

**LAPORAN PRAKTIKUM  
PEMROGRAMAN MOBILE  
MODUL 5**



**CONNECT TO THE INTERNET**

**Oleh:**

**Muhammad Firas                  NIM. 2210817110014**

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

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

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

Nama Praktikan : Muhammad Firas  
NIM : 2210817110014

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**

LEMBAR PENGESAHAN.....	2
DAFTAR ISI .....	3
DAFTAR GAMBAR.....	4
DAFTAR TABEL .....	5
SOAL.....	6
A.    Source Code .....	6
B.    Output Program.....	19
C.    Pembahasan .....	20
D.    Tautan Git .....	28

## **DAFTAR GAMBAR**

Gambar 1. Screenshot HomeScreen.kt dengan data dari API .....	19
Gambar 2. Screenshot Halaman Detail dengan data dari API.....	20

## **DAFTAR TABEL**

Tabel 1. Source Code Jawaban Soal 1 MainActivity Jetpack Compose .....	6
Tabel 2. Source Code Jawaban Soal 1 HomeScreen.kt Jetpack Compose .....	7
Tabel 3. Source Code Jawaban Soal 1 DetailScreen.kt Jetpack Compose .....	10
Tabel 4. Source Code Jawaban Soal 1 ListItem.kt Jetpack Compose .....	12
Tabel 5. Source Code Jawaban Soal 1 MovieViewModel.kt .....	13
Tabel 6. Source Code Jawaban Soal 1 MovieViewModelFactory.kt.....	14
Tabel 7. Source Code Jawaban Soal 1 MovieApp.kt .....	14
Tabel 8. Source Code Jawaban Soal 1 Movie ApiService.kt .....	15
Tabel 9. Source Code Jawaban Soal 1 RetrofitInstance.kt.....	15
Tabel 10. Source Code Jawaban Soal 1 Movie Dao.kt .....	16
Tabel 11. Source Code Jawaban Soal 1 Movie Database.kt .....	17
Tabel 12. Source Code Jawaban Soal 1 Movie Entity.kt .....	17
Tabel 13. Source Code Jawaban Soal 1 Movie Response.kt.....	18
Tabel 14. Source Code Jawaban Soal 1 Movie Repository.kt Jetpack Compose .....	18

# SOAL

## 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.

## A. Source Code

### JETPACK COMPOSE

#### MainActivity.kt

Tabel 1. Source Code Jawaban Soal 1 MainActivity Jetpack Compose

```
1 package com.example.movieList
2
3 import android.os.Bundle
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import androidx.compose.material3.MaterialTheme
7 import androidx.compose.material3.Surface
8 import androidx.compose.runtime.Composable
9 import androidx.compose.ui.tooling.preview.Preview
10 import androidx.navigation.compose.rememberNavController
11 import com.example.movieList.navigation.AppNavGraph
12 import com.example.movieList.ui.theme.MovieListTheme
13
14 class MainActivity : ComponentActivity() {
15     override fun onCreate(savedInstanceState: Bundle?) {
16         super.onCreate(savedInstanceState)
17         setContent {
```

```

18         MovieListTheme {
19             Surface(color =
20                 MaterialTheme.colorScheme.background) {
21                 val navController = rememberNavController()
22                 AppNavGraph(navController = navController)
23             }
24         }
25     }
26 }
27
28 @Composable
29 @Preview(showBackground = true, showSystemUi = true)
30 fun MainPreview() {
31     MovieListTheme {
32         val navController = rememberNavController()
33         AppNavGraph(navController = navController)
34     }
35 }
```

