LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

CONNECT TO THE INTERNET

Oleh:

Muhammad Daffa Musyafa NIM. 2310817110007

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 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 : Muhammad Daffa Musyafa
NIM : 2310817210007

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

DAFTAR GAMBAR

DAFTAR TABEL

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 XML

MainActivity.kt

Tabel 1 Source Code Jawaban soal 1 XML

```
1   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.remote.api.RetrofitClient
16  import com.example.movielist.data.repository.MovieRepositoryImpl
17  import com.example.movielist.databinding.ActivityMainBinding
18  import com.example.movielist.domain.usecase.GetPopularMoviesUseCase
19  import com.example.movielist.presentation.ui.adapter.MovieAdapter
20  import com.example.movielist.presentation.viewmodel.MovieViewModel
21  import com.example.movielist.presentation.viewmodel.ViewModelFactory
22  import com.example.movielist.utils.Result
23
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
39          val tmdbApiKey = "71819bfeac768c2a5b9a32b26e50cae1"
40          movieAppPreferences.saveApiKey(tmdbApiKey)
41
42          val movieRepositoryImpl = MovieRepositoryImpl(apiService,
43  movieDao, tmdbApiKey)
44          val getPopularMoviesUseCase =
45  GetPopularMoviesUseCase(movieRepositoryImpl)
46          ViewModelFactory(getPopularMoviesUseCase)
47      }
48
49      override fun onCreate(savedInstanceState: Bundle?) {
50          super.onCreate(savedInstanceState)
51          binding = ActivityMainBinding.inflate(layoutInflater)
52          setContentView(binding.root)
53
54          movieAppPreferences = MovieAppPreferences(this)
55
56          setupRecyclerView()
57          observeViewModel()
58          setupDarkModeToggle()
59
60          binding.btnRetry.setOnClickListener {
61              movieViewModel.fetchPopularMovies()
62          }
63      }
64
```

```
65      private fun setupRecyclerView() {
66          movieAdapter = MovieAdapter()
67          binding.rvMovies.apply {
68              layoutManager = LinearLayoutManager(this@MainActivity)
69              adapter = movieAdapter
70          }
71
72          movieAdapter.onItemClick = { movie ->
73              val intent = Intent(this,
74  DetailActivity::class.java).apply {
75                  putExtra(DetailActivity.EXTRA_MOVIE, movie)
76              }
77              startActivity(intent)
78          }
79      }
80
81      private fun observeViewModel() {
82          movieViewModel.popularMovies.observe(this, Observer { result
83  ->
84              when (result) {
85                  is Result.Loading -> {
86                      binding.progressBar.visibility = View.VISIBLE
87                      binding.tvError.visibility = View.GONE
88                      binding.btnRetry.visibility = View.GONE
89                      binding.rvMovies.visibility = View.GONE
90                  }
91                  is Result.Success -> {
92                      binding.progressBar.visibility = View.GONE
93                      binding.tvError.visibility = View.GONE
94                      binding.btnRetry.visibility = View.GONE
95                      binding.rvMovies.visibility = View.VISIBLE
96                      movieAdapter.submitList(result.data)
97                  }
98                  is Result.Error -> {
99                      binding.progressBar.visibility = View.GONE
100                     binding.rvMovies.visibility = View.GONE
101                     binding.tvError.visibility = View.VISIBLE
102                     binding.btnRetry.visibility = View.VISIBLE
103                     binding.tvError.text = "Error:
104 ${result.exception.message}"
105                     Toast.makeText(this, "Error:
106 ${result.exception.message}", Toast.LENGTH_LONG).show()
107                 }
108             }
109         })
110     }
111
112     private fun setupDarkModeToggle() {
113         binding.switchDarkMode.isChecked =
114 movieAppPreferences.getDarkModeState()
115
116         applyTheme(movieAppPreferences.getDarkModeState())
```

```
117
118         binding.switchDarkMode.setOnCheckedChangeListener { _,
119 isChecked ->
120             movieAppPreferences.saveDarkModeState(isChecked)
121             applyTheme(isChecked)
122         }
123     }
124
125     private fun applyTheme(isDarkMode: Boolean) {
126         if (isDarkMode) {
127
128 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YE
129 S)
130         } else {
131
132 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO
133 )
134         }
135     }
136 }
```

MovieDao.kt

Tabel 2 Source Code Jawaban soal 1 XML

```
1   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 com.example.movielist.data.local.entities.MovieEntity
8
9   @Dao
10  interface MovieDao {
11      @Insert(onConflict = OnConflictStrategy.REPLACE)
12      suspend fun insertAllMovies(movies: List<MovieEntity>)
13
14      @Query("SELECT * FROM movies ORDER BY popularity DESC")
15      suspend fun getAllMovies(): List<MovieEntity>
16
17      @Query("DELETE FROM movies")
18      suspend fun clearAllMovies()
19  }
```

AppDatabase.kt

Tabel 3 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.data.local.database
2
3   import androidx.room.Database
4   import androidx.room.RoomDatabase
5   import com.example.movielist.data.local.dao.MovieDao
6   import com.example.movielist.data.local.entities.MovieEntity
7
8   @Database(entities = [MovieEntity::class], version = 1, exportSchema
9   = false)
10  abstract class AppDatabase : RoomDatabase() {
11      abstract fun movieDao(): MovieDao
12
13      companion object {
14          const val DATABASE_NAME = "tmdb_app_db"
15      }
16  }
```

MovieEntity.kt

Tabel 4 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.data.local.entities
2
3   import androidx.room.Entity
4   import androidx.room.PrimaryKey
5   import com.example.movielist.domain.model.Movie
6
7   @Entity(tableName = "movies")
8   data class MovieEntity(
9       @PrimaryKey
10      val id: Int,
11      val title: String,
12      val overview: String,
13      val posterPath: String?,
14      val releaseDate: String,
15      val voteAverage: Double,
16      val popularity: Double
17  ) {
18      fun toDomainMovie(): Movie {
19          return Movie(
20              id = id,
21              title = title,
22              overview = overview,
23              posterPath = posterPath,
24              releaseDate = releaseDate,
25              voteAverage = voteAverage
```

```
        )
    }

    companion object {
        fun fromDomainMovie(movie: Movie, popularity: Double):
MovieEntity {
            return MovieEntity(
                id = movie.id,
                title = movie.title,
                overview = movie.overview,
                posterPath = movie.posterPath,
                releaseDate = movie.releaseDate,
                voteAverage = movie.voteAverage,
                popularity = popularity
            )
        }
    }
}
```

MovieAppPreferences.kt

Tabel 5 Source Code Jawaban soal 1 XML

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

import android.content.Context
import android.content.SharedPreferences

class MovieAppPreferences(context: Context) {

    private val sharedPreferences: SharedPreferences =
        context.getSharedPreferences("tmdb_app_prefs",
Context.MODE_PRIVATE)

    companion object {
        private const val KEY_API_KEY = "api_key"
        private const val KEY_DARK_MODE = "dark_mode"
    }

    fun saveApiKey(apiKey: String) {
        sharedPreferences.edit().putString(KEY_API_KEY,
apiKey).apply()
    }

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

```

```
26      fun saveDarkModeState(isDarkMode: Boolean) {
27          sharedPreferences.edit().putBoolean(KEY_DARK_MODE,
28  isDarkMode).apply()
29      }
30
31      fun getDarkModeState(): Boolean {
32          return sharedPreferences.getBoolean(KEY_DARK_MODE, false)
33      }
34  }
```

RetrofitClient.kt

Tabel 6 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.data.remote.api
2
3   import
4   com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterF
5   actory
6   import kotlinx.serialization.json.Json
7   import okhttp3.MediaType.Companion.toMediaType
8   import okhttp3.OkHttpClient
9   import okhttp3.logging.HttpLoggingInterceptor
10  import retrofit2.Retrofit
11  import java.util.concurrent.TimeUnit
12
13  object RetrofitClient {
14
15      private const val BASE_URL = "https://api.themoviedb.org/3/"
16
17      private val json = Json {
18          ignoreUnknownKeys = true
19          prettyPrint = true
20      }
21
22      private val okHttpClient: OkHttpClient by lazy {
23          val logging = HttpLoggingInterceptor()
24          logging.setLevel(HttpLoggingInterceptor.Level.BODY)
25
26          OkHttpClient.Builder()
27              .addInterceptor(logging)
28              .connectTimeout(30, TimeUnit.SECONDS)
29              .readTimeout(30, TimeUnit.SECONDS)
30              .writeTimeout(30, TimeUnit.SECONDS)
31              .build()
32      }
33
34      val tmdbApiService: TmdbApiService by lazy {
```

```
35          Retrofit.Builder()
36              .baseUrl(BASE_URL)
37              .client(okHttpClient)
38
39  .addConverterFactory(json.asConverterFactory("application/json".toMedi
40  aType()))
41              .build()
42              .create(TmdbApiService::class.java)
43      }
44  }
```

TmdbApiService.kt

Tabel 7 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.data.remote.api
2
3   import com.example.movielist.data.remote.models.MovieListResponse
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
15      ): Response<MovieListResponse>
16  }
```

MovieDto.kt

Tabel 8 Source Code Jawaban soal 1 XML

