LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Alya Rosaan NIM. 2310817320006

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE
MODUL 5

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

Nama Praktikan : Alya Rosaan
NIM : 2310817320006

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Salsabila Syifa NIM. 2010817320004	Andreyan Rizky Baskara, S.Kom., M.Kom. NIP. 19930703 201903 01 011

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Soal Praktikum

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 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.

Jetpack Compose

A. Source Code

Ui/MainActivity.kt

```
1   package com.example.myanimelistapp.ui
2
3   import android.content.Intent
4   import android.net.Uri
5   import android.os.Bundle
6   import androidx.activity.ComponentActivity
7   import androidx.activity.compose.setContent
8   import androidx.compose.foundation.layout.*
9   import androidx.compose.foundation.lazy.LazyColumn
10  import androidx.compose.foundation.lazy.items
11  import androidx.compose.foundation.shape.RoundedCornerShape
12  import androidx.compose.material3.*
13  import androidx.compose.runtime.Composable
14  import androidx.compose.runtime.collectAsState
15  import androidx.compose.runtime.getValue
16  import androidx.compose.ui.Alignment
17  import androidx.compose.ui.Modifier
18  import androidx.compose.ui.draw.clip
19  import androidx.compose.ui.graphics.Color
20  import androidx.compose.ui.layout.ContentScale
```

```
21  import androidx.compose.ui.platform.LocalContext
22  import androidx.compose.ui.text.font.FontWeight
23  import androidx.compose.ui.text.style.TextOverflow
24  import androidx.compose.ui.unit.dp
25  import androidx.lifecycle.viewmodel.compose.viewModel
26  import androidx.navigation.NavHostController
27  import androidx.navigation.NavType
28  import androidx.navigation.compose.NavHost
29  import androidx.navigation.compose.composable
30  import androidx.navigation.compose.rememberNavController
31  import androidx.navigation.navArgument
32  import coil.compose.AsyncImage
33  import com.example.myanimelistapp.MyApplication
34  import com.example.myanimelistapp.data.model.Movie
35  import com.example.myanimelistapp.ui.theme.MyAnimeListAppTheme
36  import com.example.myanimelistapp.viewmodel.MovieUiState
37  import com.example.myanimelistapp.viewmodel.MovieViewModel
38  import com.example.myanimelistapp.viewmodel.ViewModelFactory
39  import java.net.URLEncoder
40  import java.nio.charset.StandardCharsets
41
42  class MainActivity : ComponentActivity() {
43      override fun onCreate(savedInstanceState: Bundle?) {
44          super.onCreate(savedInstanceState)
45          val factory = ViewModelFactory((application as
46  MyApplication).repository)
47          setContent {
48              MyAnimeListAppTheme(darkTheme = false) {
49                  Surface(color = MaterialTheme.colorScheme.background)
50  {
51                      MovieApp(factory = factory)
52                  }
53              }
54          }
55      }
56  }
57
58  @Composable
59  fun MovieApp(factory: ViewModelFactory) {
60      val navController = rememberNavController()
61      NavHost(navController = navController, startDestination = "list")
62  {
63          composable("list") {
64              ItemList(navController = navController, factory =
    factory)
65          }
66          composable(
67              route = "detail/{title}/{overview}/{posterPath}",
68              arguments = listOf(
69                  navArgument("title") { type = NavType.StringType },
70                  navArgument("overview") { type = NavType.StringType
71  },
```

```
72                  navArgument("posterPath") { type = NavType.StringType
73  }
74              )
75          ) { backStackEntry ->
76              val title = backStackEntry.arguments?.getString("title")
    ?: ""
77              val overview =
    backStackEntry.arguments?.getString("overview") ?: ""
78              val posterPath =
    backStackEntry.arguments?.getString("posterPath") ?: ""
79              DetailScreen(title = title, detail = overview, imageUrl =
    posterPath)
80          }
81      }
82  }
83
84  @Composable
85  fun ItemList(
86      navController: NavHostController,
87      factory: ViewModelFactory
88  ) {
89      val viewModel: MovieViewModel = viewModel(factory = factory)
90      val uiState by viewModel.uiState.collectAsState()
91
92      when (val state = uiState) {
93          is MovieUiState.Loading -> {
94              Box(modifier = Modifier.fillMaxSize(), contentAlignment =
    Alignment.Center) {
95                  CircularProgressIndicator()
96              }
97          }
98          is MovieUiState.Success -> {
99              LazyColumn(modifier = Modifier.padding(8.dp)) {
100                 items(state.movies) { movie ->
101                     // Memanggil MovieCard yang sudah disesuaikan
102                     MovieCard(movie = movie, navController =
    navController)
103                 }
104             }
105         }
106         is MovieUiState.Error -> {
107             Box(modifier = Modifier.fillMaxSize(), contentAlignment =
    Alignment.Center) {
108                 Text(text = "Gagal memuat data: ${state.message}")
109             }
110         }
111     }
112 }
113
114 @Composable
115 fun MovieCard(movie: Movie, navController: NavHostController) {
116     val context = LocalContext.current
```

```
117
118     Card(
119         colors = CardDefaults.cardColors(
120             containerColor = Color(0xFFF0F0F0)
121         ),
122         shape = RoundedCornerShape(16.dp),
123         modifier = Modifier
124             .padding(8.dp)
125             .fillMaxWidth()
126     ) {
127         Row(modifier = Modifier.padding(16.dp)) {
128             // Menggunakan AsyncImage untuk memuat gambar dari URL
                AsyncImage(
129                 model =
    "https://image.tmdb.org/t/p/w500${movie.posterPath}",
130                 contentDescription = movie.title,
131                 contentScale = ContentScale.Crop,
132                 modifier = Modifier
133                     .width(150.dp)
134                     .height(250.dp)
135                     .clip(RoundedCornerShape(12.dp))
136             )
137
138             Spacer(modifier = Modifier.width(16.dp))
139
140             Column(
141                 modifier = Modifier
142                     .weight(1f)
143                     .fillMaxHeight(), // Mengisi tinggi yang tersisa
144                 // Mengatur jarak antar elemen di dalam Column ini
                    verticalArrangement = Arrangement.SpaceBetween
145             ) {
146                 // Kolom untuk Teks (Judul dan Overview)
147                 Column {
148                     Text(
149                         text = movie.title,
150                         style = MaterialTheme.typography.titleLarge,
                            fontWeight = FontWeight.Bold
151                     )
152                     Spacer(modifier = Modifier.height(8.dp))
153                     Text(
154                         text = movie.overview, // Menggunakan
    overview sebagai teks utama
155                         style = MaterialTheme.typography.bodyMedium,
156                         maxLines = 5, // Bisa disesuaikan
157                         overflow = TextOverflow.Ellipsis
158                     )
160                 }
161
162                 // Kolom untuk Tombol (agar tetap di bawah)
163                 Column(verticalArrangement =
164 Arrangement.spacedBy(8.dp)) {
```

```
                    Button(
                        onClick = {
                            val encodedPosterPath =
URLEncoder.encode(movie.posterPath ?: "",
StandardCharsets.UTF_8.toString())
                            // Navigasi dengan data dari API

navController.navigate("detail/${movie.title}/${movie.overview}/$enco
dedPosterPath")
                        },
                        colors = ButtonDefaults.buttonColors(
                            containerColor = Color(0xFF3A4C8B),
                            contentColor = Color.White
                        ),
                        modifier = Modifier.fillMaxWidth() // Dibuat
fill width agar rapi
                    ) {
                        Text("Detail", style =
MaterialTheme.typography.labelLarge)
                    }

                    Button(
                        onClick = {
                            val url =
"https://www.themoviedb.org/search?query=${URLEncoder.encode(movie.ti
tle, StandardCharsets.UTF_8.toString())}"
                            val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(url))
                            context.startActivity(intent)
                        },
                        colors = ButtonDefaults.buttonColors(
                            containerColor = Color(0xFF3A4C8B),
                            contentColor = Color.White
                        ),
                        modifier = Modifier.fillMaxWidth()
                    ) {
                        Text("Open URL", style =
MaterialTheme.typography.labelLarge)
                    }
                }
            }
        }
    }
}


// DetailScreen tidak perlu diubah, biarkan seperti ini
@Composable
fun DetailScreen(title: String, detail: String, imageUrl: String) {
    Column(
        modifier = Modifier
            .fillMaxSize()
```

```
209             .padding(16.dp),
210         horizontalAlignment = Alignment.CenterHorizontally
211     ) {
212         AsyncImage(
213             model = "https://image.tmdb.org/t/p/w500${imageUrl}",
214             contentDescription = title,
215             contentScale = ContentScale.Fit,
216             modifier = Modifier
217                 .fillMaxWidth()
218                 .height(400.dp)
219                 .clip(RoundedCornerShape(12.dp))
220         )
221         Spacer(modifier = Modifier.height(16.dp))
222         Text(text = title, style =
223 MaterialTheme.typography.headlineMedium, fontWeight =
    FontWeight.Bold)
224         Spacer(modifier = Modifier.height(8.dp))
225         Text(text = detail, style =
    MaterialTheme.typography.bodyLarge)
226     }
227 }
228
```