## HomeScreen.kt

Tabel 2. Source Code Jawaban Soal 1 HomeScreen.kt Jetpack Compose

```

1 package com.example.movieList.ui.screen
2
3 import android.content.Intent
4 import android.net.Uri
5 import androidx.compose.foundation.background
6 import androidx.compose.foundation.layout.Arrangement
7 import androidx.compose.foundation.layout.Column
8 import androidx.compose.foundation.layout.PaddingValues
9 import androidx.compose.foundation.layout.Row
10 import androidx.compose.foundation.layout.Spacer
11 import androidx.compose.foundation.layout.fillMaxWidth
12 import androidx.compose.foundation.layout.height
13 import androidx.compose.foundation.layout.padding
14 import androidx.compose.foundation.layout.size
15 import androidx.compose.foundation.layout.width
16 import androidx.compose.foundation.lazy.LazyColumn
17 import androidx.compose.foundation.lazy.itemsIndexed
18 import androidx.compose.foundation.shape.RoundedCornerShape
19 import androidx.compose.material3.Button
20 import androidx.compose.material3.ButtonDefaults
21 import androidx.compose.material3.Card
22 import androidx.compose.material3.CardDefaults
23 import androidx.compose.material3.Text
24 import androidx.compose.runtime.Composable
25 import androidx.compose.runtime.collectAsState
26 import androidx.compose.ui.Alignment
27 import androidx.compose.ui.Modifier
```

```
28 import androidx.compose.ui.draw.clip
29 import androidx.compose.ui.graphics.Color
30 import androidx.compose.ui.layout.ContentScale
31 import androidx.compose.ui.platform.LocalContext
32 import androidx.compose.ui.text.font.FontWeight
33 import androidx.compose.ui.unit.dp
34 import androidx.compose.ui.unit.sp
35 import androidx.lifecycle.viewmodel.compose.viewModel
36 import androidx.navigation.NavController
37 import coil.compose.AsyncImage
38 import com.example.movieList.viewmodel.MovieViewModel
39 import timber.log.Timber
40
41 @Composable
42 fun HomeScreen(
43     navController: NavController,
44     viewModel: MovieViewModel) {
45     val movieListState = viewModel.movieList.collectAsState()
46     val movieList = movieListState.value
47     val context = LocalContext.current
48
49     LazyColumn(
50         contentPadding = PaddingValues(20.dp),
51         verticalArrangement = Arrangement.spacedBy(20.dp),
52         modifier = Modifier.background(Color.Black)
53     ) {
54         itemsIndexed(movieList) { _, item ->
55             Card(
56                 shape = RoundedCornerShape(16.dp),
57                 colors =
58                     CardDefaults.cardColors(containerColor = Color(0xFF141F1F)),
59                 modifier = Modifier.fillMaxWidth()
60             ) {
61                 Row(
62                     modifier = Modifier
63                         .padding(12.dp)
64                         .fillMaxWidth(),
65                     verticalAlignment =
66                     Alignment.CenterVertically
67                 ) {
68                     AsyncImage(
69                         model =
70                             "https://image.tmdb.org/t/p/w500${item.poster_path}",
71                         contentDescription = item.title,
72                         contentScale = ContentScale.Crop,
73                         modifier = Modifier
74                             .size(width = 90.dp, height =
75                                 130.dp)
76                             .clip(RoundedCornerShape(12.dp))
77                 )
78
79                 Spacer(modifier = Modifier.width(12.dp))
80             }
81         }
82     }
83 }
```

```
73
74             Column(modifier = Modifier.weight(1f)) {
75                 Text(
76                     text = item.title,
77                     fontSize = 18.sp,
78                     fontWeight = FontWeight.Bold,
79                     color = Color.White
80                 )
81
82             Spacer(modifier =
83                 Modifier.height(6.dp))
84
85             Text(
86                 text = item.overview,
87                 fontSize = 13.sp,
88                 color = Color(0xFFE0E0E0),
89                 maxLines = 4
90             )
91
92             Spacer(modifier =
93                 Modifier.height(8.dp))
94
95             Row(
96                 horizontalArrangement =
97                 Arrangement.SpaceBetween,
98                 modifier = Modifier.fillMaxWidth()
99             ) {
100                 Button(
101                     onClick = {
102                         val imdbUrl =
103                             "https://www.imdb.com/find?q=${item.title}"
104                         Timber.i("Opening IMDB:
105                         $imdbUrl")
106                         val intent =
107                             Intent(Intent.ACTION_VIEW, Uri.parse(imdbUrl))
108                         context.startActivity(intent)
109                         },
110                         shape =
111                         RoundedCornerShape(50),
112                         colors =
113                         ButtonDefaults.buttonColors(containerColor =
114                             Color(0xFFB3E5FC))
115                         )
116                         Text("IMDB", color =
117                             Color.Black)
118
119             Button(
120                 onClick = {
121                     Timber.i("Opening detail
122                     for ID: ${item.id}")
123             }
124         )
125     }
126 }
```

```

118     navController.navigate("detail/${item.id}")
119             },
120             shape =
121             RoundedCornerShape(50),
122             colors =
123             ButtonDefaults.buttonColors(containerColor =
124             Color(0xFF9FA8DA))
125         ) {
126             Text("Detail", color =
127             Color.Black)
128         }
129     }
130 }
131 }
```

## DetailScreen.kt

Tabel 3. Source Code Jawaban Soal 1 DetailScreen.kt Jetpack Compose

```

1 package com.example.movieList.ui.screen
2
3 import androidx.compose.foundation.background
4 import androidx.compose.foundation.layout.*
5 import androidx.compose.foundation.shape.RoundedCornerShape
6 import androidx.compose.material3.*
7 import androidx.compose.runtime.Composable
8 import androidx.compose.ui.Alignment
9 import androidx.compose.ui.Modifier
10 import androidx.compose.ui.draw.clip
11 import androidx.compose.ui.graphics.Color
12 import androidx.compose.ui.layout.ContentScale
13 import androidx.compose.ui.text.font.FontWeight
14 import androidx.compose.ui.text.style.TextAlign
15 import androidx.compose.ui.unit.dp
16 import androidx.compose.ui.unit.sp
17 import androidx.navigation.NavController
18 import coil.compose.AsyncImage
19 import com.example.movieList.viewmodel.MovieViewModel
20 import timber.log.Timber
21
22 @Composable
23 fun DetailScreen(
24     itemId: Int,
25     navController: NavController,
26     viewModel: MovieViewModel
27 ) {
```

```
28     val item = viewModel.getMovieById(itemId)
29
30     Timber.d("Loaded detail for item ID: $itemId - "
31     "${item?.title}")
32
33     if (item != null) {
34         Column(
35             modifier = Modifier
36                 .fillMaxSize()
37                 .background(Color.Black)
38                 .padding(16.dp),
39             verticalArrangement = Arrangement.Top,
40             horizontalAlignment = Alignment.CenterHorizontally
41         ) {
42             AsyncImage(
43                 model =
44                 "https://image.tmdb.org/t/p/w500${item.poster_path}",
45                 contentDescription = item.title,
46                 modifier = Modifier
47                     .height(300.dp)
48                     .fillMaxWidth()
49                     .clip(RoundedCornerShape(12.dp)),
50                 contentScale = ContentScale.Crop
51             )
52
53             Spacer(modifier = Modifier.height(16.dp))
54
55             Text(
56                 text = item.title,
57                 fontSize = 24.sp,
58                 fontWeight = FontWeight.Bold,
59                 color = Color.White
60             )
61
62             Text(
63                 text = "★ ${item.vote_average}",
64                 fontSize = 18.sp,
65                 color = Color.White
66             )
67
68             Spacer(modifier = Modifier.height(12.dp))
69
70             Text(
71                 text = item.overview ?: "No description
72 available.",
73                 fontSize = 16.sp,
74                 color = Color.White,
75                 textAlign = TextAlign.Justify
76             )
77
78             Spacer(modifier = Modifier.height(24.dp))
79
```

```

80
81             Button(
82                 onClick = { navController.popBackStack() },
83                 shape = RoundedCornerShape(12.dp),
84                 colors =
85             ButtonDefaults.buttonColors(containerColor =
86                 Color(0xFF9C27B0))
87             ) {
88                 Text("Back")
89             }
90         } else {
91             Text("Item not found", color = Color.Red)
92         }
93     }