```
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 MovieDto(
8       val adult: Boolean,
9       @SerialName("backdrop_path")
10      val backdropPath: String?,
11      @SerialName("genre_ids")
12      val genreIds: List<Int>,
13      val id: Int,
14      @SerialName("original_language")
15      val originalLanguage: String,
16      @SerialName("original_title")
17      val originalTitle: String,
18      val overview: String,
19      val popularity: Double,
20      @SerialName("poster_path")
21      val posterPath: String?,
22      @SerialName("release_date")
23      val releaseDate: String,
24      val title: String,
25      val video: Boolean,
26      @SerialName("vote_average")
27      val voteAverage: Double,
28      @SerialName("vote_count")
29      val voteCount: Int
30  )
```

MovieDtoExtension.kt

Tabel 9 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.data.remote.models
2
3   import com.example.movielist.domain.model.Movie
4   import com.example.movielist.data.local.entities.MovieEntity
5
6   fun MovieDto.toDomainMovie(): Movie {
7       return Movie(
8           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  }
```

MovieListResponse.kt

Tabel 10 Source Code Jawaban soal 1 XML

```
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  )
```

MovieRepository.kt

Tabel 11 Source Code Jawaban soal 1 XML

```
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  {
26          emit(Result.Loading)
27
28          val cachedMovies = movieDao.getAllMovies().map {
29  it.toDomainMovie() }
30          if (cachedMovies.isNotEmpty()) {
31              emit(Result.Success(cachedMovies))
32          }
33
34          try {
35              val response = apiService.getPopularMovies(apiKey =
36  apiKey)
37              if (response.isSuccessful) {
38                  val movieDtos = response.body()?.results ?:
39  emptyList()
40                  val domainMovies = movieDtos.map { it.toDomainMovie()
41  }
42
43                  movieDao.clearAllMovies()
44                  movieDao.insertAllMovies(movieDtos.map {
45  it.toMovieEntity() })
46
47                  emit(Result.Success(domainMovies))
48              } else {
49                  emit(Result.Error(Exception("API Error:
50  ${response.code()} ${response.message()}")))
51              }
52          } catch (e: HttpException) {
53              emit(Result.Error(Exception("Network Error (HTTP
54  ${e.code()}): ${e.message()}")))
55          } catch (e: IOException) {
56              emit(Result.Error(Exception("No Internet Connection or
57  API Timeout: ${e.message}")))
58          } catch (e: Exception) {
59              emit(Result.Error(Exception("An unexpected error
60  occurred: ${e.localizedMessage}")))
61          }
62      }
63  }
```

Movie.kt

Tabel 12 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.domain.model
2
3   import android.os.Parcelable
4   import kotlinx.parcelize.Parcelize
5
6   @Parcelize
7   data class Movie(
8       val id: Int,
9       val title: String,
10      val overview: String,
11      val posterPath: String?,
12      val releaseDate: String,
13      val voteAverage: Double
14  ) : Parcelable
```

GetPopularMoviesUseCase.kt

Tabel 13 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.domain.usecase
2
3   import com.example.movielist.domain.model.Movie
4   import com.example.movielist.data.repository.MovieRepositoryImpl
5   import com.example.movielist.utils.Result
6   import kotlinx.coroutines.flow.Flow
7
8   class GetPopularMoviesUseCase (
9       private val movieRepository: MovieRepositoryImpl
10  ) {
11      operator fun invoke(): Flow<Result<List<Movie>>> {
12          return movieRepository.getPopularMovies()
13      }
14  }
```

DetailActivity.kt

Tabel 14 Source Code Jawaban soal 1 XML

```
1   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
```

```
import androidx.appcompat.app.AppCompatActivity
import com.bumptech.glide.Glide
import com.example.movielist.databinding.ActivityDetailBinding
import com.example.movielist.domain.model.Movie

class DetailActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDetailBinding

    companion object {
        const val EXTRA_MOVIE = "extra_movie"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDetailBinding.inflate(layoutInflater)
        setContentView(binding.root)


        supportActionBar?.setDisplayHomeAsUpEnabled(true)


        val movie = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            intent.getParcelableExtra(EXTRA_MOVIE, Movie::class.java)
        } else {
            @Suppress("DEPRECATION")
            intent.getParcelableExtra(EXTRA_MOVIE)
        }

        movie?.let {

            supportActionBar?.title = it.title

            binding.apply {
                tvDetailTitle.text = it.title
                tvDetailReleaseDate.text = "Release Date:
${it.releaseDate}"
                tvDetailVoteAverage.text = "Rating:
${String.format("%.1f", it.voteAverage)}"
                tvDetailOverview.text = it.overview

                val imageUrl =
"https://image.tmdb.org/t/p/w500${it.posterPath}"
                Glide.with(this@DetailActivity)
                    .load(imageUrl)
                    .centerCrop()
                    .into(ivDetailPoster)
            }
        } ?: run {
            Toast.makeText(this, "Film tidak ditemukan.",
Toast.LENGTH_SHORT).show()
```

```
59              finish()
60          }
61      }
62
63
64      override fun onOptionsItemSelected(item: MenuItem): Boolean {
65          if (item.itemId == android.R.id.home) {
66              onBackPressedDispatcher.onBackPressed()
67              return true
68          }
69          return super.onOptionsItemSelected(item)
70      }
71  }
```

DetailActivity.kt

Tabel 15 Source Code Jawaban soal 1 XML

```
1   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:
18  Int): MovieViewHolder {
19          val binding =
20  ItemMovieBinding.inflate(LayoutInflater.from(parent.context), parent,
21  false)
22          return MovieViewHolder(binding)
23      }
24
25      override fun onBindViewHolder(holder: MovieViewHolder, position:
26  Int) {
27          val movie = getItem(position)
28          holder.bind(movie)
29      }
30
31      inner class MovieViewHolder(private val binding:
32  ItemMovieBinding) :
```

```
33          RecyclerView.ViewHolder(binding.root) {
34
35          init {
36              binding.btnDetail.setOnClickListener {
37                  onItemClick?.invoke(getItem(adapterPosition))
38              }
39          }
40
41          fun bind(movie: Movie) {
42              binding.apply {
43                  tvMovieTitle.text = movie.title
44                  tvReleaseDate.text = "Release Date:
45  ${movie.releaseDate}"
46                  tvVoteAverage.text = "Rating: ${String.format("%.1f",
47  movie.voteAverage)}"
48                  tvOverview.text = movie.overview
49
50                  val imageUrl =
51  "https://image.tmdb.org/t/p/w500${movie.posterPath}"
52
53                  Glide.with(itemView.context)
54                      .load(imageUrl)
55                      .centerCrop()
56                      .into(ivPoster)
57              }
58          }
59      }
60
61      class MovieDiffCallback : DiffUtil.ItemCallback<Movie>() {
62          override fun areItemsTheSame(oldItem: Movie, newItem: Movie):
63  Boolean {
64              return oldItem.id == newItem.id
65          }
66
67          override fun areContentsTheSame(oldItem: Movie, newItem:
68  Movie): Boolean {
69              return oldItem == newItem
70          }
71      }
72  }
```

MovieViewModel.kt

Tabel 16 Source Code Jawaban soal 1 XML

```
1   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.GetPopularMoviesUseCase
9   import com.example.movielist.utils.Result
10  import kotlinx.coroutines.launch
11
12  class MovieViewModel (
13      private val getPopularMoviesUseCase: GetPopularMoviesUseCase
14  ) : ViewModel() {
15
16      private val _popularMovies =
17  MutableLiveData<Result<List<Movie>>>()
18      val popularMovies: LiveData<Result<List<Movie>>> = _popularMovies
19
20      init {
21          fetchPopularMovies()
22      }
23
24      fun fetchPopularMovies() {
25          viewModelScope.launch {
26              getPopularMoviesUseCase().collect { result ->
27                  _popularMovies.value = result
28              }
29          }
30      }
31  }
```

ViewModelFactory.kt

Tabel 17 Source Code Jawaban soal 1 XML

```
1   package com.example.movielist.presentation.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import com.example.movielist.domain.usecase.GetPopularMoviesUseCase
6
7   class ViewModelFactory(
8       private val getPopularMoviesUseCase: GetPopularMoviesUseCase
9   ) : ViewModelProvider.Factory {
10
11      override fun <T : ViewModel> create(modelClass: Class<T>): T {
12          if (modelClass.isAssignableFrom(MovieViewModel::class.java))
13  {
14              @Suppress("UNCHECKED_CAST")
15              return MovieViewModel(getPopularMoviesUseCase) as T
16          }
17          throw IllegalArgumentException("Unknown ViewModel class")
18      }
19  }
```

Result.kt

Tabel 18 Source Code Jawaban soal 1 XML

```
package com.example.movielist.utils

sealed class Result<out T> {
    object Loading : Result<Nothing>()
    data class Success<out T>(val data: T) : Result<T>()
    data class Error(val exception: Exception) : Result<Nothing>()
}
```

Dalam file layout:

activity_main.xml

Dalam XML ada beberapa file tambahan agar sama tampilannya dengan di gambar.