Table 1 Ui/MainActivity.kt

data/local/AppDatabase.kt

```
1   package com.example.myanimelistapp.data.local
2
3   import androidx.room.Database
4   import androidx.room.RoomDatabase
5
6   @Database(entities = [MovieEntity::class], version = 1)
7   abstract class AppDatabase : RoomDatabase() {
8       abstract fun movieDao(): MovieDao
9   }
```

Table 2 Source Code data/local/AppDatabase.kt

data/local/MovieDao.kt

```
1   package com.example.myanimelistapp.data.local
2
3   import androidx.room.Dao
4   import androidx.room.Insert
5   import androidx.room.OnConflictStrategy
6   import androidx.room.Query
7   import kotlinx.coroutines.flow.Flow
8
9   @Dao
10  interface MovieDao {
11      @Insert(onConflict = OnConflictStrategy.REPLACE)
```

```
12      suspend fun insertMovies(movies: List<MovieEntity>)
13
14      @Query("SELECT * FROM movies")
15      fun getAllMovies(): Flow<List<MovieEntity>> // Gunakan Flow agar
    UI update otomatis
16
17      @Query("DELETE FROM movies")
18      suspend fun deleteAllMovies()
19  }
```

Table 3 Source Code data/local/MovieDao.kt

data/local/MovieEntity.kt

```
1   package com.example.myanimelistapp.data.local
2
3   import androidx.room.Entity
4   import androidx.room.PrimaryKey
5
6   @Entity(tableName = "movies")
7   data class MovieEntity(
8       @PrimaryKey val id: Int,
9       val title: String,
10      val posterPath: String?,
11      val overview: String
12  )
```

Table 4 Source Code data/local/MovieEntity.kt

data/model/Movie.kt

```
1   package com.example.myanimelistapp.data.model
2
3   import kotlinx.serialization.SerialName
4   import kotlinx.serialization.Serializable
5
6   @Serializable
7   data class Movie(
8       val id: Int,
9       val title: String,
10      @SerialName("poster_path")
11      val posterPath: String?,
12      @SerialName("overview")
13      val overview: String
14  )
```

Table 5 Source Code data/model/Movie.kt

data/model/MovieResponse.kt

```
1   package com.example.myanimelistapp.data.model
2
3   import kotlinx.serialization.Serializable
```

```
4
5   @Serializable
6   data class MovieResponse(
7       val page: Int,
8       val results: List<Movie>
9   )
```

Table 6 Source Code data/model/Movie.kt

network/MovieApiService.kt

```
1   package com.example.myanimelistapp.network
2
3   import com.example.myanimelistapp.data.model.MovieResponse
4   import retrofit2.http.GET
5   import retrofit2.http.Query
6
7   interface MovieApiService {
8       // mengambil film populer
9       @GET("movie/popular")
10      suspend fun getPopularMovies(
11          @Query("api_key") apiKey: String
12      ): MovieResponse
13  }
```

Table 7 Source Code network/MovieApiService.kt

network/Retrofitinstance.kt

```
1   package com.example.myanimelistapp.network
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 retrofit2.Retrofit
9
10  object RetrofitInstance {
11      private const val BASE_URL = "https://api.themoviedb.org/3/"
12
13      private val json = Json {
14          ignoreUnknownKeys = true
15      }
16
17      val api: MovieApiService by lazy {
18          Retrofit.Builder()
19              .baseUrl(BASE_URL)
20
21  .addConverterFactory(json.asConverterFactory("application/json".toMedi
22  aType()))
23              .build()
```

```
            .create(MovieApiService::class.java)
    }
}

```

Table 8 Source Code network/Retrofitinstance.kt

Repository/MovieRepository.kt

```
package com.example.myanimelistapp.repository

import com.example.myanimelistapp.data.local.MovieDao
import com.example.myanimelistapp.data.local.MovieEntity
import com.example.myanimelistapp.data.model.Movie
import com.example.myanimelistapp.network.MovieApiService
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.flow

class MovieRepository(
    private val apiService: MovieApiService,
    private val movieDao: MovieDao,
    private val apiKey: String
) {
    fun getPopularMovies(): Flow<Result<List<Movie>>> = flow {
        emit(Result.Loading)

        // 1. Ambil data dari cache dulu
        val cachedMovies = movieDao.getAllMovies().first() // ambil data saat ini
        emit(Result.Success(cachedMovies.map { it.toMovie() })) // map dari Entity ke Model

        try {
            // 2. Ambil data dari network
            val response = apiService.getPopularMovies(apiKey)
            val moviesFromApi = response.results

            // 3. Hapus cache lama dan simpan data baru
            movieDao.deleteAllMovies()
            movieDao.insertMovies(moviesFromApi.map { it.toMovieEntity() })

        } catch (e: Exception) {
            // 4. Jika network gagal, emit error. UI tetap punya data dari cache.
            emit(Result.Error("Gagal mengambil data dari jaringan: ${e.message}"))
        }
    }
}

```

```
// Tambahkan fungsi mapping di suatu tempat
fun MovieEntity.toMovie() = Movie(id, title, posterPath, overview)
fun Movie.toMovieEntity() = MovieEntity(id, title, posterPath,
overview)
```

Table 9 Source Code Repository/MovieRepository.kt

network/Result.kt

```
package com.example.myanimelistapp.repository

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

Table 10 Source Code network/Result.kt

viewmodel/MovieViewModel.kt

```
package com.example.myanimelistapp.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.myanimelistapp.data.model.Movie
import com.example.myanimelistapp.repository.MovieRepository
import com.example.myanimelistapp.repository.Result
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.catch
import kotlinx.coroutines.launch

class MovieViewModel(private val repository: MovieRepository) :
ViewModel() {

    private val _uiState =
MutableStateFlow<MovieUiState>(MovieUiState.Loading)
    val uiState: StateFlow<MovieUiState> = _uiState

    init {
        fetchPopularMovies()
    }

    private fun fetchPopularMovies() {
        viewModelScope.launch {
            repository.getPopularMovies()
                .catch { e -> _uiState.value =
MovieUiState.Error(e.message ?: "Unknown Error") }
                .collect { result ->
                    when (result) {
                        is Result.Loading -> _uiState.value =
```