```

## NavGraph.kt

Tabel 4. Source Code Jawaban Soal 1 ListItem.kt Jetpack Compose

```

1 package com.example.movieList.presentation.navigation

3 import android.app.Application
4 import androidx.compose.runtime.Composable
5 import androidx.compose.ui.platform.LocalContext
6 import androidx.lifecycle.viewmodel.compose.viewModel
7 import androidx.navigation.NavHostController
8 import androidx.navigation.compose.NavHost
9 import androidx.navigation.compose.composable
10 import androidx.navigation.navArgument
11 import androidx.navigation.NavType
12 import com.example.movieList.ui.screen.DetailScreen
13 import com.example.movieList.ui.screen.HomeScreen
14 import com.example.movieList.viewmodel.MovieViewModel
15 import com.example.movieList.viewmodel.MovieViewModelFactory
16
17 object Routes {
18     const val HOME = "home"
19     const val DETAIL = "detail/{itemId}"
20 }
21
22 @Composable
23 fun AppNavGraph(navController: NavHostController) {
24     val context = LocalContext.current.applicationContext as Application
25     val viewModel: MovieViewModel = viewModel(factory =
26         MovieViewModelFactory(context))

27     NavHost(
28         navController = navController,
29         startDestination = Routes.HOME

```

```

30    ) {
31        composable(Routes.HOME) {
32            HomeScreen(navController, viewModel) // Pass shared
33            ViewModel
34        }
35
36        composable(
37            route = Routes.DETAIL,
38            arguments = listOf(navArgument("itemId") { type =
39                NavType.IntType })
40            ) { backStackEntry ->
41                val itemId =
42                    backStackEntry.arguments?.getInt("itemId") ?: -1
43                    DetailScreen(itemId = itemId, navController =
44                    navController, viewModel = viewModel)
45            }
46        }
47    }

```

## MovieViewModel.kt

Tabel 5. Source Code Jawaban Soal 1 MovieViewModel.kt

```

1 package com.example.movieList.viewmodel
2
3 import android.app.Application
4 import androidx.lifecycle.AndroidViewModel
5 import androidx.lifecycle.viewModelScope
6 import com.example.movieList.data.db.MovieDatabase
7 import com.example.movieList.data.db.MovieEntity
8 import com.example.movieList.domain.repository.MovieRepository
9 import kotlinx.coroutines.flow.MutableStateFlow
10 import kotlinx.coroutines.flow.StateFlow
11 import kotlinx.coroutines.launch
12 import timber.log.Timber
13
14 class MovieViewModel(application: Application) :
15     AndroidViewModel(application) {
16
17     private val _movieList =
18         MutableStateFlow<List<MovieEntity>>(emptyList())
19     val movieList: StateFlow<List<MovieEntity>> get() =
20         _movieList
21
22     private val repository = MovieRepository(
23         dao =
24             MovieDatabase.getDatabase(application).movieDao(),
25         context = application
26     )
27
28     private val apiKey = "5599d93394ce2696cbe43b45ce6d173d"

```

```

25     init {
26         fetchMovies()
27     }
28
29     private fun fetchMovies() {
30         viewModelScope.launch {
31             val data = repository.getMovies(apiKey)
32             _movieList.value = data
33         }
34     }
35
36     fun getMovieById(id: Int): MovieEntity? {
37         return _movieList.value.find { it.id == id }
38     }
39 }
```

## MovieViewModelFactory.kt

Tabel 6. Source Code Jawaban Soal 1 MovieViewModelFactory.kt

```

1 package com.example.movieList.viewmodel
2
3 import android.app.Application
4 import androidx.lifecycle.ViewModel
5 import androidx.lifecycle.ViewModelProvider
6
7 class MovieViewModelFactory(
8     private val application: Application
9 ) : ViewModelProvider.Factory {
10     override fun <T : ViewModel> create(modelClass: Class<T>):
11         T {
12         if
13             (modelClass.isAssignableFrom(MovieViewModel::class.java)) {
14                 return MovieViewModel(application) as T
15             }
16             throw IllegalArgumentException("Unknown ViewModel
17             class")
18     }
19 }
```

## MovieApp.kt

Tabel 7. Source Code Jawaban Soal 1 MovieApp.kt

```

1 package com.example.movieListxml
2
3 import android.app.Application
4 import timber.log.Timber
5
6 class MovieApp : Application() {
```

```

7     override fun onCreate() {
8         super.onCreate()
9
10        if (BuildConfig.DEBUG) {
11            Timber.plant(Timber.DebugTree())
12            Timber.i("Timber initialized in MovieApp")
13        }
14    }
15 }
```

## Movie ApiService.kt

Tabel 8. Source Code Jawaban Soal 1 Movie ApiService.kt

```

1 package com.example.movie.list.data.api
2
3 import com.example.movie.list.domain.model.MovieResponse
4 import retrofit2.http.GET
5 import retrofit2.http.Query
6
7 interface Movie ApiService {
8     @GET("movie/popular")
9     suspend fun getPopularMovies(
10         @Query("api_key") apiKey: String
11     ): MovieResponse
12 }
```