Tabel 19 Source Code Jawaban soal 1 XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".presentation.ui.activity.MainActivity">

    <TextView
        android:id="@+id/tv_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="Popular Movies"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintHorizontal_bias="0.454"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        app:layout_constraintTop_toBottomOf="@id/tv_title"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
30          app:layout_constraintBottom_toBottomOf="parent" />
31
32      <TextView
33          android:id="@+id/tv_error"
34          android:layout_width="wrap_content"
35          android:layout_height="wrap_content"
36          android:text="Error: Could not load movies."
37          android:textColor="@android:color/holo_red_dark"
38          android:visibility="gone"
39          android:gravity="center"
40          app:layout_constraintTop_toBottomOf="@id/tv_title"
41          app:layout_constraintStart_toStartOf="parent"
42          app:layout_constraintEnd_toEndOf="parent"
43          app:layout_constraintBottom_toBottomOf="parent" />
44
45      <Button
46          android:id="@+id/btn_retry"
47          android:layout_width="wrap_content"
48          android:layout_height="wrap_content"
49          android:text="Retry"
50          android:visibility="gone"
51          android:layout_marginTop="8dp"
52          app:layout_constraintTop_toBottomOf="@id/tv_error"
53          app:layout_constraintStart_toStartOf="parent"
54          app:layout_constraintEnd_toEndOf="parent"
55          app:layout_constraintBottom_toBottomOf="parent" />
56
57
58      <androidx.recyclerview.widget.RecyclerView
59          android:id="@+id/rv_movies"
60          android:layout_width="0dp"
61          android:layout_height="0dp"
62          android:layout_marginTop="16dp"
63          app:layout_constraintBottom_toBottomOf="parent"
64          app:layout_constraintEnd_toEndOf="parent"
65          app:layout_constraintStart_toStartOf="parent"
66          app:layout_constraintTop_toBottomOf="@+id/tv_title"
67          tools:listitem="@layout/item_movie" />
68
69      <com.google.android.material.switchmaterial.SwitchMaterial
70          android:id="@+id/switch_dark_mode"
71          android:layout_width="wrap_content"
72          android:layout_height="wrap_content"
73          android:layout_marginEnd="8dp"
74          android:layout_marginBottom="8dp"
75          android:text="Dark Mode"
76          app:layout_constraintEnd_toEndOf="parent"
77          app:layout_constraintTop_toTopOf="parent" />
78
79  </androidx.constraintlayout.widget.ConstraintLayout>
```

item_movie.xml:

Tabel 20 Source Code Jawaban soal 1 XML

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.cardview.widget.CardView
3   xmlns:android="http://schemas.android.com/apk/res/android"
4       xmlns:app="http://schemas.android.com/apk/res-auto"
5       xmlns:tools="http://schemas.android.com/tools"
6       android:layout_width="match_parent"
7       android:layout_height="wrap_content"
8       android:layout_margin="8dp"
9       app:cardCornerRadius="8dp"
10      app:cardElevation="4dp">
11
12      <androidx.constraintlayout.widget.ConstraintLayout
13          android:layout_width="match_parent"
14          android:layout_height="wrap_content"
15          android:padding="16dp">
16
17          <ImageView
18              android:id="@+id/iv_poster"
19              android:layout_width="100dp"
20              android:layout_height="150dp"
21              android:scaleType="centerCrop"
22              app:layout_constraintStart_toStartOf="parent"
23              app:layout_constraintTop_toTopOf="parent"
24              tools:src="@tools:sample/avatars" />
25
26          <TextView
27              android:id="@+id/tv_movie_title"
28              android:layout_width="0dp"
29              android:layout_height="wrap_content"
30              android:layout_marginStart="16dp"
31              android:textStyle="bold"
32              android:textSize="18sp"
33              app:layout_constraintEnd_toEndOf="parent"
34              app:layout_constraintStart_toEndOf="@id/iv_poster"
35              app:layout_constraintTop_toTopOf="@id/iv_poster"
36              tools:text="Movie Title" />
37
38          <TextView
39              android:id="@+id/tv_release_date"
40              android:layout_width="0dp"
41              android:layout_height="wrap_content"
42              android:layout_marginStart="16dp"
43              android:layout_marginTop="4dp"
44              android:textSize="14sp"
45              app:layout_constraintEnd_toEndOf="parent"
```

```
            app:layout_constraintStart_toEndOf="@id/iv_poster"
            app:layout_constraintTop_toBottomOf="@id/tv_movie_title"
            tools:text="Release Date: 2023-01-01" />

        <TextView
            android:id="@+id/tv_vote_average"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="4dp"
            android:textSize="14sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@id/iv_poster"
            app:layout_constraintTop_toBottomOf="@id/tv_release_date"
            tools:text="Rating: 7.5" />

        <TextView
            android:id="@+id/tv_overview"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:maxLines="3"
            android:ellipsize="end"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@id/iv_poster"
            tools:text="This is a short overview of the movie. It
talks about the plot and characters..." />

        <Button
            android:id="@+id/btn_detail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:text="Detail"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/tv_overview" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>
```

activity_detail.xml

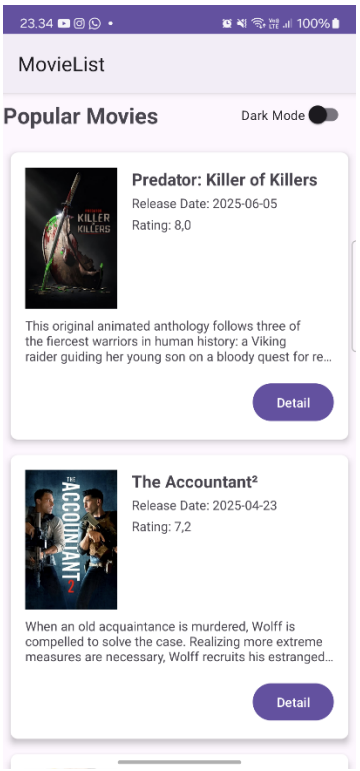
Tabel 21 Source Code Jawaban soal 1 XML

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

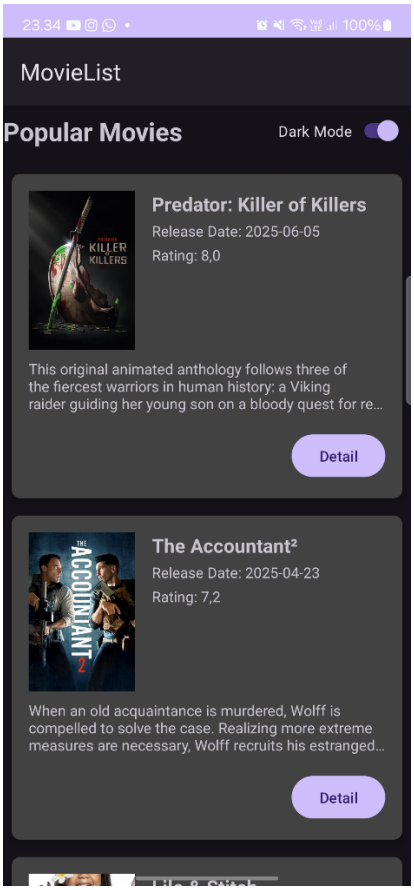
```
7       android:layout_height="match_parent"
8       tools:context=".presentation.ui.activity.DetailActivity">
9
10      <androidx.constraintlayout.widget.ConstraintLayout
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          android:padding="16dp">
14
15          <ImageView
16              android:id="@+id/iv_detail_poster"
17              android:layout_width="0dp"
18              android:layout_height="300dp"
19              android:scaleType="centerCrop"
20              app:layout_constraintEnd_toEndOf="parent"
21              app:layout_constraintStart_toStartOf="parent"
22              app:layout_constraintTop_toTopOf="parent"
23              tools:src="@tools:sample/avatars" />
24
25          <TextView
26              android:id="@+id/tv_detail_title"
27              android:layout_width="0dp"
28              android:layout_height="wrap_content"
29              android:layout_marginTop="16dp"
30              android:textSize="24sp"
31              android:textStyle="bold"
32              app:layout_constraintEnd_toEndOf="parent"
33              app:layout_constraintStart_toStartOf="parent"
34
35  app:layout_constraintTop_toBottomOf="@id/iv_detail_poster"
36              tools:text="Movie Title on Detail Page" />
37
38          <TextView
39              android:id="@+id/tv_detail_release_date"
40              android:layout_width="0dp"
41              android:layout_height="wrap_content"
42              android:layout_marginTop="8dp"
43              android:textSize="16sp"
44              app:layout_constraintEnd_toEndOf="parent"
45              app:layout_constraintStart_toStartOf="parent"
46              app:layout_constraintTop_toBottomOf="@id/tv_detail_title"
47              tools:text="Release Date: 2023-01-01" />
48
49          <TextView
50              android:id="@+id/tv_detail_vote_average"
51              android:layout_width="0dp"
52              android:layout_height="wrap_content"
53              android:layout_marginTop="4dp"
54              android:textSize="16sp"
55              app:layout_constraintEnd_toEndOf="parent"
56              app:layout_constraintStart_toStartOf="parent"
57
58  app:layout_constraintTop_toBottomOf="@id/tv_detail_release_date"
```

```
            tools:text="Rating: 7.5" />

        <TextView
            android:id="@+id/tv_detail_overview_label"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:text="Overview:"
            android:textSize="18sp"
            android:textStyle="bold"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@id/tv_detail_vote_average" />

        <TextView
            android:id="@+id/tv_detail_overview"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:textSize="16sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@id/tv_detail_overview_label"
            tools:text="This is a very long and detailed overview of
the movie. It covers the plot, characters, themes, and critical
reception." />

    </androidx.constraintlayout.widget.ConstraintLayout>
</ScrollView>
```