```
    MovieUiState.Loading
30                          is Result.Success -> _uiState.value =
    MovieUiState.Success(result.data)
31                          is Result.Error -> _uiState.value =
    MovieUiState.Error(result.message)
32                      }
33                  }
34          }
35      }
36  }
37
38  sealed interface MovieUiState {
39      data object Loading : MovieUiState
40      data class Success(val movies: List<Movie>) : MovieUiState
41      data class Error(val message: String) : MovieUiState
42  }
```

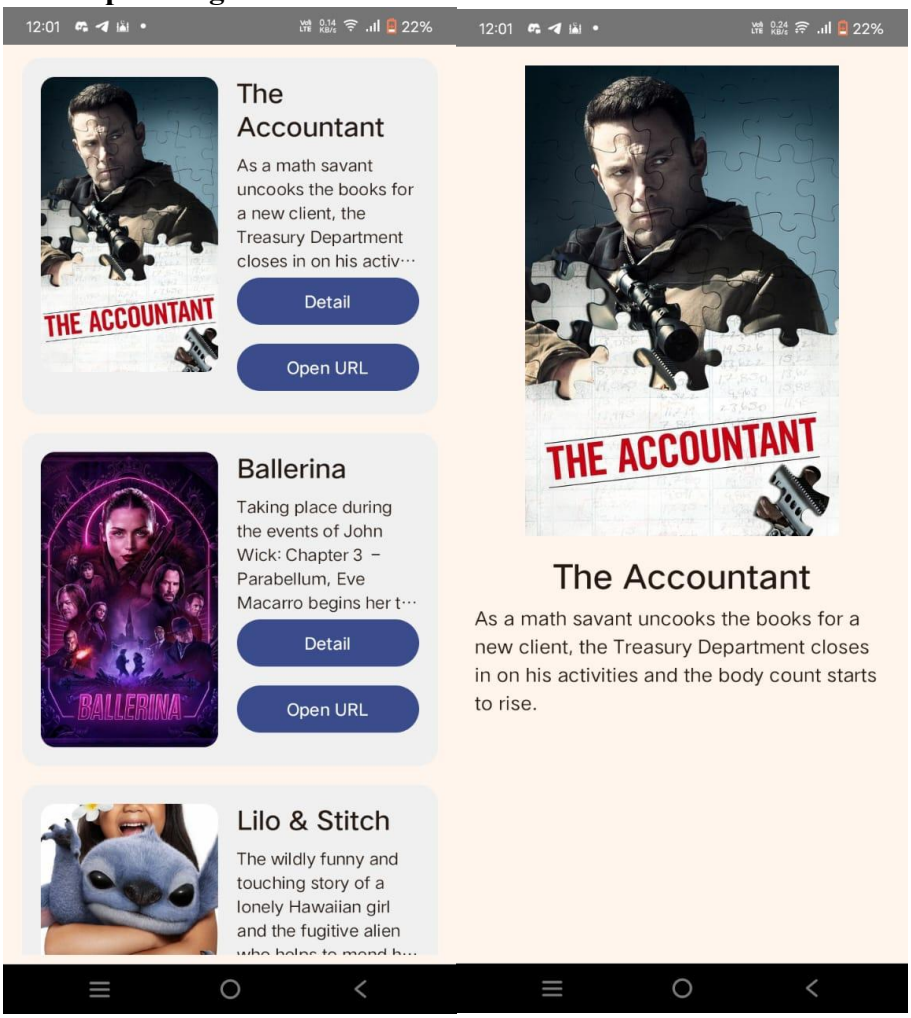
Table 11 Source Code viewmodel/MovieViewModel.kt

viewmodel/MovieViewFactory.kt

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

Table 12 Source Code viewmodel/MovieViewFactory.kt

B. Output Program



Gambar 1 Screenshot Output Compose

C. Pembahasan

MainActivity.kt:

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.ui Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.ui, yang membantu mengorganisir struktur proyek.
2. **Baris 7–42:** import ... Bagian ini mengimpor semua kelas dan fungsi yang diperlukan dari *library* eksternal agar bisa digunakan di dalam file ini.
   - android.content.Intent, android.net.Uri: Untuk membuat *intent* yang membuka URL di browser.
   - androidx.activity.*: Komponen dasar untuk Activity, terutama setContent untuk menggunakan Jetpack Compose.
   - androidx.compose.*: Kumpulan *library* inti Jetpack Compose untuk membangun UI, seperti foundation.layout (untuk Row, Column, Spacer), foundation.lazy (untuk

LazyColumn), material3 (untuk Card, Button, Text), dan runtime (untuk manajemen *state* seperti collectAsState).

- androidx.lifecycle.viewmodel.compose.viewModel: Untuk mengintegrasikan ViewModel dengan UI Compose.
- androidx.navigation.*: *Library* Navigation Compose untuk perpindahan antar layar (NavHost, composable, rememberNavController).
- coil.compose.AsyncImage: *Library* Coil untuk memuat gambar dari URL secara asinkron.
- com.example.myanimelistapp.*: Mengimpor kelas-kelas dari proyek ini sendiri, seperti MyApplication, Movie (model data), ViewModelFactory, dan MovieViewModel.
- java.net.URLEncoder, java.nio.charset.StandardCharsets: Untuk melakukan *encoding* pada string agar aman digunakan dalam URL.

Bagian: MainActivity

3. **Baris 44:** class MainActivity : ComponentActivity() Mendefinisikan kelas MainActivity sebagai kelas utama aplikasi yang akan diluncurkan. Kelas ini mewarisi ComponentActivity, yang merupakan basis untuk aplikasi yang menggunakan Jetpack Compose.
4. **Baris 45:** override fun onCreate(savedInstanceState: Bundle?) Fungsi ini adalah titik masuk utama Activity, yang dieksekusi saat Activity pertama kali dibuat.
5. **Baris 47:** val factory = ViewModelFactory(...) Membuat sebuah instance dari ViewModelFactory. *Factory* ini digunakan untuk menyediakan ViewModel dengan dependensi yang diperlukannya (dalam hal ini, repository), yang diambil dari kelas MyApplication.
6. **Baris 48:** setContent { ... } Fungsi ini adalah jembatan antara *framework* UI Android tradisional dan Jetpack Compose. Semua UI yang didefinisikan di dalam blok lambda ini akan ditampilkan di layar.
7. **Baris 49:** MyAnimeListAppTheme(darkTheme = false) Menerapkan tema kustom aplikasi (MyAnimeListAppTheme). darkTheme = false secara eksplisit mengaktifkan mode terang.
8. **Baris 50:** Surface(color = MaterialTheme.colorScheme.background) Membuat sebuah Surface (bidang gambar) yang menutupi seluruh layar dan memberinya warna latar belakang standar dari tema Material 3.
9. **Baris 51:** MovieApp(factory = factory) Memanggil *composable* utama aplikasi, yaitu MovieApp, dan meneruskan ViewModelFactory yang telah dibuat sebelumnya sebagai parameter.

Bagian: MovieApp (Navigasi)

10. **Baris 59:** @Composable fun MovieApp(factory: ViewModelFactory) Mendefinisikan fungsi *composable* MovieApp yang bertanggung jawab untuk mengatur struktur navigasi aplikasi.

11. **Baris 60:** val navController = rememberNavController() Membuat dan "mengingat" sebuah NavHostController. Objek ini berfungsi sebagai pengontrol utama untuk semua operasi navigasi antar layar.
12. **Baris 61:** NavHost(navController = navController, startDestination = "list") Menginisialisasi NavHost, yang merupakan kontainer untuk semua tujuan (layar) navigasi. startDestination = "list" menetapkan bahwa layar dengan rute "list" akan menjadi layar pertama yang ditampilkan.
13. **Baris 62:** composable("list") { ... } Mendefinisikan tujuan navigasi pertama dengan rute "list". Saat aplikasi menavigasi ke rute ini, konten di dalam blok lambda akan ditampilkan.
14. **Baris 63:** ItemList(navController = navController, factory = factory) Menampilkan *composable* ItemList untuk rute "list". navController dan factory diteruskan agar ItemList dapat menavigasi ke layar lain dan membuat ViewModel.
15. **Baris 65–72:** composable(route = "detail/{title}/{overview}/{posterPath}", ...) Mendefinisikan tujuan navigasi kedua dengan rute "detail". Rute ini dinamis dan menerima tiga argumen: title, overview, dan posterPath. Tipe data untuk setiap argumen didefinisikan sebagai NavType.StringType.
16. **Baris 73:** backStackEntry -> Lambda ini menyediakan akses ke NavBackStackEntry, yang berisi informasi tentang rute dan argumen yang sedang aktif.
17. **Baris 74–76:** val title = backStackEntry.arguments?.getString(...) ?: "" Mengambil nilai argumen title, overview, dan posterPath dari backStackEntry. Operator ?: "" memberikan nilai default (string kosong) jika argumen tidak ditemukan (null).
18. **Baris 77:** DetailScreen(title = title, detail = overview, imageUrl = posterPath) Memanggil *composable* DetailScreen dan meneruskan data yang telah diambil dari argumen navigasi untuk ditampilkan.

Bagian: ItemList (Layar Daftar Film)

19. **Baris 89:** val viewModel: MovieViewModel = viewModel(factory = factory) Mengambil instance dari MovieViewModel. Fungsi viewModel() secara cerdas akan membuat ViewModel baru atau memberikan instance yang sudah ada, sehingga data tetap terjaga saat terjadi perubahan konfigurasi (misalnya rotasi layar).
20. **Baris 90:** val uiState by viewModel.uiState.collectAsState() Berlangganan (subscribe) ke StateFlow bernama uiState dari ViewModel. collectAsState mengubah Flow ini menjadi State Compose. Akibatnya, setiap kali uiState di ViewModel diperbarui, UI akan secara otomatis diperbarui (recompose).
21. **Baris 92:** when (val state = uiState) Struktur when (mirip switch di bahasa lain) untuk menampilkan UI yang berbeda berdasarkan kondisi uiState saat ini: Loading, Success, atau Error.
22. **Baris 93–97:** is MovieUiState.Loading -> { ... } Jika state adalah Loading, maka sebuah CircularProgressIndicator (ikon pemuatan berputar) akan ditampilkan di tengah layar.