## Retrofit Instance.kt

Tabel 9. Source Code Jawaban Soal 1 Retrofit Instance.kt

```

1 package com.example.movie.list.data.api
2
3 import
4 com.jakewharton.retrofit2.converter.kotlinx.serialization.asCon
5 verterFactory
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 retrofit2.create
12
13 object RetrofitInstance {
14
15     private val loggingInterceptor =
16         HttpLoggingInterceptor().apply {
17             level = HttpLoggingInterceptor.Level.BODY
18         }
19
20     private val client = OkHttpClient.Builder()
```

```

18     .addInterceptor(loggingInterceptor)
19     .build()
20
21     private val json = Json {
22         ignoreUnknownKeys = true // ignores unused fields
23     }
24
25     private val retrofit by lazy {
26         Retrofit.Builder()
27             .baseUrl("https://api.themoviedb.org/3/")
28             .client(client)
29
30         .addConverterFactory(json.asConverterFactory("application/json"
31             .toMediaType()))
32             .build()
33     }
34
35     val api: MovieApiService by lazy {
36         retrofit.create(MovieApiService::class.java)
37     }
38 }
```

## MovieDao.kt

Tabel 10. Source Code Jawaban Soal 1 MovieDao.kt

```

1 package com.example.movieList.data.db
2
3 import androidx.room.Dao
4 import androidx.room.Insert
5 import androidx.room.OnConflictStrategy
6 import androidx.room.Query
7
8 @Dao
9 interface MovieDao {
10     @Query("SELECT * FROM movies")
11     suspend fun getAllMovies(): List<MovieEntity>
12
13     @Insert(onConflict = OnConflictStrategy.REPLACE)
14     suspend fun insertAll(movies: List<MovieEntity>)
15
16     @Query("DELETE FROM movies")
17     suspend fun clearAll()
18 }
```

## MovieDatabase.kt

Tabel 11. Source Code Jawaban Soal 1 MovieDatabase.kt

```
1 package com.example.movieList.data.db
2
3 import android.content.Context
4 import androidx.room.Database
5 import androidx.room.Room
6 import androidx.room.RoomDatabase
7
8 @Database(entities = [MovieEntity::class], version = 1,
9 exportSchema = false)
10 abstract class MovieDatabase : RoomDatabase() {
11     abstract fun movieDao(): MovieDao
12
13     companion object {
14         @Volatile private var INSTANCE: MovieDatabase? = null
15
16         fun getDatabase(context: Context): MovieDatabase {
17             return INSTANCE ?: synchronized(this) {
18                 val instance = Room.databaseBuilder(
19                     context.applicationContext,
20                     MovieDatabase::class.java,
21                     "movie_database"
22                 ).build()
23                 INSTANCE = instance
24                 instance
25             }
26         }
27     }
}
```

## MovieEntity.kt

Tabel 12. Source Code Jawaban Soal 1 MovieEntity.kt

```
1 package com.example.movieList.data.db
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 overview: String,
11    val poster_path: String,
12    val vote_average: Double
13 )
```

## MovieResponse.kt

Tabel 13. Source Code Jawaban Soal 1 MovieResponse.kt

```
1 package com.example.movieList.domain.model
2
3 import kotlinx.serialization.SerialName
4 import kotlinx.serialization.Serializable
5
6 @Serializable
7 data class MovieResponse(
8     val results: List<Movie>
9 )
10
11 @Serializable
12 data class Movie(
13     val id: Int,
14     val title: String,
15     val overview: String,
16     @SerialName("poster_path") val posterPath: String,
17     @SerialName("vote_average") val voteAverage: Double
18 )
```

## MovieRepository.kt

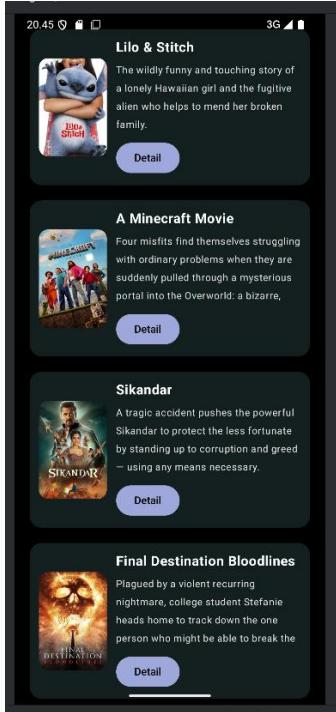
Tabel 14. Source Code Jawaban Soal 1 MovieRepository.kt Jetpack Compose

```
1 package com.example.movieList.domain.repository
2
3 import android.content.Context
4 import com.example.movieList.data.api.RetrofitInstance
5 import com.example.movieList.data.db.MovieDao
6 import com.example.movieList.data.db.MovieDatabase
7 import com.example.movieList.data.db.MovieEntity
8 import kotlinx.coroutines.Dispatchers
9 import kotlinx.coroutines.withContext
10 import timber.log.Timber
11
12 class MovieRepository(private val dao: MovieDao, private val
13 context: Context) {
14
15     suspend fun getMovies(apiKey: String): List<MovieEntity> {
16         // Step 1: Load from local cache first
17         val cachedMovies = dao.getAllMovies()
18         Timber.i("Loaded ${cachedMovies.size} movies from
19 cache.")
20
21         return try {
22             // Step 2: Fetch from API
23             val response =
24             RetrofitInstance.api.getPopularMovies(apiKey)
25             val movies = response.results.map {
26                 MovieEntity(
27                     id = it.id,
```

```

27         title = it.title,
28         overview = it.overview ?: "",
29         poster_path = it.posterPath ?: "",
30         vote_average = it.voteAverage
31     )
32
33     // Step 3: Cache to DB
34     dao.clearAll()
35     dao.insertAll(movies)
36
37     Timber.i("Fetched and cached ${movies.size} movies
from API.")
38     movies
39 } catch (e: Exception) {
40     Timber.e(e, "Error fetching movies. Using cache.")
41     cachedMovies
42 }
43 }
```

## B. Output Program



Gambar 1. Screenshot HomeScreen.kt dengan data dari API



Gambar 2. Screenshot Halaman Detail dengan data dari API

## C. Pembahasan

### 1. Pembahasan Praktikum:

#### JETPACK COMPOSE

##### MainActivity.kt :

Pada baris [1], package com.example.movieList digunakan untuk mendefinisikan bahwa file ini berada dalam package com.example.movieList. Pada baris [2–8], dilakukan import berbagai library yang dibutuhkan seperti ComponentActivity, setContent, elemen dari MaterialTheme, Surface, hingga Navigation.