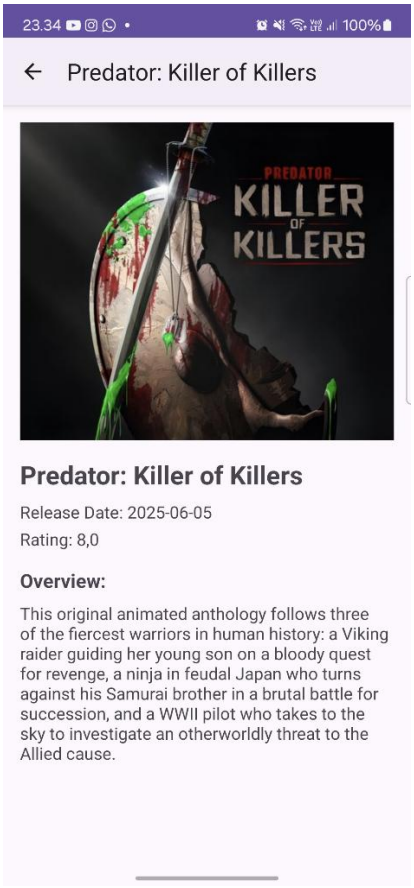
B. Output Program



Gambar 1. Screenshot Hasil Jawaban Soal 1 XML



Gambar 2. Screenshot Hasil Jawaban Soal 1 XML(darkmode)



Gambar 3. Detail Button

Gambar 4. Error Button

C. Pembahasan

MainActivity.kt:

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan package untuk Activity utama, bagian dari lapisan presentasi.

Pada baris 3-21, import berbagai class dan interface yang dibutuhkan untuk Activity (Intent, Bundle, View, Toast, ViewModel, dll.).

Pada baris 23, class MainActivity : AppCompatActivity() mendefinisikan class MainActivity yang merupakan turunan dari AppCompatActivity.

Pada baris 26, private lateinit var binding: ActivityMainBinding mendeklarasikan variabel binding untuk View Binding, yang memungkinkan akses mudah ke elemen UI di activity_main.xml. lateinit berarti akan diinisialisasi nanti.

Pada baris 27, private lateinit var movieAdapter: MovieAdapter mendeklarasikan adapter untuk RecyclerView yang akan menampilkan daftar film.

Pada baris 28, private lateinit var movieAppPreferences: MovieAppPreferences mendeklarasikan instance MovieAppPreferences untuk mengelola SharedPreferences.

Pada baris 30, private var darkModeSwitch: SwitchMaterial? = null mendeklarasikan variabel darkModeSwitch yang akan menampung referensi ke SwitchMaterial di menu.

Pada baris 32, private val movieViewModel: MovieViewModel by viewModels { ... } mendeklarasikan dan menginisialisasi MovieViewModel menggunakan by viewModels delegate. by viewModels memastikan ViewModel bertahan hidup saat konfigurasi berubah. Blok { ... } adalah tempat Dependensi diinjeksi secara manual untuk ViewModelFactory.

Pada baris 33-41, Inisialisasi dependensi: apiService dari RetrofitClient, database Room, movieDao, tmdbApiKey (disimpan ke preferences), movieRepositoryImpl, dan getPopularMoviesUseCase. Semua ini kemudian dilewatkan ke ViewModelFactory.

Pada baris 44, override fun onCreate(savedInstanceState: Bundle?) adalah method yang dipanggil saat Activity pertama kali dibuat.

Pada baris 45, super.onCreate(savedInstanceState) memanggil implementasi onCreate dari superclass (AppCompatActivity).

Pada baris 46, binding = ActivityMainBinding.inflate(layoutInflater) menginisialisasi View Binding.

Pada baris 47, setContentView(binding.root) menetapkan root layout dari binding sebagai tampilan Activity.

Pada baris 49, supportActionBar?.setDisplayHomeAsUpEnabled(false) menonaktifkan tombol back di ActionBar untuk MainActivity.

Pada baris 50, supportActionBar?.title = "Popular Movies" mengatur judul ActionBar menjadi "Popular Movies".

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, binding.btnRetry.setOnClickListener { ... } mengatur listener klik untuk tombol "Retry".

Pada baris 59, movieViewModel.fetchPopularMovies() memanggil fungsi di ViewModel untuk memuat ulang data saat tombol "Retry" diklik.

Pada baris 63, override fun onCreateOptionsMenu(menu: Menu?): Boolean adalah method yang dipanggil untuk membuat menu opsi di ActionBar.

Pada baris 64, menuInflater.inflate(R.menu.main_menu, menu) meng-inflate layout menu XML (main_menu.xml) ke dalam ActionBar.

Pada baris 66, val darkModeMenuItem = menu?.findItem(R.id.action_dark_mode_toggle) menemukan MenuItem untuk dark mode berdasarkan ID-nya.

Pada baris 67, darkModeSwitch = darkModeMenuItem?.actionView as? SwitchMaterial mendapatkan referensi ke SwitchMaterial yang merupakan actionLayout dari MenuItem.

Pada baris 69, darkModeSwitch?.apply { ... } adalah blok apply untuk mengkonfigurasi SwitchMaterial jika tidak null.

Pada baris 70, isChecked = movieAppPreferences.getDarkModeState() mengatur status awal SwitchMaterial berdasarkan preferensi yang tersimpan.

Pada baris 71, applyTheme(isChecked) menerapkan tema (dark/light) saat aplikasi dimulai.

Pada baris 73, setOnCheckedChangeListener { _, isChecked -> ... } mengatur listener untuk perubahan status SwitchMaterial.

Pada baris 74, movieAppPreferences.saveDarkModeState(isChecked) menyimpan status mode gelap yang baru ke SharedPreferences.

Pada baris 75, applyTheme(isChecked) menerapkan tema baru, yang akan membuat ulang Activity.

Pada baris 81, private fun setupRecyclerView() mendefinisikan fungsi untuk menyiapkan RecyclerView.

Pada baris 82, movieAdapter = MovieAdapter() menginisialisasi MovieAdapter.

Pada baris 83-86, binding.rvMovies.apply { ... } mengatur LinearLayoutManager dan adapter untuk RecyclerView.

Pada baris 88, movieAdapter.onItemClick = { movie -> ... } mengatur lambda callback yang akan dipanggil saat tombol "Detail" di item RecyclerView diklik. Sekarang menerima objek Movie.

Pada baris 89, val intent = Intent(this, DetailActivity::class.java) membuat Intent untuk meluncurkan DetailActivity.

Pada baris 90, putExtra(DetailActivity.EXTRA_MOVIE, movie) menambahkan objek Movie sebagai extra ke Intent untuk diteruskan ke DetailActivity.

Pada baris 92, startActivity(intent) meluncurkan DetailActivity.

Pada baris 95, private fun observeViewModel() mendefinisikan fungsi untuk mengamati LiveData dari ViewModel.

Pada baris 96, movieViewModel.popularMovies.observe(this, Observer { result -> ... }) mengamati LiveData popularMovies dari movieViewModel. Blok Observer akan dieksekusi setiap kali nilai LiveData berubah.

Pada baris 97, when (result) { ... } adalah ekspresi when yang memeriksa status Result (Loading, Success, Error) dan memperbarui UI sesuai.

Pada baris 98, is Result.Loading -> { ... } menangani status loading: ProgressBar terlihat, elemen lain disembunyikan.

Pada baris 104, is Result.Success -> { ... } menangani status sukses: ProgressBar dan error/retry disembunyikan, RecyclerView terlihat, dan data disubmit ke adapter.

Pada baris 110, is Result.Error -> { ... } menangani status error: ProgressBar dan RecyclerView disembunyikan, pesan error dan tombol retry terlihat, dan Toast ditampilkan.

Pada baris 119, private fun applyTheme(isDarkMode: Boolean) mendefinisikan fungsi untuk menerapkan tema terang atau gelap.

Pada baris 120-123, AppCompatDelegate.setDefaultNightMode(...) adalah inti dari logika dark mode, yang memberitahu sistem untuk menggunakan mode malam atau tidak. Ini akan menyebabkan Activity dibuat ulang.

MovieDao.kt

Pada baris 1, package com.example.movielist.data.local.dao mendefinisikan package untuk Data Access Object (DAO) film, bagian dari lapisan data lokal.

Pada baris 3, import androidx.room.Dao mengimpor anotasi @Dao, yang menandai antarmuka sebagai DAO Room.

Pada baris 4, import androidx.room.Insert mengimpor anotasi @Insert, digunakan untuk operasi penyisipan data.

Pada baris 5, import androidx.room.OnConflictStrategy mengimpor enum untuk strategi penanganan konflik saat penyisipan.

Pada baris 6, import androidx.room.Query mengimpor anotasi @Query, digunakan untuk kueri SQL kustom.