23. **Baris 98–105:** is MovieUiState.Success -> { ... } Jika state adalah Success (data berhasil dimuat), maka sebuah LazyColumn akan ditampilkan. LazyColumn adalah daftar yang dapat di-scroll secara efisien karena hanya merender item yang terlihat di layar.

24. **Baris 100:** items(state.movies) { movie -> ... } Fungsi ini mengiterasi daftar movies dari state dan membuat UI untuk setiap movie di dalamnya.
25. **Baris 102:** MovieCard(movie = movie, navController = navController) Untuk setiap item film, *composable* MovieCard dipanggil untuk menampilkan detailnya dalam bentuk kartu.
26. **Baris 106–110:** is MovieUiState.Error -> { ... } Jika state adalah Error, sebuah pesan teks yang berisi informasi kesalahan akan ditampilkan di tengah layar.

Bagian: MovieCard (Kartu Item Film)

27. **Baris 117:** val context = LocalContext.current Mendapatkan Context Android saat ini, yang diperlukan untuk memulai aktivitas baru seperti membuka browser.
28. **Baris 119–126:** Card(...) Membuat komponen Card dari Material Design yang berfungsi sebagai kontainer untuk setiap item film. Kartu ini diberi warna latar abu-abu muda, sudut membulat, dan padding.
29. **Baris 127:** Row(modifier = Modifier.padding(16.dp)) Mengatur elemen-elemen di dalamnya (gambar dan kolom teks) dalam tata letak horizontal.
30. **Baris 130–138:** AsyncImage(...) Menggunakan AsyncImage dari *library* Coil untuk memuat dan menampilkan gambar poster film dari URL. URL lengkapnya dibentuk dengan menggabungkan URL dasar TMDb dan movie.posterPath.
31. **Baris 140:** Spacer(modifier = Modifier.width(16.dp)) Membuat jarak horizontal selebar 16.dp antara gambar dan kolom teks.
32. **Baris 142–150:** Column(...) Membuat kolom vertikal untuk menampung teks dan tombol. modifier = Modifier.weight(1f) membuat kolom ini mengisi sisa ruang horizontal yang tersedia. verticalArrangement = Arrangement.SpaceBetween menyebarkan elemen di dalamnya (teks di atas, tombol di bawah) secara merata di sepanjang sumbu vertikal.
33. **Baris 153–157:** Text(text = movie.title, ...) Menampilkan judul film dengan gaya teks tebal.
34. **Baris 159–164:** Text(text = movie.overview, ...) Menampilkan sinopsis (overview) film. maxLines = 5 dan overflow = TextOverflow.Ellipsis memastikan teks yang terlalu panjang akan dipotong dan diakhiri dengan "...".
35. **Baris 168–184:** Button(...) (Tombol Detail) Membuat tombol "Detail".
36. **Baris 170–174:** onClick mendefinisikan aksi saat tombol diklik: URLEncoder.encode digunakan untuk mengamankan posterPath agar valid sebagai bagian dari URL, lalu navController.navigate memicu perpindahan ke layar detail sambil membawa data film.
37. **Baris 186–205:** Button(...) (Tombol Open URL) Membuat tombol "Open URL".
38. **Baris 188–192:** onClick mendefinisikan aksi: membangun URL pencarian di situs The Movie Database, membuat Intent untuk membuka URL tersebut, dan menggunakan context.startActivity untuk meluncurkan browser.

Bagian: DetailScreen (Layar Detail)

39. **Baris 217:** @Composable fun DetailScreen(...) Mendefinisikan *composable* untuk layar detail yang hanya bertugas menampilkan data yang diterimanya melalui parameter.
40. **Baris 218–223:** Column(...) Mengatur semua elemen (gambar, judul, detail) dalam tata letak vertikal di tengah layar.

41. **Baris 224–232:** AsyncImage(...) Memuat dan menampilkan gambar poster film dalam ukuran yang lebih besar.
42. **Baris 234:** Text(...) Menampilkan judul film dengan gaya tipografi yang lebih besar dan tebal.
43. **Baris 236:** Text(...) Menampilkan teks detail atau sinopsis lengkap dari film tersebut

data/local/AppDatabase.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.data.local Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.data.local, yang biasanya digunakan untuk menempatkan semua kode yang berkaitan dengan sumber data lokal seperti database.
2. **Baris 7–8:** import ... Mengimpor kelas-kelas yang diperlukan dari *library* Room Persistence.
   - androidx.room.Database: Anotasi utama untuk menandai sebuah kelas sebagai database Room.
   - androidx.room.RoomDatabase: Kelas dasar yang harus di-extend oleh setiap kelas database Room.

Bagian: Definisi Database Room

3. **Baris 10:** @Database(entities = [MovieEntity::class], version = 1) Anotasi @Database yang mengkonfigurasi kelas ini sebagai database.
   - entities = [MovieEntity::class]: Parameter ini memberitahu Room tabel apa saja yang akan ada di dalam database ini. Dalam kasus ini, hanya ada satu tabel, yang strukturnya didefinisikan oleh kelas MovieEntity.
   - version = 1: Parameter ini menentukan versi skema database. Nomor versi ini sangat penting untuk proses migrasi. Jika Anda mengubah struktur tabel (misalnya, menambah kolom), Anda harus menaikkan nomor versi ini dan menyediakan strategi migrasi.
4. **Baris 11:** abstract class AppDatabase : RoomDatabase() Mendefinisikan AppDatabase sebagai kelas *abstract* yang mewakili seluruh database aplikasi.
5. Kelas ini harus *abstract* karena Room akan secara otomatis membuat implementasi konkretnya di belakang layar saat proses kompilasi.
6. Kelas ini harus mewarisi (meng-extend) RoomDatabase agar mendapatkan fungsionalitas dasar dari Room.
7. **Baris 12:** abstract fun movieDao(): MovieDao Mendefinisikan sebuah fungsi *abstract* yang akan menyediakan akses ke DAO (*Data Access Object*).
   a. DAO adalah antarmuka (interface) yang berisi metode-metode untuk berinteraksi dengan tabel di database (seperti insert, query, delete).
   b. Dengan mendeklarasikan fungsi ini, Anda memberitahu Room bahwa database ini memiliki MovieDao. Room akan mengurus implementasi fungsi

ini untuk menyediakan instance MovieDao yang bisa digunakan oleh bagian lain dari aplikasi untuk mengakses tabel film.

data/local/MovieDao.kt

Bagian: Package dan Imports

1. **Baris 1:** package com.example.myanimelistapp.data.local Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.data.local, tempat kode-kode untuk akses data lokal (database) dikelompokkan.
2. **Baris 4–8:** import ... Mengimpor anotasi dan kelas yang diperlukan dari *library* Room dan Kotlin Coroutines.
    - androidx.room.Dao: Anotasi untuk menandai sebuah *interface* sebagai **D**ata **A**ccess **O**bject.
    - androidx.room.Insert: Anotasi untuk fungsi yang bertugas memasukkan data ke dalam tabel.
    - androidx.room.OnConflictStrategy: Menyediakan berbagai strategi untuk menangani konflik data (misalnya, jika data yang dimasukkan memiliki *primary key* yang sama dengan data yang sudah ada).
    - androidx.room.Query: Anotasi untuk fungsi yang menjalankan kueri SQL untuk membaca data dari database.
    - kotlinx.coroutines.flow.Flow: Tipe data dari Kotlin Coroutines yang memungkinkan observasi perubahan data secara *real-time* atau reaktif.

Bagian: Definisi DAO (Data Access Object)

3. **Baris 11:** @Dao Anotasi ini memberitahu Room bahwa MovieDao adalah sebuah *interface* DAO. Room akan menggunakan *interface* ini untuk secara otomatis membuat kode implementasi yang diperlukan untuk semua operasi database yang didefinisikan di dalamnya.