Pada baris [10], kelas MainActivity didefinisikan dan mewarisi ComponentActivity yang merupakan komponen utama dalam aplikasi berbasis Jetpack Compose. Pada baris [11], fungsi onCreate() dioverride untuk menyiapkan tampilan awal aplikasi. Pada baris [12], super.onCreate(savedInstanceState) memanggil implementasi dari superclass. Pada baris [13], setContent digunakan untuk menentukan isi UI dengan menggunakan Jetpack Compose.

Pada baris [14], MovieListTheme digunakan untuk menerapkan tema aplikasi. Pada baris [15], Surface digunakan sebagai container dasar dengan warna latar belakang dari MaterialTheme. Pada baris [16], rememberNavController() dipanggil untuk membuat dan menyimpan instance NavController yang digunakan untuk navigasi. Pada baris [17],

`AppNavGraph(navController = navController)` memanggil fungsi navigasi utama aplikasi yang mendefinisikan alur navigasi antar layar.

Pada baris [21], `@Composable` menandai fungsi `MainPreview` sebagai fungsi komposabel untuk menampilkan preview UI. Pada baris [22], anotasi `@Preview` digunakan agar fungsi ini dapat dipreview di Android Studio dengan tampilan UI dan sistem. Pada baris [23–26], isi fungsi preview sama dengan `setContent` di `MainActivity`, yaitu menggunakan tema, membuat `navController`, dan menjalankan navigasi melalui `AppNavGraph`.

## NavGraph.kt

Pada baris [1], `package com.example.movieList.presentation.navigation` digunakan untuk mendefinisikan lokasi file ini dalam struktur package aplikasi. Pada baris [2], `import android.app.Application` mengimpor kelas `Application` untuk digunakan dalam pembuatan `ViewModel`. Pada baris [3]–[9], berbagai komponen Compose, Navigation, dan `ViewModel` diimpor untuk digunakan di dalam composable function. Pada baris [11], `object Routes` mendefinisikan konstanta untuk rute navigasi dalam aplikasi: `HOME` dan `DETAIL/{itemId}`.

Pada baris [16], `@Composable fun AppNavGraph(navController: NavHostController)` mendefinisikan fungsi composable yang berfungsi sebagai pengatur navigasi (navigation graph) utama. Pada baris [17], `val context = LocalContext.current.applicationContext as Application` mengambil konteks aplikasi agar bisa diteruskan ke `viewModelFactory`. Pada baris [18], `val viewModel = viewModel(...)` digunakan untuk membuat instance `MovieViewModel` menggunakan factory custom.

Pada baris [20], `NavHost(...)` digunakan untuk mendeklarasikan host navigasi dengan `navController` dan `startDestination` yang mengarah ke halaman home. Pada baris [22], `composable(Routes.HOME)` mendefinisikan rute halaman home dan memanggil `HomeScreen`, sambil meneruskan `navController` dan `viewModel`. Pada baris [26], `composable(route = Routes.DETAIL, arguments = ...)` mendefinisikan rute untuk halaman detail yang membutuhkan argumen `itemId` bertipe `Int`. Pada baris [28], `val itemId = ...` mengekstrak nilai `itemId` dari argumen navigasi. Pada baris [29], `DetailScreen(...)` dipanggil dengan `itemId`, `navController`, dan `viewModel` yang sama seperti halaman home untuk memastikan data tetap konsisten.

## HomeScreen.kt

Pada baris [1], `package com.example.movieList.ui.screen` menyatakan bahwa file ini berada dalam package `ui.screen` dari proyek aplikasi. Pada baris [2–27], berbagai `import` digunakan untuk memasukkan komponen Jetpack Compose, pengelolaan layout, warna, ukuran teks, `ViewModel`, dan library eksternal seperti `coil` untuk memuat gambar dan `Timber` untuk logging. Pada baris [29], anotasi `@Composable` menunjukkan bahwa fungsi `HomeScreen` adalah composable, yaitu fungsi yang menyusun UI dalam Jetpack Compose.

Pada baris [30], `fun HomeScreen(...)` mendefinisikan fungsi dengan parameter `navController` untuk navigasi dan `viewModel` untuk data film.

Pada baris [31], `val movieListState = viewModel.movieList.collectAsState()` digunakan untuk mengamati state dari daftar film secara reaktif. Pada baris [32], `val movieList = movieListState.value` menyimpan nilai terkini dari daftar film. Pada baris [33], `val context = LocalContext.current` mendapatkan konteks Android saat ini untuk keperluan tertentu seperti navigasi atau membuka intent.

Pada baris [35], `LazyColumn(...)` digunakan untuk membuat daftar scroll vertikal dengan padding dan jarak antar item. Pada baris [39], `itemsIndexed(movieList)` mengiterasi daftar film dan memberikan `Card` untuk setiap item. Pada baris [40], `Card(...)` membungkus setiap item film dengan bentuk dan warna latar tertentu. Pada baris [44], `Row(...)` menyusun gambar dan informasi film secara horizontal. Pada baris [47], `AsyncImage(...)` dari Coil digunakan untuk memuat poster film dari URL.

Pada baris [52], `Spacer(modifier = Modifier.width(12.dp))` memberikan jarak antara gambar dan teks. Pada baris [54], `Column(...)` menampung teks judul, deskripsi, dan tombol aksi. Pada baris [55], `Text(...)` menampilkan judul film dengan ukuran dan warna tertentu. Pada baris [59], `Text(...)` menampilkan ringkasan film dengan batasan maksimum empat baris. Pada baris [63], `Spacer(...)` memberi jarak sebelum tombol.