Pada baris 7, import com.example.movielist.data.local.entities.MovieEntity mengimpor class MovieEntity, entitas Room yang akan dioperasikan.

Pada baris 9, @Dao menandai antarmuka ini sebagai DAO.

Pada baris 10, interface MovieDao mendeklarasikan antarmuka MovieDao.

Pada baris 11, @Insert(onConflict = OnConflictStrategy.REPLACE) menandai fungsi ini sebagai operasi penyisipan. OnConflictStrategy.REPLACE berarti jika ada konflik (misalnya, ID yang sama), data lama akan diganti.

Pada baris 12, suspend fun insertAllMovies(movies: List<MovieEntity>) mendefinisikan fungsi suspend untuk menyisipkan daftar MovieEntity. Kata kunci suspend menunjukkan bahwa ini adalah fungsi coroutine yang dapat dihentikan (pausable) dan dilanjutkan.

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" dan mengurutkannya berdasarkan popularitas secara descending.

Pada baris 15, suspend fun getAllMovies(): List<MovieEntity> mendefinisikan fungsi suspend untuk mengambil semua MovieEntity.

Pada baris 17, @Query("DELETE FROM movies") menandai fungsi ini dengan kueri SQL kustom untuk menghapus semua data dari tabel "movies".

Pada baris 18, suspend fun clearAllMovies() mendefinisikan fungsi suspend untuk menghapus semua film dari cache.

AppDatabase.kt

Pada baris 1, package com.example.movielist.data.local.database mendefinisikan package untuk class database Room, bagian dari lapisan data lokal.

Pada baris 3, import androidx.room.Database mengimpor anotasi @Database, yang menandai class sebagai database Room.

Pada baris 4, import androidx.room.RoomDatabase mengimpor class RoomDatabase, superclass dari database Room.

Pada baris 5, import com.example.movielist.data.local.dao.MovieDao mengimpor antarmuka MovieDao.

Pada baris 6, import com.example.movielist.data.local.entities.MovieEntity mengimpor class MovieEntity, entitas yang akan menjadi bagian dari database.

Pada baris 8, @Database(entities = [MovieEntity::class], version = 1, exportSchema = false) menandai class sebagai database Room.

Pada baris 8, entities = [MovieEntity::class] mendaftarkan MovieEntity sebagai entitas yang akan menjadi tabel dalam database ini.

Pada baris 8, version = 1 menentukan versi database. Jika skema database berubah, versi harus di-increment.

Pada baris 8, exportSchema = false menonaktifkan ekspor skema database ke file, cocok untuk pengembangan.

Pada baris 9, abstract class AppDatabase : RoomDatabase() mendefinisikan class abstrak AppDatabase yang mewarisi dari RoomDatabase.

Pada baris 10, abstract fun movieDao(): MovieDao mendefinisikan fungsi abstrak untuk mendapatkan instance MovieDao, yang akan diimplementasikan oleh Room secara otomatis.

Pada baris 12, companion object { ... } adalah objek pendamping class.

Pada baris 13, 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 untuk entitas Room, bagian dari lapisan data lokal.

Pada baris 3, import androidx.room.Entity mengimpor anotasi @Entity, yang menandai class data sebagai tabel database Room.

Pada baris 4, import androidx.room.PrimaryKey mengimpor anotasi @PrimaryKey, yang menandai properti sebagai primary key tabel.

Pada baris 5, import com.example.movielist.domain.model.Movie mengimpor class Movie, model domain yang akan dipetakan.

Pada baris 7, @Entity(tableName = "movies") menandai class data ini sebagai entitas Room dan menentukan nama tabel database menjadi "movies".

Pada baris 8, data class MovieEntity(...) mendefinisikan class data MovieEntity, yang akan mewakili satu baris dalam tabel movies.

Pada baris 9, @PrimaryKey val id: Int mendeklarasikan id sebagai primary key untuk tabel, setiap MovieEntity harus memiliki ID unik.

Pada baris 10-15, properti lain seperti title, overview, posterPath, releaseDate, voteAverage, dan popularity adalah kolom-kolom dalam tabel movies.

Pada baris 17, fun toDomainMovie(): Movie mendefinisikan fungsi ekstensi yang mengonversi instance MovieEntity menjadi Movie domain model. Ini digunakan saat membaca data dari database dan menyediakannya ke lapisan domain/presentasi.

Pada baris 27, companion object { ... } adalah objek pendamping class.

Pada baris 28, fun fromDomainMovie(movie: Movie, popularity: Double): MovieEntity mendefinisikan fungsi factory dalam companion object yang mengonversi Movie domain model menjadi MovieEntity yang cocok untuk penyimpanan di Room. popularity disertakan karena merupakan kolom yang disimpan di database.

MovieAppPreferences.kt

Pada baris 1, package com.example.movielist.data.local mendefinisikan nama package dari file Kotlin ini, mengelompokkannya dalam lapisan data lokal.

Pada baris 3, import android.content.Context mengimpor class Context yang menyediakan akses ke sumber daya dan layanan sistem.

Pada baris 4, import android.content.SharedPreferences mengimpor class SharedPreferences, API untuk menyimpan data primitif dalam format key-value pairs.

Pada baris 6, class MovieAppPreferences(context: Context) mendefinisikan class MovieAppPreferences yang bertanggung jawab untuk mengelola SharedPreferences, menerima Context untuk inisialisasi.

Pada baris 8, private val sharedPreferences: SharedPreferences = ... mendeklarasikan properti private untuk instance SharedPreferences.

Pada baris 9, context.getSharedPreferences("tmdb_app_prefs", Context.MODE_PRIVATE) menginisialisasi SharedPreferences dengan nama file "tmdb_app_prefs" dan mode private (hanya bisa diakses oleh aplikasi ini).

Pada baris 11, companion object { ... } adalah objek pendamping yang berisi properti dan fungsi yang terkait dengan class, tetapi tidak memerlukan instance class.

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 status mode gelap.

Pada baris 15, fun saveApiKey(apiKey: String) mendefinisikan fungsi untuk menyimpan API key.

Pada baris 16, sharedPreferences.edit().putString(KEY_API_KEY, apiKey).apply() mengambil editor SharedPreferences, menyimpan string dengan kunci KEY_API_KEY, dan menerapkan perubahan secara asinkron.

Pada baris 19, fun getApiKey(): String? mendefinisikan fungsi untuk mengambil API key yang tersimpan.

Pada baris 20, return sharedPreferences.getString(KEY_API_KEY, null) mengambil string dari SharedPreferences dengan kunci KEY_API_KEY; jika tidak ada, mengembalikan null.

Pada baris 23, fun saveDarkModeState(isDarkMode: Boolean) mendefinisikan fungsi untuk menyimpan status mode gelap (boolean).

Pada baris 24, sharedPreferences.edit().putBoolean(KEY_DARK_MODE, isDarkMode).apply() menyimpan boolean dengan kunci KEY_DARK_MODE.

Pada baris 27, fun getDarkModeState(): Boolean mendefinisikan fungsi untuk mengambil status mode gelap yang tersimpan.

Pada baris 28, return sharedPreferences.getBoolean(KEY_DARK_MODE, false) mengambil boolean dari SharedPreferences dengan kunci KEY_DARK_MODE; jika tidak ada, mengembalikan false (default light mode).

RetrofitClient.kt:

Pada baris 1, package com.example.movielist.data.remote.api mendefinisikan package untuk klien API, bagian dari lapisan data remote.

Pada baris 3, import com.jakewharton.retrofit.retrofit2.kotlinx.serialization.asConverterFactory mengimpor fungsi ekstensi untuk menggunakan KotlinX Serialization dengan Retrofit.

Pada baris 4, import kotlinx.serialization.json.Json mengimpor class Json dari KotlinX Serialization, digunakan untuk mengonfigurasi parser JSON.

Pada baris 5, import okhttp3.MediaType.Companion.toMediaType mengimpor fungsi ekstensi untuk membuat MediaType.

Pada baris 6, import okhttp3.OkHttpClient mengimpor class OkHttpClient, klien HTTP yang akan digunakan Retrofit.

Pada baris 7, import okhttp3.logging.HttpLoggingInterceptor mengimpor interceptor untuk logging permintaan dan respons HTTP.

Pada baris 8, import retrofit2.Retrofit mengimpor class Retrofit, builder utama untuk API service.

Pada baris 9, import java.util.concurrent.TimeUnit mengimpor class TimeUnit untuk mengonfigurasi durasi timeout.

Pada baris 11, object RetrofitClient mendeklarasikan objek singleton RetrofitClient, artinya hanya ada satu instance dari class ini di seluruh aplikasi.

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

Pada baris 15, private val json = Json { ... } menginisialisasi instance Json untuk konfigurasi parser JSON.

Pada baris 16, ignoreUnknownKeys = true mengonfigurasi parser untuk mengabaikan kunci JSON yang tidak ada di model data Kotlin Anda, mencegah crash jika ada perubahan di API.

Pada baris 17, prettyPrint = true mengonfigurasi output JSON agar mudah dibaca (berguna untuk debugging).

Pada baris 20, private val okHttpClient: OkHttpClient by lazy { ... } mendeklarasikan instance OkHttpClient secara lazy (dibuat saat pertama kali diakses).