4. **Baris 12:** interface MovieDao { Mendefinisikan MovieDao sebagai sebuah *interface*. Di dalam *interface* inilah semua metode untuk mengakses tabel movie (seperti menambah, membaca, atau menghapus data) akan dideklarasikan.

Bagian: Operasi Insert (Menambahkan Data)

5. **Baris 13:** @Insert(onConflict = OnConflictStrategy.REPLACE) Ini adalah anotasi untuk fungsi yang akan didefinisikan di baris berikutnya.
    - @Insert: Menandakan bahwa fungsi ini adalah untuk operasi **insert** (memasukkan data).
    - onConflict = OnConflictStrategy.REPLACE: Ini adalah strategi penanganan konflik. Artinya, jika Anda mencoba memasukkan data film dengan *primary key* yang sama dengan film yang sudah ada di dalam tabel, Room akan secara otomatis **mengganti** (REPLACE) data lama dengan data yang baru. Ini sangat berguna untuk menyinkronkan data dari API ke database lokal.

data/local/MovieEntity.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.data.local Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.data.local, yang merupakan lokasi umum untuk kelas-kelas yang berhubungan dengan data lokal seperti entitas database.
2. **Baris 7–8:** import ... Mengimpor anotasi yang diperlukan dari *library* Room.
    - androidx.room.Entity: Anotasi untuk menandai sebuah kelas data sebagai sebuah **entitas**, yang merepresentasikan sebuah tabel dalam database.
    - androidx.room.PrimaryKey: Anotasi untuk menandai sebuah properti di dalam entitas sebagai **kunci utama** (primary key) dari tabel.

Bagian: Definisi Entitas (Tabel Database)

3. **Baris 10:** @Entity(tableName = "movies") Anotasi @Entity memberitahu Room bahwa kelas ini adalah sebuah cetak biru untuk tabel di database.
    - tableName = "movies": Parameter ini secara eksplisit menamai tabel di dalam database sebagai "movies". Jika parameter ini tidak diberikan, Room akan menggunakan nama kelas (MovieEntity) sebagai nama tabel secara default.
4. **Baris 11:** data class MovieEntity(Mendefinisikan sebuah **data class** Kotlin bernama MovieEntity.
    - data class sangat cocok untuk menjadi entitas Room karena kelas ini secara otomatis menghasilkan fungsi-fungsi standar (seperti equals(), hashCode(), toString()) yang berguna untuk mengelola objek data.
    - Setiap properti (val) yang didefinisikan di dalam data class ini akan menjadi sebuah **kolom** di dalam tabel "movies".

Bagian: Properti Kelas (Kolom Tabel)

5. **Baris 12:** @PrimaryKey val id: Int, Mendefinisikan kolom pertama dari tabel.
    - @PrimaryKey: Anotasi ini menandai properti id sebagai **kunci utama** tabel. Ini berarti setiap baris data (setiap film) dalam tabel ini harus memiliki nilai id yang unik dan tidak boleh sama dengan baris lainnya.
    - val id: Int: Menentukan bahwa kolom ini akan bernama id dan memiliki tipe data INTEGER.
6. **Baris 13:** val title: String, Mendefinisikan kolom title dengan tipe data String. Di dalam database SQLite, ini akan menjadi tipe TEXT.
7. **Baris 14:** val posterPath: String?, Mendefinisikan kolom posterPath dengan tipe data String. Tanda tanya (?) setelah String menandakan bahwa kolom ini **bisa bernilai null**. Ini berguna jika ada data film yang tidak memiliki gambar poster.
8. **Baris 15:** val overview: String Mendefinisikan kolom overview dengan tipe data String (atau TEXT di SQLite).

data/model/Movie.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.data.model Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.data.model. Package ini biasanya digunakan untuk menyimpan kelas-kelas model data yang merepresentasikan struktur data dari sumber eksternal, seperti respons API.
2. **Baris 7–8:** import ... Mengimpor anotasi yang diperlukan dari *library* kotlinx.serialization.
   - kotlinx.serialization.SerialName: Anotasi untuk menentukan nama kunci (key) dalam format data (seperti JSON) yang berbeda dengan nama properti di kelas Kotlin.
   - kotlinx.serialization.Serializable: Anotasi untuk menandai sebuah kelas agar dapat diubah dari/ke format data lain (misalnya JSON) secara otomatis.

Bagian: Definisi Model Data

3. **Baris 10:** @Serializable Anotasi ini memberitahu *library* kotlinx.serialization bahwa kelas Movie ini dapat melalui proses **serialisasi** (mengubah objek Kotlin menjadi JSON) dan **deserialisasi** (mengubah JSON menjadi objek Kotlin). Ini sangat penting saat bekerja dengan API, misalnya saat menggunakan Ktor atau Retrofit.
4. **Baris 11:** data class Movie(Mendefinisikan Movie sebagai sebuah **data class** Kotlin. Kelas ini berfungsi sebagai model atau cetak biru untuk objek film yang diterima dari API. Setiap properti di dalamnya merepresentasikan sebuah *field* dari data JSON.

Bagian: Properti Kelas

5. **Baris 12:** val id: Int, Mendefinisikan properti id dengan tipe Int. *Library* serialisasi akan mencari kunci "id" dalam JSON untuk mengisi nilai properti ini.
6. **Baris 13:** val title: String, Mendefinisikan properti title dengan tipe String. *Library* akan mencari kunci "title" dalam JSON.
7. **Baris 14:** @SerialName("poster_path") Anotasi **@SerialName** ini adalah bagian yang sangat penting.
   - Fungsinya adalah untuk memetakan kunci dari JSON ke properti di kelas Kotlin.
   - Dalam kasus ini, respons JSON dari API memiliki kunci bernama "poster_path" (menggunakan format *snake_case*). Namun, konvensi penulisan kode di Kotlin adalah menggunakan *camelCase* (posterPath). Anotasi ini menjembatani perbedaan tersebut, sehingga *library* tahu bahwa nilai dari "poster_path" harus dimasukkan ke dalam properti posterPath.
8. **Baris 15:** val posterPath: String?, Mendefinisikan properti posterPath. Tanda tanya (?) menunjukkan bahwa properti ini bersifat *nullable*, artinya nilainya bisa saja tidak ada (null) dalam respons JSON.
9. **Baris 16:** @SerialName("overview") Sama seperti sebelumnya, anotasi ini memetakan kunci "overview" dari JSON ke properti overview di kelas ini.

10. **Baris 17:** val overview: String Mendefinisikan properti overview untuk menyimpan sinopsis film.

data/model/MovieResponse.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.data.model Menentukan bahwa file ini berada dalam package com.example.myanimelistapp.data.model, yang berisi kelas-kelas model data yang sesuai dengan struktur data dari API.
2. **Baris 7:** import kotlinx.serialization.Serializable Mengimpor anotasi Serializable dari *library* kotlinx.serialization, yang diperlukan untuk proses konversi otomatis dari JSON ke objek Kotlin.

Bagian: Definisi Model Respons API

3. **Baris 9:** @Serializable Anotasi ini menandai kelas MovieResponse agar dapat di-deserialisasi secara otomatis oleh *library* kotlinx.serialization. Artinya, ketika API memberikan respons dalam format JSON, *library* ini dapat mengubah seluruh respons tersebut menjadi sebuah objek MovieResponse.