Pada baris [65], `Row(...)` menyusun tombol aksi secara horizontal. Pada baris [66], `Button(...)` mendefinisikan tombol "Detail" yang ketika diklik akan menavigasi ke layar detail dengan ID film, dan mencatat ID menggunakan `Timber`. Pada baris [71], `Text("Detail", ...)` memberi label pada tombol tersebut.

## DetailScreen.kt

Pada baris [1], `package com.example.movieList.ui.screen` menunjukkan bahwa file ini berada dalam package `ui.screen` dari proyek aplikasi. Pada baris [2–18], berbagai `import` digunakan untuk mengimpor komponen Jetpack Compose seperti layout, warna, teks, gambar dengan Coil, dan juga logging dengan Timber. Pada baris [20], anotasi `@Composable` menyatakan bahwa `DetailScreen` adalah fungsi composable yang membentuk UI di Jetpack Compose. Pada baris [21], fungsi `DetailScreen(...)` didefinisikan dengan tiga parameter: `itemId`, `navController`, dan `viewModel`.

Pada baris [23], `val item = viewModel.getMovieById(itemId)` digunakan untuk mengambil data film berdasarkan ID. Pada baris [25], `Timber.d(...)` mencatat ID dan judul film untuk keperluan debugging. Pada baris [27], `if (item != null)` memeriksa apakah data film tersedia sebelum menampilkan UI-nya.

Pada baris [28], `Column(...)` membuat layout vertikal untuk menyusun konten detail film. Pada baris [30], `modifier` digunakan untuk memenuhi ukuran layar, memberi latar belakang hitam, dan padding. Pada baris [34], `AsyncImage(...)` digunakan untuk menampilkan gambar poster film dari URL, dengan gaya klipping dan skala konten tertentu.

Pada baris [40], `Spacer(...)` memberi jarak antar elemen. Pada baris [42], `Text(...)` menampilkan judul film dengan ukuran besar, warna putih, dan tebal. Pada baris [46], `Text(...)` menampilkan rating film dalam bentuk bintang. Pada baris [50], `Spacer(...)` memberi jarak sebelum deskripsi. Pada baris [52], `Text(...)` menampilkan ringkasan film atau teks fallback jika tidak ada overview, dengan warna putih dan perataan teks `justify`.

Pada baris [57], `Spacer(...)` memberi jarak sebelum tombol. Pada baris [59], `Button(...)` membuat tombol "Back" dengan bentuk bulat dan warna ungu. Pada baris [63], `Text("Back")` memberi label pada tombol. Terakhir, pada baris [66], jika data film tidak ditemukan, maka `Text("Item not found")` ditampilkan dengan warna merah sebagai penanda kesalahan.

## **MovieViewModel.kt**

Pada baris [1], `package com.example.movieList.viewmodel` menunjukkan bahwa file ini merupakan bagian dari package `viewmodel` dalam proyek. Pada baris [3–9], berbagai `import` digunakan untuk mengimpor kelas-kelas penting seperti `Application`, `AndroidViewModel`, `coroutine`, `database`, `repository`, dan `logging` dengan `Timber`. Pada baris [11], kelas `MovieViewModel` didefinisikan sebagai subclass dari `AndroidViewModel`, yang memungkinkan class ini mengakses konteks `Application`.

Pada baris [13], variabel `_movieList` dideklarasikan sebagai `MutableStateFlow` yang menyimpan daftar `MovieEntity` kosong sebagai state internal. Pada baris [14], `movieList` dideklarasikan sebagai `StateFlow` untuk mengekspos data `_movieList` ke luar secara hanya-baca. Pada baris [16], objek `repository` dibuat dari `MovieRepository` dengan parameter `DAO` dan `context` aplikasi.

Pada baris [20], `apiKey` disimpan sebagai string konstanta API untuk mengakses data dari server eksternal. Pada baris [22], blok `init` digunakan untuk memanggil fungsi `fetchMovies()` saat `ViewModel` pertama kali dibuat. Pada baris [25], fungsi `fetchMovies()` berjalan dalam `viewModelScope.launch`, yang artinya proses berjalan secara asynchronous menggunakan `coroutine`. Pada baris [26], data film diambil dari `repository` dengan `getMovies(apiKey)`, lalu hasilnya disimpan ke `_movieList`.

Pada baris [30], fungsi `getMovieById(id: Int)` digunakan untuk mencari dan mengembalikan satu data film dari `_movieList` berdasarkan `id`. Jika tidak ditemukan, fungsi ini akan mengembalikan `null`.

## **MovieViewModelFactory.kt**

Pada baris [1], `package com.example.movieList.viewmodel` berfungsi untuk menunjukkan bahwa file ini merupakan bagian dari package `viewmodel`. Pada baris [3–5], `import` digunakan untuk mengimpor class-class penting dari Android seperti `Application`, `ViewModel`, dan `ViewModelProvider`, yang dibutuhkan untuk membuat dan mengelola `ViewModel`.

Pada baris [7], didefinisikan class `MovieViewModelFactory` yang mewarisi dari `ViewModelProvider.Factory`, dan berfungsi sebagai factory untuk membuat instance dari `MovieViewModel`. Pada baris [8], `private val application: Application` menyimpan context aplikasi yang akan diteruskan ke `ViewModel` saat dibuat.

Pada baris [9], override function `create()` digunakan untuk membuat instance dari `ViewModel`. Pada baris [10], terdapat kondisi `if` yang memeriksa apakah `modelClass` merupakan subclass dari `MovieViewModel`. Jika ya, maka pada baris [11], akan dibuat dan dikembalikan instance `MovieViewModel` dengan meneruskan `application`. Pada baris [13], jika `modelClass` bukan `MovieViewModel`, maka akan dilempar exception `IllegalArgumentException` dengan pesan bahwa class `ViewModel` tidak dikenal.