Pada baris 21, val logging = HttpLoggingInterceptor() membuat instance HttpLoggingInterceptor.

Pada baris 22, logging.setLevel(HttpLoggingInterceptor.Level.BODY) mengatur level logging untuk menampilkan header dan body dari permintaan/respons HTTP di Logcat.

Pada baris 24, OkHttpClient.Builder() memulai proses membangun OkHttpClient.

Pada baris 25, .addInterceptor(logging) menambahkan interceptor logging ke klien HTTP.

Pada baris 26, .connectTimeout(30, TimeUnit.SECONDS) mengatur waktu tunggu untuk membuat koneksi.

Pada baris 27, .readTimeout(30, TimeUnit.SECONDS) mengatur waktu tunggu untuk membaca data dari server.

Pada baris 28, .writeTimeout(30, TimeUnit.SECONDS) mengatur waktu tunggu untuk mengirim data ke server.

Pada baris 29, .build() membangun instance OkHttpClient.

Pada baris 32, val tmdbApiService: TmdbApiService by lazy { ... } mendeklarasikan instance TmdbApiService secara lazy.

Pada baris 33, Retrofit.Builder() memulai proses membangun Retrofit.

Pada baris 34, .baseUrl(BASE_URL) menetapkan URL dasar untuk semua permintaan.

Pada baris 35, .client(okHttpClient) menetapkan OkHttpClient kustom yang baru dibuat.

Pada baris 36, .addConverterFactory(json.asConverterFactory("application/json".toMediaType())) menambahkan konverter untuk mengubah JSON menjadi objek Kotlin menggunakan KotlinX Serialization.

Pada baris 37, .build() membangun instance Retrofit.

Pada baris 38, .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 untuk antarmuka API, bagian dari lapisan data remote.

Pada baris 3, import com.example.movielist.data.remote.models.MovieListResponse mengimpor class model respons untuk daftar film.

Pada baris 4, import retrofit2.Response mengimpor class Response dari Retrofit untuk membungkus respons HTTP.

Pada baris 5, import retrofit2.http.GET mengimpor anotasi @GET untuk menentukan jenis permintaan HTTP (GET).

Pada baris 6, import retrofit2.http.Query mengimpor anotasi @Query untuk menambahkan parameter kueri ke URL.

Pada baris 8, interface TmdbApiService mendeklarasikan antarmuka TmdbApiService.

Pada baris 10, @GET("movie/popular") menandai fungsi ini untuk melakukan permintaan GET ke endpoint "movie/popular" dari URL dasar TMDB API.

Pada baris 11, suspend fun getPopularMovies(...) mendefinisikan fungsi suspend untuk mendapatkan daftar film populer.

Pada baris 12, @Query("api_key") apiKey: String mendeklarasikan parameter kueri "api_key" yang wajib diisi.

Pada baris 13, @Query("language") language: String = "en-US" mendeklarasikan parameter kueri "language" dengan nilai default "en-US".

Pada baris 14, @Query("page") page: Int = 1 mendeklarasikan parameter kueri "page" dengan nilai default 1 untuk pagination.

Pada baris 15, : Response<MovieListResponse> menentukan bahwa fungsi ini akan mengembalikan objek Response yang membungkus MovieListResponse.

MovieDto.kt:

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan package untuk model data remote (DTOs), bagian dari lapisan data.

Pada baris 3, import kotlinx.serialization.SerialName mengimpor anotasi @SerialName, digunakan untuk memetakan nama properti Kotlin ke nama kunci JSON yang berbeda.

Pada baris 4, import kotlinx.serialization.Serializable mengimpor anotasi @Serializable, yang menandai class data ini agar dapat di-serialize dan di-deserialize oleh KotlinX Serialization.

Pada baris 6, @Serializable menandai class data ini sebagai class yang bisa diubah menjadi/dari JSON.

Pada baris 7, data class MovieDto(...) mendefinisikan class data MovieDto, yang merupakan Data Transfer Object (DTO) untuk film dari TMDB API. Struktur propertinya mencerminkan struktur JSON yang diterima dari API.

Pada baris 8, val adult: Boolean mendefinisikan properti adult (apakah film ditujukan untuk dewasa).

Pada baris 9, @SerialName("backdrop_path") val backdropPath: String? memetakan kunci JSON "backdrop_path" ke properti backdropPath di Kotlin; tanda ? menunjukkan properti ini bisa null.

Pada baris 11, @SerialName("genre_ids") val genreIds: List<Int> memetakan kunci JSON "genre_ids" ke daftar ID genre.

Pada baris 13, val id: Int mendefinisikan properti id (ID unik film).

Pada baris 14, @SerialName("original_language") val originalLanguage: String memetakan kunci JSON "original_language".

Pada baris 16, @SerialName("original_title") val originalTitle: String memetakan kunci JSON "original_title".

Pada baris 18, val overview: String mendefinisikan properti overview (ringkasan film).

Pada baris 19, val popularity: Double mendefinisikan properti popularity.

Pada baris 20, @SerialName("poster_path") val posterPath: String? memetakan kunci JSON "poster_path" ke properti posterPath (jalur ke gambar poster).

Pada baris 22, @SerialName("release_date") val releaseDate: String memetakan kunci JSON "release_date" ke properti releaseDate (tanggal rilis film).

Pada baris 24, val title: String mendefinisikan properti title (judul film).

Pada baris 25, val video: Boolean mendefinisikan properti video.

Pada baris 26, @SerialName("vote_average") val voteAverage: Double memetakan kunci JSON "vote_average" (rata-rata rating).

Pada baris 28, @SerialName("vote_count") val voteCount: Int memetakan kunci JSON "vote_count" (jumlah voting).

MovieListResponse.kt

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan package untuk model data remote (DTOs), bagian dari lapisan data.

Pada baris 3, import kotlinx.serialization.SerialName mengimpor anotasi @SerialName.

Pada baris 4, import kotlinx.serialization.Serializable mengimpor anotasi @Serializable.

Pada baris 6, @Serializable menandai class ini agar dapat di-serialize dan di-deserialize oleh KotlinX Serialization.

Pada baris 7, data class MovieListResponse(...) mendefinisikan class data MovieListResponse, yang mewakili struktur respons keseluruhan dari API ketika meminta daftar film (misalnya, endpoint movie/popular akan mengembalikan objek dengan properti seperti page, results, total_pages, total_results).

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 dari MovieDto, yaitu daftar film yang sebenarnya.

Pada baris 10, @SerialName("total_pages") val totalPages: Int memetakan kunci JSON "total_pages" ke properti totalPages (jumlah total halaman).

Pada baris 12, @SerialName("total_results") val totalResults: Int memetakan kunci JSON "total_results" ke properti totalResults (jumlah total hasil).

MovieDtoExtension.kt:

Pada baris 1, package com.example.movielist.data.remote.models mendefinisikan package dari file ekstensi ini, yang berisi fungsi-fungsi untuk mengubah model data.

Pada baris 3, import com.example.movielist.domain.model.Movie mengimpor class Movie, model domain.

Pada baris 4, import com.example.movielist.data.local.entities.MovieEntity mengimpor class MovieEntity, entitas Room.

Pada baris 6, fun MovieDto.toDomainMovie(): Movie mendefinisikan fungsi ekstensi untuk class MovieDto. Fungsi ini akan mengubah (memetakan) sebuah objek MovieDto (dari API) menjadi Movie domain model. Ini penting untuk menjaga pemisahan lapisan, memastikan bahwa lapisan domain hanya berinteraksi dengan modelnya sendiri.

Pada baris 15, fun MovieDto.toMovieEntity(): MovieEntity mendefinisikan fungsi ekstensi lain untuk class MovieDto. Fungsi ini akan mengubah (memetakan) sebuah objek MovieDto (dari API) menjadi MovieEntity yang dapat disimpan di Room Database.

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 yang didefinisikan 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. Ini juga mengambil dependensi TmdbApiService (untuk jaringan) dan MovieDao (untuk database lokal) melalui konstruktornya.

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, if (cachedMovies.isNotEmpty()) { emit(Result.Success(cachedMovies)) } Jika ada data di cache, data tersebut segera dipancarkan sebagai Result.Success. Ini memastikan aplikasi dapat menampilkan data dengan cepat (misalnya, dalam mode offline atau saat jaringan lambat) sebelum mencoba dari jaringan.

Pada baris 28, try { ... } catch (...) { ... } adalah blok penanganan kesalahan yang akan mencoba mengambil data dari jaringan dan menangani berbagai jenis error.

Pada baris 29, val response = apiService.getPopularMovies(apiKey = apiKey) melakukan panggilan API ke TMDB untuk mendapatkan daftar film populer terbaru.

Pada baris 30, if (response.isSuccessful) { ... } memeriksa apakah panggilan API berhasil (kode status 2xx).

Pada baris 31, val movieDtos = response.body()?.results ?: emptyList() mengambil daftar DTO film dari body respons; jika null, mengembalikan daftar kosong.

Pada baris 32, val domainMovies = movieDtos.map { it.toDomainMovie() } memetakan DTO yang diterima dari API ke domain model Movie.