4. **Baris 10:** data class MovieResponse(Mendefinisikan MovieResponse sebagai sebuah **data class**. Kelas ini merepresentasikan struktur **keseluruhan** dari respons yang dikirim oleh API. Seringkali, API tidak hanya mengirimkan daftar data, tetapi juga metadata tambahan seperti informasi halaman (*pagination*).

Bagian: Properti Kelas

5. **Baris 11:** val page: Int, Mendefinisikan properti page dengan tipe Int. Properti ini akan diisi dengan nilai dari kunci "page" yang ada di dalam JSON respons, yang biasanya menunjukkan nomor halaman dari data yang ditampilkan.
6. **Baris 12:** val results: List<Movie> Mendefinisikan properti results sebagai sebuah daftar (List) dari objek Movie.
    - Properti ini akan diisi dengan nilai dari kunci "results" di dalam JSON, yang mana nilainya adalah sebuah *array* dari objek-objek film.
    - Setiap objek di dalam *array* tersebut akan diubah menjadi objek dari kelas Movie (yang telah Anda definisikan sebelumnya). Dengan kata lain, kelas ini adalah "pembungkus" untuk daftar film yang sebenarnya.

network/MovieApiService.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.network Menentukan bahwa file ini berada dalam *package* com.example.myanimelistapp.network. *Package* ini umumnya berisi semua kode yang terkait dengan komunikasi jaringan, seperti antarmuka API dan klien HTTP.
2. **Baris 7–9:** import ... Mengimpor kelas dan anotasi yang diperlukan:

- com.example.myanimelistapp.data.model.MovieResponse: Mengimpor kelas model data yang merepresentasikan struktur respons JSON dari API.
- retrofit2.http.GET: Anotasi dari Retrofit untuk mendeklarasikan sebuah permintaan HTTP GET.
- retrofit2.http.Query: Anotasi dari Retrofit untuk menambahkan *query parameter* ke URL permintaan.

Bagian: Definisi Antarmuka API

3. **Baris 11:** interface MovieApiService { ... } Mendefinisikan sebuah **interface** bernama MovieApiService. Dalam Retrofit, Anda tidak menulis implementasi kode jaringan secara manual. Sebaliknya, Anda mendefinisikan permintaan dalam sebuah antarmuka, dan Retrofit akan secara otomatis membuat implementasi yang berfungsi di belakang layar.

Bagian: Definisi Endpoint

4. **Baris 13:** @GET("movie/popular") Anotasi **@GET** memberitahu Retrofit bahwa fungsi ini akan melakukan permintaan HTTP GET.
   String "movie/popular" adalah **jalur relatif (relative path)** untuk *endpoint* API. Jalur ini akan ditambahkan ke **URL dasar (base URL)** yang Anda konfigurasikan saat membuat klien Retrofit. Contoh: jika URL dasar adalah https://api.themoviedb.org/3/, maka URL lengkapnya menjadi https://api.themoviedb.org/3/movie/popular.
5. **Baris 14:** suspend fun getPopularMovies(Mendefinisikan sebuah fungsi.
   Kata kunci **suspend** menandakan bahwa ini adalah *suspending function*. Ini memungkinkan fungsi dipanggil dari dalam sebuah *coroutine* tanpa memblokir *thread* utama, sehingga UI aplikasi tetap responsif saat menunggu data dari jaringan.
6. **Baris 15:** @Query("api_key") apiKey: String Anotasi **@Query** digunakan untuk menambahkan *query parameter* ke URL.
   - @Query("api_key") berarti sebuah parameter bernama api_key akan ditambahkan ke URL.
   - Nilai dari parameter ini akan diambil dari argumen apiKey yang diberikan saat fungsi dipanggil. Contoh: jika Anda memanggil fungsi dengan apiKey bernilai "12345", maka URL akhirnya akan menjadi .../movie/popular?api_key=12345.
7. **Baris 16:**): MovieResponse Menentukan **tipe data kembalian (return type)** dari fungsi ini adalah MovieResponse. Retrofit akan secara otomatis mengambil respons JSON dari server dan mengubahnya (deserialisasi) menjadi objek Kotlin MovieResponse.

network/Retrofitinstance.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.network Menentukan bahwa file ini berada dalam *package* com.example.myanimelistapp.network, tempat semua kode terkait jaringan dikelompokkan.
2. **Baris 7–10:** import ... Mengimpor kelas dan fungsi yang diperlukan:
   - com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterFactory: Fungsi ekstensi dari *library* tambahan yang dibuat oleh Jake Wharton. Fungsi ini berfungsi sebagai jembatan yang memungkinkan Retrofit menggunakan kotlinx.serialization untuk mengubah data JSON menjadi objek Kotlin.
   - kotlinx.serialization.json.Json: Kelas utama dari *library* kotlinx.serialization untuk mengkonfigurasi parser JSON.
   - okhttp3.MediaType.Companion.toMediaType: Fungsi untuk membuat objek MediaType yang mendeskripsikan tipe konten data (dalam hal ini, "application/json").
   - retrofit2.Retrofit: Kelas utama dari *library* Retrofit untuk membangun klien HTTP.

Bagian: Definisi Singleton Object

3. **Baris 12:** object RetrofitInstance { ... } Mendefinisikan RetrofitInstance sebagai sebuah **object** (singleton). Dalam Kotlin, object adalah cara mudah untuk membuat sebuah kelas yang hanya akan memiliki **satu instance** di seluruh aplikasi. Ini adalah pola desain yang sangat efisien untuk objek-objek seperti klien jaringan, karena Anda tidak perlu membuatnya berulang kali.

Bagian: Konfigurasi

4. **Baris 13:** private const val BASE_URL = "https://api.themoviedb.org/3/" Mendeklarasikan URL dasar (base URL) dari API.
5. private: Hanya dapat diakses dari dalam object RetrofitInstance.
6. const: Nilainya ditetapkan pada saat kompilasi (compile-time), membuatnya sedikit lebih efisien.
7. Semua *endpoint* yang didefinisikan di MovieApiService (seperti "movie/popular") akan digabungkan dengan URL ini.
8. **Baris 15–17:** private val json = Json { ... } Membuat dan mengkonfigurasi *instance* dari parser JSON.
9. ignoreUnknownKeys = true: Ini adalah konfigurasi yang sangat penting. Jika API mengirimkan *field* JSON yang tidak didefinisikan di dalam kelas data Kotlin (Movie atau MovieResponse), aplikasi **tidak akan crash**. Sebaliknya, *field* tersebut akan diabaikan. Ini membuat aplikasi lebih tangguh terhadap perubahan di sisi API.
10. **Baris 19:** val api: MovieApiService by lazy { ... } Ini adalah bagian inti dari kode ini, di mana *instance* layanan API dibuat.
11. val api: MovieApiService: Mendeklarasikan sebuah properti publik bernama api yang akan dapat diakses dari bagian lain aplikasi untuk melakukan panggilan jaringan.
12. by lazy: Ini adalah delegasi properti Kotlin yang sangat berguna. Kode di dalam kurung kurawal {...} hanya akan dieksekusi **satu kali**, yaitu saat properti api diakses

untuk pertama kalinya. Untuk semua akses berikutnya, nilai yang sudah ada akan dikembalikan. Ini memastikan proses pembuatan Retrofit yang mahal hanya terjadi saat benar-benar dibutuhkan dan hanya sekali saja.

13. **Baris 20:** Retrofit.Builder() Memulai proses pembuatan objek Retrofit menggunakan pola *builder*.
14. **Baris 21:** .baseUrl(BASE_URL) Menetapkan URL dasar yang telah didefinisikan sebelumnya.
15. **Baris 22:** .addConverterFactory(...) Memberitahu Retrofit bagaimana cara mengonversi data. Dalam hal ini, ia menggunakan kotlinx.serialization untuk mengubah respons JSON dari server menjadi objek data Kotlin (MovieResponse).