## MovieApp.kt

Pada baris [1], `package com.example.movieListxml` berfungsi untuk mendefinisikan lokasi package dari class `MovieApp`, agar sesuai dengan struktur proyek Android. Pada baris [2] dan [3], `import android.app.Application` dan `import timber.log.Timber` digunakan untuk mengimpor class `Application` yang menjadi superclass dari `MovieApp`, serta library `Timber` yang digunakan untuk logging.

Pada baris [5], `class MovieApp : Application()` mendefinisikan class `MovieApp` sebagai turunan dari class `Application`, yang berarti kode di dalamnya akan dijalankan saat aplikasi pertama kali dibuka. Pada baris [6], `override fun onCreate()` digunakan untuk menimpa method `onCreate()` bawaan dari class `Application`.

Pada baris [7], `super.onCreate()` dipanggil agar inisialisasi bawaan dari `Application` tetap berjalan. Pada baris [9], terdapat pengecekan `if (BuildConfig.DEBUG)` yang berfungsi untuk memastikan bahwa blok kode di dalamnya hanya dijalankan saat mode debug, bukan pada rilis aplikasi.

Pada baris [10], `Timber.plant(Timber.DebugTree())` berfungsi untuk menginisialisasi `Timber` dengan `DebugTree`, sehingga kita bisa menampilkan log saat proses debugging. Pada baris [11], `Timber.i("Timber initialized in MovieApp")` mencetak log informasi bahwa `Timber` berhasil diinisialisasi.

## Movie ApiService.kt

Pada baris [1], `package com.example.movieList.data.api` berfungsi untuk menyatakan bahwa file ini termasuk dalam package `data.api`, yang mengelompokkan file sesuai fungsinya dalam struktur proyek. Pada baris [3–4], `import` digunakan untuk memasukkan class `MovieResponse` dan anotasi `Retrofit @GET` serta `@Query`, yang diperlukan untuk mendeklarasikan endpoint API.

Pada baris [6], didefinisikan interface `Movie ApiService` yang digunakan untuk mendeklarasikan layanan API menggunakan `Retrofit`. Pada baris [7], anotasi

`@GET("movie/popular")` menunjukkan bahwa fungsi berikutnya akan melakukan permintaan HTTP GET ke endpoint `movie/popular`. Pada baris [8], fungsi `getPopularMovies` dideklarasikan sebagai fungsi suspend, yang berarti dapat dipanggil dalam coroutine, dan menerima parameter `apiKey` yang akan disisipkan sebagai query parameter dengan anotasi `@Query("api_key")`. Pada baris [9], fungsi ini akan mengembalikan objek bertipe `MovieResponse`, yang merepresentasikan respons dari server.

### RetrofitInstance.kt

Pada baris [1], `package com.example.movieList.data.api` berfungsi untuk menandai bahwa file ini merupakan bagian dari package `data.api`, yang mengelompokkan file berdasarkan fungsi. Pada baris [3–8], `import` digunakan untuk memasukkan berbagai dependensi eksternal seperti converter JSON dari `kotlinx.serialization`, `OkHttpClient` untuk koneksi jaringan, dan `Retrofit` untuk membuat instance API service.

Pada baris [10], dideklarasikan object `RetrofitInstance` yang bersifat singleton untuk memastikan hanya satu instance Retrofit yang digunakan dalam aplikasi. Pada baris [12], dibuat `loggingInterceptor` dengan level `BODY` untuk mencatat semua detail request dan response HTTP. Pada baris [15], dibangun objek `OkHttpClient` dan `logging interceptor` ditambahkan ke dalamnya.

Pada baris [18], didefinisikan konfigurasi `Json` dengan properti `ignoreUnknownKeys = true`, artinya jika ada field JSON yang tidak dikenali, akan diabaikan. Pada baris [21], dideklarasikan `retrofit` menggunakan builder pattern, dengan base URL mengarah ke `https://api.themoviedb.org/3/`, client yang sudah dikonfigurasi, dan converter untuk parsing JSON.

Pada baris [27], `val api` dideklarasikan sebagai instance dari `Movie ApiService` yang dibuat dari `retrofit.create(Movie ApiService::class.java)`, dan dipanggil secara lazy agar hanya dibuat ketika dibutuhkan.

### MovieDao.kt

Pada baris [1], `package com.example.movieList.data.db` berfungsi untuk menandakan bahwa file ini berada dalam package `data.db`, yang mengelompokkan semua file terkait database. Pada baris [3–6], `import` digunakan untuk mengimpor class dan anotasi dari Room, yaitu `Dao`, `Insert`, `OnConflictStrategy`, dan `Query`, yang diperlukan untuk mendeklarasikan akses ke database.

Pada baris [8], `@Dao` merupakan anotasi yang digunakan untuk menandakan bahwa `MovieDao` adalah interface yang berfungsi sebagai Data Access Object untuk entitas movie. Pada baris [9], `@Query("SELECT * FROM movies")` digunakan untuk menulis query SQL yang mengambil seluruh data dari tabel `movies`, dan method `getAllMovies()` akan mengembalikan hasilnya dalam bentuk list `MovieEntity`.

Pada baris [12], `@Insert(onConflict = OnConflictStrategy.REPLACE)` digunakan untuk menyisipkan data ke tabel `movies` dan akan menggantikan data lama jika terjadi konflik. Method `insertAll()` menerima parameter berupa list dari `MovieEntity`. Pada baris [15], `@Query("DELETE FROM movies")` digunakan untuk menghapus seluruh data dari tabel `movies`, dan method `clearAll()` menjalankan perintah tersebut secara asynchronous.