Pada baris 34, movieDao.clearAllMovies() menghapus semua data film yang ada di cache Room. Ini adalah bagian dari strategi refresh cache.

Pada baris 35, movieDao.insertAllMovies(movieDtos.map { it.toMovieEntity() }) menyisipkan data film yang baru diterima dari API ke dalam cache Room.

Pada baris 37, emit(Result.Success(domainMovies)) memancarkan data film terbaru yang berhasil diambil dari API ke Flow. Ini akan memperbarui UI dengan data terbaru.

Pada baris 39-47, catch (e: HttpException), catch (e: IOException), dan catch (e: Exception) menangani berbagai jenis kesalahan (HTTP error, masalah koneksi/timeout, atau error umum) dan memancarkan Result.Error yang sesuai.

Movie.kt

Pada baris 1, package com.example.movielist.domain.model mendefinisikan package untuk model domain, bagian dari lapisan domain.

Pada baris 3, import android.os.Parcelable mengimpor antarmuka Parcelable, yang memungkinkan objek ini untuk dikirim antar komponen Android (misalnya antar Activity) secara efisien.

Pada baris 4, import kotlinx.parcelize.Parcelize mengimpor anotasi @Parcelize dari plugin Kotlin Parcelize.

Pada baris 6, @Parcelize adalah anotasi yang secara otomatis menghasilkan implementasi kode Parcelable boilerplate untuk class data ini. Ini menggantikan kebutuhan untuk menulis implementasi Parcelable secara manual.

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

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

GetPopularMoviesUseCase.kt

Pada baris 1, package com.example.movielist.domain.usecase mendefinisikan package untuk use case, bagian dari lapisan domain.

Pada baris 3-6, import class yang dibutuhkan (model domain, repository implementasi, 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 Clean Architecture yang lebih ketat, ini seharusnya bergantung pada antarmuka MovieRepository dari lapisan domain, namun disesuaikan dengan keputusan Anda untuk menggabungkannya.

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.

DetailActivity.kt

Pada baris 1, package com.example.movielist.presentation.ui.activity mendefinisikan package untuk Activity detail, bagian dari lapisan presentasi.

Pada baris 3-9, import berbagai class dan interface yang dibutuhkan untuk Activity (Bundle, MenuItem, Toast, AppCompatActivity, Glide, dll.).

Pada baris 11, class DetailActivity : AppCompatActivity() mendefinisikan class DetailActivity yang akan menampilkan detail film.

Pada baris 14, private lateinit var binding: ActivityDetailBinding mendeklarasikan variabel binding untuk View Binding.

Pada baris 16, companion object { ... } adalah objek pendamping class.

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 method yang dipanggil saat Activity pertama kali dibuat.

Pada baris 21, super.onCreate(savedInstanceState) memanggil implementasi onCreate dari superclass.

Pada baris 22, binding = ActivityDetailBinding.inflate(layoutInflater) menginisialisasi View Binding.

Pada baris 23, setContentView(binding.root) menetapkan layout activity_detail.xml sebagai tampilan Activity.

Pada baris 25, supportActionBar?.setDisplayHomeAsUpEnabled(true) mengaktifkan tombol "Home" (biasanya panah kembali) di ActionBar, memungkinkan navigasi ke Activity sebelumnya.

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. Ini menggunakan cara yang berbeda untuk API level Tiramisu (33) ke atas, dan cara yang sudah didepresiasi untuk API level di bawahnya.

Pada baris 33, movie?.let { ... } adalah blok yang akan dieksekusi hanya jika movie tidak null (artinya objek berhasil diterima).

Pada baris 34, supportActionBar?.title = it.title mengatur judul ActionBar menjadi judul film.

Pada baris 36, binding.apply { ... } adalah blok apply untuk mengatur data film ke elemen UI.

Pada baris 37-40, tvDetailTitle.text = it.title, tvDetailReleaseDate.text, tvDetailVoteAverage.text, dan tvDetailOverview.text = it.overview mengisi TextViews dengan data dari objek Movie.

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 ke ImageView.

Pada baris 48-51, ?: run { ... } adalah operator Elvis yang akan dieksekusi jika movie adalah null; ini menampilkan Toast dan menutup Activity.

Pada baris 54, override fun onOptionsItemSelected(item: MenuItem): Boolean adalah method yang dipanggil ketika item di ActionBar diklik.

Pada baris 55, if (item.itemId == android.R.id.home) memeriksa apakah item yang diklik adalah tombol "Home" (tombol back).

Pada baris 56, onBackPressedDispatcher.onBackPressed() mensimulasikan penekanan tombol back perangkat, membawa pengguna kembali ke Activity sebelumnya.

MovieAdapter.kt

Pada baris 1, package com.example.movielist.presentation.ui.adapter mendefinisikan package untuk adapter RecyclerView, bagian dari lapisan presentasi.

Pada baris 3-8, import berbagai class yang dibutuhkan (LayoutInflater, ViewGroup, DiffUtil, ListAdapter, RecyclerView, Glide, Binding, model domain).

Pada baris 10, class MovieAdapter : ListAdapter<Movie, MovieAdapter.MovieViewHolder>(MovieDiffCallback()) mendefinisikan MovieAdapter. Ini adalah ListAdapter, yang merupakan jenis adapter RecyclerView yang efisien dalam memperbarui daftar item. Ia menerima Movie sebagai tipe data dan MovieViewHolder sebagai ViewHolder. MovieDiffCallback() digunakan untuk membandingkan item secara efisien.

Pada baris 12, var onItemClick: ((Movie) -> Unit)? = null mendeklarasikan sebuah properti lambda nullable bernama onItemClick. Ini akan berfungsi sebagai callback yang dapat diatur dari MainActivity untuk menangani klik pada tombol "Detail" di setiap item.

Pada baris 14, override fun onCreateViewHolder(...) membuat dan mengembalikan instance MovieViewHolder. Ini meng-inflate layout item_movie.xml menggunakan View Binding.

Pada baris 19, override fun onBindViewHolder(...) mengikat data Movie ke ViewHolder pada posisi tertentu di daftar.

Pada baris 24, inner class MovieViewHolder(...) mendefinisikan inner class MovieViewHolder yang memegang referensi ke tampilan setiap item di RecyclerView.

Pada baris 27, init { binding.btnDetail.setOnClickListener { ... } } adalah blok inisialisasi untuk ViewHolder. Di sinilah OnClickListener untuk tombol "Detail" diatur.

Pada baris 28, onItemClick?.invoke(getItem(adapterPosition)) memanggil lambda onItemClick yang telah diatur dari MainActivity, meneruskan objek Movie yang sesuai

dengan posisi item yang diklik. Tanda ? memastikan invoke hanya dipanggil jika onItemClick tidak null.

Pada baris 32, fun bind(movie: Movie) mendefinisikan fungsi bind yang bertanggung jawab untuk mengisi tampilan item dengan data dari objek Movie.

Pada baris 33, binding.apply { ... } menggunakan fungsi scope apply untuk bekerja dengan properti binding.

Pada baris 34-37, tvMovieTitle.text, tvReleaseDate.text, tvVoteAverage.text, tvOverview.text mengisi TextViews dengan data dari movie.

Pada baris 39, val imageUrl = "https://image.tmdb.org/t/p/w500${movie.posterPath}" membangun URL 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 DiffUtil.ItemCallback kustom. Ini digunakan oleh ListAdapter untuk menghitung perbedaan antara daftar lama dan baru, sehingga RecyclerView dapat diperbarui secara efisien.

Pada baris 47, override fun areItemsTheSame(...) memeriksa apakah dua item mewakili objek yang sama (biasanya dengan membandingkan ID unik mereka).

Pada baris 50, override fun areContentsTheSame(...) memeriksa apakah konten dari dua item yang sama persis juga sama (digunakan untuk mendeteksi perubahan dalam data item).

MovieViewModel.kt

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan package untuk ViewModel, bagian dari lapisan presentasi.

Pada baris 3-9, import berbagai class yang dibutuhkan (LiveData, ViewModel, ViewModelScope, model domain, use case, Result, Flow, Coroutines).

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 agar data tetap ada saat konfigurasi perangkat berubah (misalnya, rotasi layar).

Pada baris 12, private val getPopularMoviesUseCase: GetPopularMoviesUseCase mendeklarasikan dependensi pada GetPopularMoviesUseCase. ViewModel tidak berinteraksi langsung dengan repository atau API, tetapi mendelegasikan 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. Ini akan menampung hasil pengambilan data film (berupa Result yang membungkus daftar Movie).

Pada baris 16, val popularMovies: LiveData<Result<List<Movie>>> = _popularMovies mendeklarasikan LiveData publik yang tidak dapat diubah (immutable). UI akan mengamati LiveData ini untuk mendapatkan pembaruan data.

Pada baris 18, init { fetchPopularMovies() } adalah blok inisialisasi yang akan dipanggil saat instance MovieViewModel pertama kali dibuat. Ini memicu 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 dibatalkan secara otomatis 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. Perubahan ini akan secara otomatis memberitahu Observer di UI (MainActivity) untuk memperbarui tampilannya.

ViewModelFactory.kt

Pada baris 1, package com.example.movielist.presentation.viewmodel mendefinisikan package untuk ViewModel Factory, bagian dari lapisan presentasi.

