LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Muhammad Adh-Dhiya'Us Salim NIM. 2310817210022

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 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 Prakitkum ini dikerjakan oleh:

Nama Praktikan : Muhammad Adh-Dhiya'Us Salim
NIM : 2310817210022

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Muhammad Raka Azwar NIM. 2210817210012	Ir. Eka Setya Wijaya, S.T., M.Kom. NIP. 198205082008011010

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

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

Data / Local / AppDatabase

```
1  package com.example.modul5.data.local
2
3  import android.content.Context
4  import androidx.room.Database
5  import androidx.room.Room
6  import androidx.room.RoomDatabase
7  import com.example.modul5.data.model.Movie
8
9  @Database(entities = [Movie::class], version = 2, exportSchema
10 = false)
11 abstract class AppDatabase : RoomDatabase() {
12     abstract fun movieDao(): MovieDao
13
14     companion object {
15         @Volatile
16         private var INSTANCE: AppDatabase? = null
17
18         fun getDatabase(context: Context): AppDatabase {
19             return INSTANCE ?: synchronized(this) {
20                 val instance = Room.databaseBuilder(
21                     context.applicationContext,
22                     AppDatabase::class.java,
23                     "movie_database"
24                 ).fallbackToDestructiveMigration()
```

```
                    .build()
                INSTANCE = instance
                instance

            }
        }
    }
}
```

Tabel 1 Source Code Soal 1

Data / Local / MovieDao

```
package com.example.modul5.data.local

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update
import com.example.modul5.data.model.Movie
import kotlinx.coroutines.flow.Flow

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

    @Query("SELECT * FROM movies WHERE isBookmarked = 1")
    fun getBookmarkedMovies(): Flow<List<Movie>>

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

    @Update
    suspend fun updateMovie(movie: Movie)

    @Query("DELETE FROM movies")
    suspend fun clearAll()
}
```

Tabel 2 Source Code Soal 1

Data / Model / Movie

```
package com.example.modul5.data.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

@Serializable
@Entity(tableName = "movies")
data class Movie(
    @PrimaryKey
    @SerialName("id")
    val id: Int,
```

```
    @SerialName("name")
    val                         title:                          String,

    @SerialName("first_air_date")
    val                          year:                          String,

    @SerialName("overview")
    val                          plot:                          String,

    @SerialName("poster_path")
    val                     posterPath:                     String?,

    val       isBookmarked:       Boolean       =       false,
)

@Serializable
data                    class                    MovieApiResponse(
    val                    results:                    List<Movie>
)
```

Tabel 3 Source Code Soal 1

Data / remote / ApiService

```
1  package                              com.example.modul5.data.remote
2
3  import           com.example.modul5.data.model.MovieApiResponse
4  import                                        retrofit2.http.GET
5  import                                      retrofit2.http.Query
6
7  interface                         ApiService                        {
8      @GET("discover/tv")
9      suspend                  fun                 getPopularMovies(
           @Query("api_key")             apiKey:             String,
           @Query("with_original_language")       origin_language:
   String                              =                          "ko",
           @Query("sort_by")  sortBy:  String  =  "popularity.desc",
           @Query("with_watch_providers") providers: String = "8",
           @Query("watch_region")    region:    String    =    "ID",
       ):                                              MovieApiResponse
   }
```

Tabel 4 Source Code Soal 1

Data / remote / RetrofitClient

```
1  package                                    com.example.modul5.data.remote
2
3  import
4  com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverter
5  Factory
6  import                                    kotlinx.serialization.json.Json
7  import                           okhttp3.MediaType.Companion.toMediaType
8  import                                              retrofit2.Retrofit
9
```

```
object RetrofitClient {
    private const val BASE_URL = "https://api.themoviedb.org/3/"
    private val json = Json { ignoreUnknownKeys = true }

    val instance: ApiService by lazy {
        val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)

.addConverterFactory(json.asConverterFactory("application/json".toMediaType()))
            .build()
        retrofit.create(ApiService::class.java)
    }
}
```

Tabel 5 Source Code Soal 1

Data / repository / MovieRepository

```
1  package com.example.modul5.data.repository
2
3  import com.example.modul5.BuildConfig
4  import com.example.modul5.data.local.MovieDao
5  import com.example.modul5.data.model.Movie
6  import com.example.modul5.data.remote.ApiService
7  import kotlinx.coroutines.Dispatchers
8  import kotlinx.coroutines.flow.first
9  import kotlinx.coroutines.withContext

   class MovieRepository(
       private val apiService: ApiService,
       private val movieDao: MovieDao
   ) {
       val movies = movieDao.getAllMovies()
       val bookmarkedMovies = movieDao.getBookmarkedMovies()

       suspend fun updateMovie(movie: Movie) {
           movieDao.updateMovie(movie)
       }

       suspend fun refreshMovies() {
           withContext(Dispatchers.IO) {
               try {
                   val oldMovies = movieDao.getAllMovies().first()
                   val bookmarkedIds = oldMovies.filter {
   it.isBookmarked }.map { it.id }.toSet()

                   val newMoviesFromApi =
   apiService.getPopularMovies(BuildConfig.TMDB_API_KEY).results

                   val newMovies = newMoviesFromApi.map { apiMovie
   ->
                       apiMovie.copy(isBookmarked =
   bookmarkedIds.contains(apiMovie.id))
                   }
```

```
                movieDao.insertAll(newMovies)
            }        catch        (e:        Exception)        {
                e.printStackTrace()
            }
        }
    }
}
```

