalah TextView untuk menampilkan ringkasan (overview) film. Ia dikonfigurasi dengan maxLines dan ellipsize="end" untuk memotong teks jika terlalu panjang dan menampilkan elipsis (...). Ini diposisikan di bawah poster utama.

Pada baris 58-63, <Button android:id="@+id/btn_detail" ... /> adalah tombol "Detail". Ini diposisikan di pojok kanan bawah kartu item dan akan meluncurkan DetailActivity saat diklik.

activity_detail.xml:

Pada baris 1, <?xml version="1.0" encoding="utf-8"?> adalah deklarasi standar XML. Pada baris 2, <ScrollView ...> adalah tag root layout. ScrollView memungkinkan konten di dalamnya untuk digulir jika ukurannya melebihi tinggi layar, yang penting untuk halaman detail film yang mungkin memiliki sinopsis panjang. Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan *namespace*. Pada baris 6-7, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat ScrollView mengisi seluruh layar. Pada baris 8, tools:context=".presentation.ui.activity.DetailActivity" adalah atribut *tooling* untuk Android Studio.

Pada baris 10, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam ScrollView, digunakan untuk mengatur posisi elemen-elemen detail. Pada baris 11, android:padding="16dp" menambahkan *padding* di dalam ConstraintLayout.

Pada baris 13-18, <ImageView android:id="@+id/iv_detail_poster" ... /> mendeklarasikan ImageView untuk menampilkan poster film detail. Ini dikonfigurasi dengan lebar mengisi parent (0dp), tinggi tetap (300dp), skala centerCrop, dan diposisikan di bagian atas layout.

Pada baris 20-27, <TextView android:id="@+id/tv_detail_title" ... /> adalah TextView untuk menampilkan judul film. Ini dikonfigurasi dengan ukuran teks besar, gaya tebal, dan diposisikan di bawah poster.

Pada baris 29-36, <TextView android:id="@+id/tv_detail_release_date" ... /> adalah TextView untuk menampilkan tanggal rilis film, diposisikan di bawah judul.

Pada baris 38-45, <TextView android:id="@+id/tv_detail_vote_average" ... /> adalah TextView untuk menampilkan rata-rata *voting* film, diposisikan di bawah tanggal rilis.

Pada baris 47-54, <TextView android:id="@+id/tv_detail_overview_label" ... /> adalah TextView sebagai label "Overview:". Ini dikonfigurasi dengan teks tebal dan diposisikan di bawah rata-rata *voting*.

Pada baris 56-62, <TextView android:id="@+id/tv_detail_overview" ... /> adalah TextView untuk menampilkan teks *overview* sebenarnya. Ini dikonfigurasi dengan ukuran teks normal dan diposisikan di bawah label *overview*.

themes.xml dan themes.xml(night):

Pada baris 1, <resources xmlns:tools="http://schemas.android.com/tools"> adalah tag root untuk file sumber daya, mendeklarasikan *namespace* Tools.

Pada baris 3, <style name="Base.Theme.MovieList" parent="Theme.Material3.DayNight"> mendefinisikan tema dasar aplikasi. name="Base.Theme.MovieList" adalah nama tema. parent="Theme.Material3.DayNight" adalah tema induk yang diwarisi. Theme.Material3.DayNight adalah tema Material Design 3 standar yang secara otomatis mendukung mode siang dan malam, dan yang penting, ia menyertakan ActionBar default yang diperlukan untuk judul dan tombol kembali di Activity.

Pada baris 6, <!-- <item name="colorPrimary">@color/my_light_primary</item> --> adalah komentar dan contoh bagaimana Anda bisa menyesuaikan atribut tema tertentu, seperti warna primer aplikasi.

Pada baris 9, <style name="Theme.MovieList" parent="Base.Theme.MovieList" /> adalah tema akhir yang sebenarnya digunakan oleh aplikasi. Ia mewarisi semua properti dari Base.Theme.MovieList. Ini adalah tema yang diterapkan secara *default* ke semua Activity kecuali jika Activity tersebut secara eksplisit menentukan tema lain di AndroidManifest.xml.

Tautan Git

Berikut adalah tautan untuk semua source code yang telah dibuat.

natnutnot/PrakMobile at master + https://github.com/natnutnot/Mobile