Pada baris 3, import androidx.lifecycle.ViewModel mengimpor class ViewModel.

Pada baris 4, import androidx.lifecycle.ViewModelProvider mengimpor class ViewModelProvider, yang digunakan untuk membuat instance ViewModel.

Pada baris 5, import com.example.movielist.domain.usecase.GetPopularMoviesUseCase mengimpor use case.

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, 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 method yang harus diimplementasikan dari ViewModelProvider.Factory. Method 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.

Pada baris 14, return MovieViewModel(getPopularMoviesUseCase) as T jika ViewModel yang diminta adalah MovieViewModel, maka instance baru MovieViewModel dibuat 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 untuk utilitas umum.

Pada baris 3, sealed class Result<out T> mendefinisikan sealed class bernama Result. Sealed class adalah class yang nilai-nilainya terbatas pada satu set subclass yang didefinisikan dalam class itu sendiri. Ini sangat berguna untuk merepresentasikan state yang berbeda (seperti Loading, Success, Error) dengan cara yang aman dan type-safe. out T menunjukkan bahwa T adalah tipe kovarian, artinya Result<Subtype> adalah subclass dari Result<Supertype>.

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 class data 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 class data 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 untuk layout ini. ConstraintLayout adalah layout yang fleksibel yang memungkinkan Anda memposisikan dan mengukur tampilan secara relatif satu sama lain atau ke parent layout.

Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan namespace yang berbeda untuk atribut layout.

Pada baris 6, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat layout mengisi seluruh lebar dan tinggi layar.

Pada baris 7, tools:context=".presentation.ui.activity.MainActivity" adalah atribut tools untuk Android Studio, membantu dalam pratinjau layout.

Pada baris 9, <TextView ...> mendeklarasikan TextView untuk judul "Popular Movies".

Pada baris 10, android:id="@+id/tv_title" memberikan ID unik untuk TextView ini.

Pada baris 11-15, mengatur teks, ukuran, gaya, margin, dan constraint posisinya di pojok kiri atas parent.

Pada baris 18, <ProgressBar ...> mendeklarasikan ProgressBar untuk indikator loading.

Pada baris 19, android:id="@+id/progress_bar" memberikan ID unik.

Pada baris 20, android:visibility="gone" menyembunyikan ProgressBar secara default.

Pada baris 21-24, mengatur constraint posisi ProgressBar di tengah layar.

Pada baris 26, <TextView ...> mendeklarasikan TextView untuk pesan error.

Pada baris 27, android:id="@+id/tv_error" memberikan ID unik.

Pada baris 28-30, mengatur teks, warna, visibilitas (default gone), dan perataan teks.

Pada baris 31, app:layout_constraintVertical_chainStyle="packed" dan baris 32-35, mengatur constraint yang membentuk rantai vertikal dengan tombol "Retry", memastikan keduanya terpusat sebagai grup.

Pada baris 37, <Button ...> mendeklarasikan tombol "Retry".

Pada baris 38, android:id="@+id/btn_retry" memberikan ID unik.

Pada baris 39, android:visibility="gone" menyembunyikan tombol secara default.

Pada baris 40-44, mengatur teks, margin, dan constraint posisi di bawah tv_error dan terpusat horizontal.

Pada baris 46, <androidx.recyclerview.widget.RecyclerView ...> mendeklarasikan RecyclerView untuk menampilkan daftar film.

Pada baris 47, android:id="@+id/rv_movies" memberikan ID unik.

Pada baris 48-52, mengatur lebar, tinggi, margin, dan constraint posisi RecyclerView agar mengisi sisa ruang di bawah judul.

Pada baris 53, tools:listitem="@layout/item_movie" adalah atribut tools untuk Android Studio, menampilkan pratinjau item layout di RecyclerView.

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 untuk layout item ini. CardView digunakan untuk memberikan tampilan item dengan sudut membulat dan elevasi (bayangan), yang umum di Material Design.

Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan namespace.

Pada baris 6, android:layout_width="match_parent" membuat lebar CardView mengisi lebar parent.

Pada baris 7, android:layout_height="wrap_content" membuat tinggi CardView pas dengan kontennya.

Pada baris 8, android:layout_margin="8dp" menambahkan margin 8dp di semua sisi CardView, memberikan ruang antar item.

Pada baris 9, app:cardCornerRadius="8dp" mengatur radius sudut CardView menjadi 8dp.

Pada baris 10, app:cardElevation="4dp" memberikan bayangan (elevasi) pada CardView.

Pada baris 12, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam CardView, digunakan untuk mengatur posisi elemen-elemen detail film.

Pada baris 13, android:padding="16dp" menambahkan padding 16dp di dalam ConstraintLayout.

Pada baris 15, <ImageView ...> mendeklarasikan ImageView untuk menampilkan poster film.

Pada baris 16, android:id="@+id/iv_poster" memberikan ID unik.

Pada baris 17-20, mengatur lebar, tinggi, skala gambar, dan constraint posisinya di pojok kiri atas layout.

Pada baris 22, <TextView ...> mendeklarasikan TextView untuk judul film.

Pada baris 23, android:id="@+id/tv_movie_title" memberikan ID unik.

Pada baris 24-29, mengatur lebar, tinggi, margin, gaya teks, ukuran, dan constraint posisinya di kanan poster.

Pada baris 31, <TextView ...> mendeklarasikan TextView untuk tanggal rilis.

Pada baris 32, android:id="@+id/tv_release_date" memberikan ID unik.

Pada baris 33-38, mengatur lebar, tinggi, margin, ukuran teks, dan constraint posisinya di bawah judul.

Pada baris 40, <TextView ...> mendeklarasikan TextView untuk rata-rata voting/rating.

Pada baris 41, android:id="@+id/tv_vote_average" memberikan ID unik.

Pada baris 42-47, mengatur lebar, tinggi, margin, ukuran teks, dan constraint posisinya di bawah tanggal rilis.

Pada baris 49, <TextView ...> mendeklarasikan TextView untuk ringkasan (overview) film.

Pada baris 50, android:id="@+id/tv_overview" memberikan ID unik.

Pada baris 51-56, mengatur lebar, tinggi, margin, batas baris (maxLines), elipsis jika teks terlalu panjang, dan constraint posisinya di bawah poster.

Pada baris 58, <Button ...> mendeklarasikan tombol "Detail".

Pada baris 59, android:id="@+id/btn_detail" memberikan ID unik.

Pada baris 60-63, mengatur lebar, tinggi, margin, teks tombol, dan constraint posisinya di pojok kanan

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 detail film/serial yang mungkin panjang.

Pada baris 3-5, xmlns:android, xmlns:app, dan xmlns:tools mendeklarasikan namespace.

Pada baris 6, android:layout_width="match_parent" dan android:layout_height="match_parent" membuat ScrollView mengisi seluruh layar.

Pada baris 7, tools:context=".presentation.ui.activity.DetailActivity" adalah atribut tools untuk Android Studio.

Pada baris 9, <androidx.constraintlayout.widget.ConstraintLayout ...> adalah layout di dalam ScrollView, digunakan untuk mengatur posisi elemen-elemen detail.

Pada baris 10, android:padding="16dp" menambahkan padding di dalam ConstraintLayout.

Pada baris 12, <ImageView ...> mendeklarasikan ImageView untuk menampilkan poster detail.

Pada baris 13, android:id="@+id/iv_detail_poster" memberikan ID unik.

Pada baris 14-18, mengatur lebar (0dp mengisi parent), tinggi, skala, dan constraint posisinya di bagian atas layout.

Pada baris 20, <TextView ...> mendeklarasikan TextView untuk judul film.

Pada baris 21, android:id="@+id/tv_detail_title" memberikan ID unik.

Pada baris 22-27, mengatur lebar, tinggi, margin, ukuran teks, gaya teks, dan constraint posisinya di bawah poster.

Pada baris 29, <TextView ...> mendeklarasikan TextView untuk tanggal rilis.

Pada baris 30, android:id="@+id/tv_detail_release_date" memberikan ID unik.

Pada baris 31-36, mengatur lebar, tinggi, margin, ukuran teks, dan constraint posisinya di bawah judul.

Pada baris 38, <TextView ...> mendeklarasikan TextView untuk rata-rata voting/rating.

Pada baris 39, android:id="@+id/tv_detail_vote_average" memberikan ID unik.

Pada baris 40-45, mengatur lebar, tinggi, margin, ukuran teks, dan constraint posisinya di bawah tanggal rilis.

Pada baris 47, <TextView ...> mendeklarasikan TextView sebagai label "Overview:".

Pada baris 48, android:id="@+id/tv_detail_overview_label" memberikan ID unik.

Pada baris 49-54, mengatur lebar, tinggi, margin, teks, ukuran teks, gaya teks, dan constraint posisinya di bawah rating.

Pada baris 56, <TextView ...> mendeklarasikan TextView untuk teks overview sebenarnya.

Pada baris 57, android:id="@+id/tv_detail_overview" memberikan ID unik.

Pada baris 58-63, mengatur lebar, tinggi, margin, ukuran teks, dan constraint posisinya di bawah label overview.

Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/Easydaf/Praktikum_Mobile/tree/main/Modul4