16. **Baris 23:** .build() Menyelesaikan konfigurasi dan membuat objek Retrofit yang siap digunakan.
17. **Baris 24:** .create(MovieApiService::class.java) Langkah terakhir. Metode ini mengambil antarmuka MovieApiService dan secara dinamis membuat implementasi konkret dari antarmuka tersebut, lengkap dengan semua kode jaringan yang diperlukan untuk berkomunikasi dengan API sesuai definisi.

Repository/MovieRepository.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.repository Menempatkan kelas ini dalam *package* repository, sesuai dengan perannya dalam arsitektur aplikasi.
2. **Baris 7–12:** import ... Mengimpor semua komponen yang diperlukan:
   - MovieDao dan MovieEntity: Untuk berinteraksi dengan database lokal (Room).
   - Movie: Model data yang digunakan oleh UI dan lapisan domain.
   - MovieApiService: Antarmuka untuk melakukan panggilan jaringan (Retrofit).
   - kotlinx.coroutines.flow.*: Komponen dari Kotlin Coroutines untuk bekerja dengan aliran data asinkron (Flow).

Bagian: Definisi dan Dependensi Kelas

3. **Baris 14:** class MovieRepository(Mendefinisikan kelas MovieRepository.
4. **Baris 15–17:** private val apiService: ..., private val movieDao: ..., private val apiKey: String Mendefinisikan dependensi yang dibutuhkan oleh *repository*. Dependensi ini dimasukkan melalui *constructor* (prinsip *Dependency Injection*).
   - **apiService**: Untuk mengambil data dari internet.
   - **movieDao**: Untuk mengakses, menyimpan, dan menghapus data di database lokal.
   - **apiKey**: Kunci API yang diperlukan untuk otentikasi saat melakukan panggilan jaringan.

Bagian: Fungsi Utama - getPopularMovies()

5. **Baris 19:** fun getPopularMovies(): Flow<Result<List<Movie>>> = flow { ... } Mendefinisikan fungsi utama *repository* ini.

- Fungsi ini mengembalikan **Flow**, yang merupakan aliran data yang dapat mengeluarkan (*emit*) beberapa nilai dari waktu ke waktu. Ini sangat cocok untuk menampilkan data yang bisa berubah (misalnya, loading, lalu data dari cache, lalu error).
- Tipe data yang di-emit adalah Result<List<Movie>>, sebuah kelas pembungkus (*wrapper*) untuk merepresentasikan berbagai status: Loading, Success (dengan data), atau Error.
- Blok flow { ... } adalah *builder* dari Coroutines untuk membangun Flow ini.

6. **Baris 20:** emit(Result.Loading) Langkah pertama adalah mengeluarkan status **Loading**. UI yang mengamati *flow* ini dapat menampilkan indikator pemuatan.
7. **Baris 23–24:** val cachedMovies = ...; emit(Result.Success(...)) Ini adalah strategi **cache-first**.
   - Aplikasi segera mengambil data yang sudah ada di database (movieDao.getAllMovies().first()).
   - Data dari cache ini (yang merupakan MovieEntity) dipetakan menjadi Movie dan langsung di-emit sebagai Result.Success. Hasilnya, pengguna akan melihat data di layar dengan sangat cepat, bahkan sebelum panggilan jaringan selesai (atau jika tidak ada koneksi internet).
8. **Baris 26:** try { ... } Blok try digunakan untuk membungkus panggilan jaringan, yang berpotensi gagal.
9. **Baris 28–29:** val response = ...; val moviesFromApi = ... Melakukan panggilan jaringan sesungguhnya menggunakan apiService untuk mendapatkan data film terbaru dari API.
10. **Baris 32–33:** movieDao.deleteAllMovies(); movieDao.insertMovies(...) Jika panggilan jaringan berhasil, *repository* akan memperbarui cache lokal. Data lama dihapus, dan data baru dari API (setelah dipetakan ke MovieEntity) disimpan ke dalam database. Ini memastikan data lokal selalu segar.
11. **Baris 35–37:** catch (e: Exception) { emit(Result.Error(...)) } Jika terjadi kesalahan di dalam blok try (misalnya, tidak ada koneksi internet), blok catch akan dieksekusi.
    - Sebuah status Result.Error akan di-emit dengan pesan kesalahan.
    - Poin pentingnya adalah, meskipun jaringan gagal, UI sudah menampilkan data dari cache yang di-emit pada langkah sebelumnya, sehingga aplikasi tidak terlihat kosong atau rusak.

Bagian: Fungsi Pemetaan (Mapping Functions)

12. **Baris 42–43:** fun MovieEntity.toMovie() ... dan fun Movie.toMovieEntity() ... Ini adalah **fungsi ekstensi (extension functions)** yang sangat berguna untuk mengubah objek dari satu tipe ke tipe lain.
13. **toMovie()**: Mengubah objek database (MovieEntity) menjadi objek domain/UI (Movie).
14. **toMovieEntity()**: Mengubah objek domain/UI (Movie) menjadi objek database (MovieEntity).

- Penggunaan fungsi ini adalah praktik yang baik untuk memisahkan model data yang spesifik untuk database dari model data yang digunakan oleh seluruh aplikasi.

network/Result.kt

Bagian: Definisi Sealed Class

1. **Baris 5:** sealed class Result<out T> Mendefinisikan sebuah **sealed class** bernama Result.
   - **sealed class**: Ini adalah jenis kelas khusus di Kotlin. Keistimewaannya adalah semua sub-kelasnya (dalam hal ini Loading, Success, dan Error) harus didefinisikan di dalam file yang sama. Hal ini memungkinkan *compiler* untuk mengetahui semua kemungkinan status yang ada. Hasilnya, saat Anda menggunakan when untuk memeriksa sebuah objek Result, Anda tidak perlu menambahkan cabang else, karena semua kemungkinan sudah tercakup.
   - **<out T>**: Ini adalah parameter **generic** yang bersifat **covariant**. Sederhananya, ini berarti Result dapat membungkus tipe data apa pun (T). Misalnya, Anda bisa memiliki Result<List<Movie>> untuk daftar film atau Result<User> untuk data pengguna.

Bagian: Sub-Kelas (Representasi Status)

Result memiliki tiga sub-kelas yang masing-masing merepresentasikan status yang berbeda dari sebuah operasi.

2. **Baris 6:** object Loading : Result<Nothing>() Merepresentasikan status bahwa operasi **sedang berlangsung**.
   - **object**: Didefinisikan sebagai object karena status Loading tidak perlu membawa data apa pun; ini hanyalah sebuah sinyal. Menggunakan object membuatnya menjadi *singleton* (hanya ada satu *instance* Loading).
   - **Result<Nothing>**: Nothing adalah tipe khusus di Kotlin yang tidak memiliki nilai. Ini digunakan karena Loading tidak membawa data hasil.
3. **Baris 7:** data class Success<T>(val data: T) : Result<T>() Merepresentasikan status bahwa operasi telah **berhasil**.
   - **data class**: Digunakan karena kelas ini membawa data.
   - **(val data: T)**: Properti ini menyimpan data hasil operasi yang berhasil, dengan tipe generic T.
4. **Baris 8:** data class Error(val message: String) : Result<Nothing>() Merepresentasikan status bahwa operasi telah **gagal**.
   - **data class**: Digunakan untuk membawa informasi tentang kesalahan.
   - **(val message: String)**: Properti ini menyimpan pesan kesalahan dalam bentuk String untuk menjelaskan apa yang salah.

viewmodel/MovieViewModel.kt

Bagian: Definisi dan Dependensi ViewModel

1. **Baris 15:** class MovieViewModel(private val repository: MovieRepository) : ViewModel() Mendefinisikan kelas MovieViewModel.