## **MovieDatabase.kt**

Pada baris [1], `package com.example.movieList.data.db` berfungsi untuk menandakan bahwa file ini berada dalam package `data.db`, yaitu bagian dari struktur aplikasi yang menangani penyimpanan data lokal. Pada baris [3–6], `import` digunakan untuk mengimpor class dan fungsi dari Android dan Room seperti `Context`, `Database`, `Room`, dan `RoomDatabase` yang dibutuhkan untuk membuat database lokal.

Pada baris [8], anotasi `@Database` berfungsi untuk mendeklarasikan bahwa `MovieDatabase` adalah database Room dengan satu entitas yaitu `MovieEntity`, versi 1, dan tidak mengekspor skema. Pada baris [9], `abstract class MovieDatabase : RoomDatabase()` mendefinisikan class abstrak yang mewarisi dari `RoomDatabase` dan akan digunakan untuk mengakses DAO.

Pada baris [10], `abstract fun movieDao(): MovieDao` mendeklarasikan fungsi abstrak untuk mengambil instance dari DAO (`MovieDao`). Pada baris [12], `companion object` digunakan agar fungsi dan variabel di dalamnya bersifat statis dan bisa diakses tanpa membuat objek `MovieDatabase`.

Pada baris [13], `@Volatile private var INSTANCE menyimpan instance tunggal dari MovieDatabase yang bersifat volatile agar perubahan nilainya terlihat oleh thread lain.` Pada baris [15], `getDatabase(context: Context)` adalah fungsi yang mengembalikan instance `MovieDatabase`, dan pada baris [16–21], dilakukan proses singleton menggunakan `synchronized` agar hanya satu instance database yang dibuat dengan `Room.databaseBuilder()`, menggunakan nama "movie\_database". Pada baris [22–23], instance disimpan ke `INSTANCE` dan dikembalikan sebagai hasil fungsi.

## **MovieEntity.kt**

Pada baris [1], `package com.example.movieList.data.db` berfungsi untuk menunjukkan bahwa file ini merupakan bagian dari package `data.db`, yaitu tempat penyimpanan data lokal dalam struktur aplikasi. Pada baris [3–4], `import` digunakan untuk mengimpor anotasi `@Entity` dan `@PrimaryKey` dari Room, yang diperlukan untuk mendefinisikan struktur tabel pada database.

Pada baris [6], `@Entity(tableName = "movies")` digunakan untuk menandakan bahwa class `MovieEntity` adalah sebuah entitas Room yang akan disimpan dalam tabel bernama `movies`. Pada baris [7], `data class MovieEntity` mendefinisikan class data yang merepresentasikan satu baris data dalam tabel tersebut.

Pada baris [8], `@PrimaryKey val id: Int` berfungsi untuk menandai bahwa properti `id` adalah primary key dari entitas ini. Pada baris [9–12], properti `title`, `overview`, `poster_path`, dan `vote_average` masing-masing digunakan untuk menyimpan judul film, deskripsi singkat, path poster, dan nilai rating film. Kelima properti ini akan menjadi kolom pada tabel `movies` di database Room.

### **MovieResponse.kt**

Pada baris [1], `package com.example.movieList.domain.model` berfungsi untuk mendefinisikan package tempat kode ini berada, yaitu model domain aplikasi. Pada baris [3–4], `import` digunakan untuk memasukkan anotasi `@Serializable` dan `@SerializedName` dari `kotlinx.serialization` yang memungkinkan objek data ini dapat diserialisasi dan deserialisasi dari/ke JSON.

Pada baris [6], `@Serializable` menandakan bahwa class `MovieResponse` bisa diubah menjadi format JSON dan sebaliknya. Pada baris [7], `data class MovieResponse` mendefinisikan model data yang berisi properti `results`, yaitu daftar objek `Movie`.

Pada baris [11], `@Serializable` juga menandakan bahwa class `Movie` dapat diserialisasi. Pada baris [12–17], `data class Movie` mendefinisikan properti `id`, `title`, dan `overview` sebagai informasi dasar film. Pada baris [15–16], anotasi `@SerializedName` mengaitkan properti `posterPath` dan `voteAverage` dengan nama kunci JSON `poster_path` dan `vote_average` agar sesuai dengan data dari API.

### **MovieRepository.kt**

Pada baris [1], `package com.example.movieList.domain.repository` berfungsi untuk mendefinisikan package tempat kode repository ini berada. Pada baris [3–7], `import` berfungsi untuk memasukkan dependensi seperti `RetrofitInstance`, `MovieDao`, `MovieDatabase`, `MovieEntity`, `coroutine dispatcher`, dan `Timber` untuk logging.

Pada baris [9], `class MovieRepository` mendefinisikan kelas repository dengan dua properti utama, yaitu `dao` untuk akses database lokal dan `context` untuk konteks aplikasi. Pada baris [11], `suspend fun getMovies(apiKey: String): List<MovieEntity>` adalah fungsi asinkron yang bertugas mengambil data film.

Pada baris [13], kode memuat data film dari cache lokal dengan memanggil `dao.getAllMovies()`, dan pada baris [14], mencatat jumlah film yang diambil dari cache menggunakan Timber. Pada baris [16], kode mencoba mengambil data film terbaru dari API menggunakan `RetrofitInstance`.

Pada baris [18–24], data hasil response API diubah menjadi daftar `MovieEntity` untuk disimpan ke database lokal. Pada baris [27–28], database lokal dibersihkan lalu data baru dimasukkan sebagai cache. Pada baris [30], logging berhasilnya pengambilan dan penyimpanan data film dari API. Pada baris [32–34], jika terjadi kesalahan saat mengambil data dari API, maka error dicatat dan data dari cache dikembalikan.

#### **D. Tautan Git**

Berikut adalah tautan untuk source code yang telah dibuat.

<https://github.com/MuhammadFiras/Praktikum-Mobile/tree/main>