Tabel 6 Source Code Soal 1

Data / navigation / AppNavigation

```
1  package                              com.example.modul5.navigation
2
3  import                         androidx.compose.runtime.Composable
4  import            androidx.lifecycle.viewmodel.compose.viewModel
5  import                         androidx.navigation.compose.NavHost
6  import                    androidx.navigation.compose.composable
7  import         androidx.navigation.compose.rememberNavController
8  import                  com.example.modul5.ui.theme.DetailScreen
9  import               com.example.modul5.ui.theme.MovieListScreen
   import               com.example.modul5.viewmodel.MovieViewModel
   import         com.example.modul5.viewmodel.SharedMovieViewModel

   @Composable
   fun                                                AppNavigation(
       movieViewModel:                               MovieViewModel,
       sharedViewModel:                          SharedMovieViewModel,
   )                                                                 {
       val       navController       =       rememberNavController()

       NavHost(navController  =  navController,  startDestination  =
   Screen.MovieList.route)                                           {
           composable(Screen.MovieList.route)                        {
               MovieListScreen(
                   navController          =          navController,
                   movieViewModel         =         movieViewModel,
                   sharedViewModel         =         sharedViewModel
               )
           }
           composable(Screen.Detail.route)                           {
               DetailScreen(sharedViewModel    =    sharedViewModel)
           }
       }
   }
```

Tabel 7 Source Code Soal 1

Data / navigation / Screen

```
package com.example.modul5.navigation

sealed class Screen(val route: String) {
    object MovieList : Screen("movie_list")
    object Detail : Screen("movie_detail")
}
```

Tabel 8 Source Code Soal 1

Ui.theme / BookmarkScreen

```
package com.example.modul5.ui.theme

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.navigation.NavController
import com.example.modul5.viewmodel.MovieViewModel
import com.example.modul5.viewmodel.SharedMovieViewModel

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BookmarkScreen(
    navController: NavController,
    movieViewModel: MovieViewModel,
    sharedViewModel: SharedMovieViewModel
) {

    val bookmarkedList by
movieViewModel.bookmarkedMovies.collectAsState()

    Scaffold(
        topBar = {
            TopAppBar(title = { Text("Daftar Bookmark") })
        }
    ) { innerPadding ->
        if (bookmarkedList.isEmpty()) {
            Box(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(innerPadding),
                contentAlignment = Alignment.Center
            ) {
```

```
                Text(
                    text = "Belum ada serial yang di-bookmark.",
                    style = MaterialTheme.typography.bodyLarge,
                    textAlign = TextAlign.Center
                )
            }
        } else {
            LazyColumn(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(innerPadding)
            ) {
                items(bookmarkedList) { movie ->
                    MovieListItem(
                        movie = movie,
                        movieViewModel = movieViewModel,
                        sharedViewModel = sharedViewModel,
                        navController = navController
                    )
                }
            }
        }
    }
}
```

Tabel 9 Source Code Soal 1

Ui.theme / DetailScreen

```
1  package com.example.modul5.ui.theme
2
3  import androidx.compose.foundation.layout.*
4  import androidx.compose.foundation.lazy.LazyColumn
5  import androidx.compose.foundation.shape.RoundedCornerShape
6  import androidx.compose.material3.*
7  import androidx.compose.runtime.Composable
8  import androidx.compose.ui.Modifier
9  import androidx.compose.ui.draw.clip
   import androidx.compose.ui.layout.ContentScale
   import androidx.compose.ui.unit.dp
   import coil.compose.AsyncImage
   import com.example.modul5.viewmodel.SharedMovieViewModel

   @OptIn(ExperimentalMaterial3Api::class)
   @Composable
   fun DetailScreen(sharedViewModel: SharedMovieViewModel) {
       val selectedMovie = sharedViewModel.selectedMovie ?: return

       Scaffold(
           topBar = {
               TopAppBar(
                   title = { Text("Movie Detail") }
               )
           }
       ) { paddingValues ->
           LazyColumn(
```

```
            modifier                    =                    Modifier
                .fillMaxSize()
                .padding(paddingValues)
                .padding(16.dp)
        )                                                                {
            item                                                         {
                AsyncImage(
                    model                                                =
"https://image.tmdb.org/t/p/w500${selectedMovie.posterPath}",
                    contentDescription  =  selectedMovie.title,
                    modifier             =              Modifier
                        .fillMaxWidth()
                        .height(500.dp)
                        .clip(RoundedCornerShape(16.dp)),
                    contentScale       =       ContentScale.Crop
                )
                Spacer(modifier    =    Modifier.height(16.dp))
            }

            item                                                         {
                Text(
                    text        =        "${selectedMovie.title}
(${selectedMovie.year.substring(0,                          4)})",
                    style                                                =
MaterialTheme.typography.headlineMedium
                )
                Spacer(modifier    =    Modifier.height(16.dp))
            }

            item                                                         {
                Text(
                    text              =              "Overview",
                    style                                                =
MaterialTheme.typography.titleMedium
                )
                Spacer(modifier     =     Modifier.height(8.dp))
                Text(
                    text          =          selectedMovie.plot,
                    style  =  MaterialTheme.typography.bodyLarge
                )
            }
        }
    }
}
```