   - Ia menerima **MovieRepository** sebagai dependensi melalui *constructor*. Ini adalah praktik *Dependency Injection* yang baik, membuat ViewModel lebih mudah diuji.
   - Ia mewarisi (meng-extend) **ViewModel** dari Android Jetpack. Ini membuatnya sadar akan siklus hidup (lifecycle-aware), artinya ia tidak akan hancur saat terjadi perubahan konfigurasi seperti rotasi layar.

Bagian: Manajemen State (State Management)

2. **Baris 17:** private val _uiState = MutableStateFlow<MovieUiState>(MovieUiState.Loading) Ini adalah properti internal untuk menampung *state* UI saat ini.
   - **MutableStateFlow** adalah jenis Flow khusus yang menyimpan satu nilai *state* yang dapat diperbarui.
   - Nilai awalnya diatur ke MovieUiState.Loading, jadi saat UI pertama kali muncul, ia akan langsung berada dalam status memuat.
   - Properti ini bersifat **private** agar hanya bisa diubah dari dalam ViewModel ini.
3. **Baris 18:** val uiState: StateFlow<MovieUiState> = _uiState Ini adalah properti publik yang akan diobservasi oleh UI.
   - Tipe datanya adalah **StateFlow**, yang merupakan versi *read-only* (hanya bisa dibaca) dari MutableStateFlow.
   - Pola ini (properti private yang bisa diubah dan public yang hanya bisa dibaca) disebut **backing property** dan merupakan praktik terbaik untuk memastikan *state* tidak diubah secara tidak sengaja dari luar ViewModel.

Bagian: Inisialisasi dan Pengambilan Data

4. **Baris 20-22:** init { fetchPopularMovies() } Blok **init** dieksekusi secara otomatis saat ViewModel pertama kali dibuat. Blok ini langsung memanggil fetchPopularMovies() untuk memulai proses pengambilan data.
5. **Baris 24:** private fun fetchPopularMovies() { ... } Fungsi privat yang berisi logika untuk mengambil data film.
6. **Baris 25:** viewModelScope.launch { ... } Memulai sebuah *coroutine* baru di dalam **viewModelScope**. Ini adalah cara yang aman untuk menjalankan operasi asinkron. *Coroutine* yang diluncurkan di sini akan secara otomatis dibatalkan jika ViewModel dihancurkan, sehingga mencegah kebocoran memori (*memory leak*).
7. **Baris 26-34:** repository.getPopularMovies().catch { ... }.collect { ... } Ini adalah inti dari pengambilan data menggunakan Flow.
   - .getPopularMovies(): Memanggil fungsi dari *repository* untuk mendapatkan Flow yang berisi hasil (Result).

- .catch { ... }: Menangani jika ada kesalahan tak terduga yang terjadi di dalam *flow* dari *repository*. Ini adalah jaring pengaman.
- .collect { result -> ... }: Mulai "mendengarkan" atau mengoleksi setiap nilai (result) yang di-*emit* oleh *flow* dari *repository*.

8. **when (result) { ... }**: Memeriksa tipe dari result yang diterima (Loading, Success, atau Error) dan memperbarui _uiState dengan MovieUiState yang sesuai. Ini mengubah *state* dari lapisan data menjadi *state* yang siap digunakan oleh lapisan UI.

Bagian: State UI (MovieUiState)

9. **Baris 38:** sealed interface MovieUiState { ... } Mendefinisikan **sealed interface** yang mewakili semua kemungkinan *state* yang bisa dimiliki oleh UI. Menggunakan sealed interface atau sealed class adalah praktik terbaik untuk merepresentasikan *state* secara terbatas dan aman.
10. **Baris 39-41:** Loading, Success, Error Tiga kemungkinan *state* didefinisikan di sini:
    - **Loading**: Menandakan data sedang dimuat.
    - **Success**: Menandakan data berhasil diambil dan menyimpannya dalam properti movies.
    - **Error**: Menandakan terjadi kesalahan dan menyimpan pesan kesalahannya.

viewmodel/MovieViewFactory.kt

Bagian: Package dan Imports

1. **Baris 3:** package com.example.myanimelistapp.viewmodel Menempatkan kelas ini dalam *package* viewmodel, karena perannya adalah untuk membuat objek-objek ViewModel.
2. **Baris 7–9:** import ... Mengimpor kelas-kelas yang diperlukan:
    - androidx.lifecycle.ViewModel: Kelas dasar untuk semua ViewModel.
    - androidx.lifecycle.ViewModelProvider: Menyediakan kelas Factory yang dapat kita implementasikan untuk membuat ViewModel.
    - com.example.myanimelistapp.repository.MovieRepository: Dependensi yang dibutuhkan oleh MovieViewModel, yang akan disediakan oleh *factory* ini.

Bagian: Definisi Factory Class

3. **Baris 11:** class ViewModelFactory(private val repository: MovieRepository) : ViewModelProvider.Factory { ... } Mendefinisikan kelas ViewModelFactory.
4. **(private val repository: MovieRepository)**: *Constructor* kelas ini menerima sebuah MovieRepository. Ini adalah inti dari **Dependency Injection**. *Factory* ini "memegang" *repository* agar bisa "menyuntikkannya" ke ViewModel saat dibuat.
5. **: ViewModelProvider.Factory**: Kelas ini mengimplementasikan antarmuka ViewModelProvider.Factory dari Android Jetpack. Ini adalah sebuah "kontrak" yang mengharuskan kelas ini untuk menyediakan metode create, yang akan digunakan oleh sistem untuk membuat *instance* ViewModel.

Bagian: Metode create

6. **Baris 12:** override fun <T : ViewModel> create(modelClass: Class<T>): T { ... } Ini adalah satu-satunya metode yang wajib diimplementasikan dari antarmuka ViewModelProvider.Factory.
   - **Tugasnya**: Untuk membuat dan mengembalikan *instance* dari kelas ViewModel yang diminta.
   - **modelClass: Class<T>**: Parameter ini adalah kelas ViewModel yang ingin dibuat oleh sistem (misalnya, MovieViewModel.class).
7. **Baris 13:** if (modelClass.isAssignableFrom(MovieViewModel::class.java)) { ... } Ini adalah pemeriksaan kondisi yang sangat penting. Artinya adalah: **"Jika kelas yang diminta (modelClass) adalah MovieViewModel atau turunan dari MovieViewModel..."**
   Ini memastikan bahwa *factory* ini hanya membuat ViewModel yang ia kenali. Jika ada ViewModel lain di aplikasi (misalnya, UserViewModel), *factory* ini tidak akan mencoba membuatnya.
8. **Baris 14:** @Suppress("UNCHECKED_CAST") Anotasi ini digunakan untuk menekan peringatan dari *compiler*. *Compiler* tidak bisa 100% yakin bahwa MovieViewModel(repository) dapat di-*cast* menjadi tipe generik T. Namun, karena kita sudah memeriksanya di baris 13, kita tahu bahwa operasi ini aman.
9. **Baris 15:** return MovieViewModel(repository) as T Jika kondisi if terpenuhi, inilah yang terjadi:
10. **MovieViewModel(repository)**: Sebuah *instance* baru dari MovieViewModel dibuat. Yang terpenting, repository yang dipegang oleh *factory* ini diteruskan ke dalam *constructor* ViewModel.
11. **as T**: Hasilnya di-*cast* ke tipe T (tipe ViewModel yang diminta) dan dikembalikan.
12. **Baris 17:** throw IllegalArgumentException("Unknown ViewModel class") Jika kondisi if di baris 13 tidak terpenuhi (artinya sistem meminta ViewModel yang tidak dikenali oleh *factory* ini), maka sebuah IllegalArgumentException akan dilemparkan. Ini adalah cara yang benar untuk menandakan bahwa *factory* ini tidak dapat memenuhi permintaan tersebut.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/ALYAROSAAN/Pemrograman-Mobile-Modul5