Tabel 10 Source Code Soal 1

Ui.theme / MainScreen

```
1  package com.example.modul5.ui.theme
2
3  import androidx.compose.foundation.layout.padding
4  import androidx.compose.material.icons.Icons
5  import androidx.compose.material.icons.filled.Bookmark
6  import androidx.compose.material.icons.filled.Home
7  import androidx.compose.material3.*
8  import androidx.compose.runtime.Composable
9  import androidx.compose.runtime.getValue
   import androidx.compose.ui.Modifier
   import androidx.navigation.NavDestination.Companion.hierarchy
   import
   androidx.navigation.NavGraph.Companion.findStartDestination
   import androidx.navigation.compose.NavHost
   import androidx.navigation.compose.composable
   import androidx.navigation.compose.currentBackStackEntryAsState
   import androidx.navigation.compose.rememberNavController
   import com.example.modul5.navigation.Screen
   import com.example.modul5.viewmodel.MovieViewModel
   import com.example.modul5.viewmodel.SharedMovieViewModel

   @OptIn(ExperimentalMaterial3Api::class)
   @Composable
   fun MainScreen(movieViewModel: MovieViewModel, sharedViewModel:
   SharedMovieViewModel) {
       val navController = rememberNavController()

       Scaffold(
           bottomBar = {
               NavigationBar {
                   val navBackStackEntry by
   navController.currentBackStackEntryAsState()
                   val currentDestination =
   navBackStackEntry?.destination

                   // Item untuk Home
                   NavigationBarItem(
                       icon = { Icon(Icons.Default.Home,
   contentDescription = "Home") },
                       label = { Text("Home") },
                       selected =
   currentDestination?.hierarchy?.any { it.route ==
   Screen.MovieList.route } == true,
                       onClick = {

   navController.navigate(Screen.MovieList.route) {

   popUpTo(navController.graph.findStartDestination().id) {
   saveState = true }
                               launchSingleTop = true
                               restoreState = true
                           }
                       }
                   )
```

```
                // Item untuk Bookmark
                NavigationBarItem(
                    icon = { Icon(Icons.Default.Bookmark,
contentDescription = "Bookmark") },
                    label = { Text("Bookmarks") },
                    selected = currentDestination?.hierarchy?.any { it.route ==
"bookmark_screen" } == true,
                    onClick = {

navController.navigate("bookmark_screen") {

popUpTo(navController.graph.findStartDestination().id) {
saveState = true }
                        launchSingleTop = true
                        restoreState = true
                    }
                }
            )
        }
    ) { innerPadding ->
        NavHost(
            navController = navController,
            startDestination = Screen.MovieList.route,
            modifier = Modifier.padding(innerPadding)
        ) {
            composable(Screen.MovieList.route) {
                MovieListScreen(
                    navController = navController,
                    movieViewModel = movieViewModel,
                    sharedViewModel = sharedViewModel
                )
            }
            composable("bookmark_screen") {
                BookmarkScreen(
                    navController = navController,
                    movieViewModel = movieViewModel,
                    sharedViewModel = sharedViewModel
                )
            }
            composable(Screen.Detail.route) {
                DetailScreen(sharedViewModel = sharedViewModel)
            }
        }
    }
}
```

Tabel 11 Source Code Soal 1

Ui.theme / MovieLisrtItem

```
1  package com.example.modul5.ui.theme
2
3  import android.content.Intent
4  import android.net.Uri
5  import androidx.compose.foundation.layout.*
6  import androidx.compose.foundation.shape.RoundedCornerShape
7  import androidx.compose.material.icons.Icons
8  import androidx.compose.material.icons.filled.Bookmark
9  import androidx.compose.material.icons.outlined.BookmarkBorder
   import androidx.compose.material3.*
   import androidx.compose.runtime.Composable
   import androidx.compose.ui.Alignment
   import androidx.compose.ui.Modifier
   import androidx.compose.ui.draw.clip
   import androidx.compose.ui.layout.ContentScale
   import androidx.compose.ui.platform.LocalContext
   import androidx.compose.ui.text.font.FontWeight
   import androidx.compose.ui.text.style.TextOverflow
   import androidx.compose.ui.unit.dp
   import androidx.navigation.NavController
   import coil.compose.AsyncImage
   import com.example.modul5.data.model.Movie
   import com.example.modul5.navigation.Screen
   import com.example.modul5.viewmodel.MovieViewModel
   import com.example.modul5.viewmodel.SharedMovieViewModel

   @OptIn(ExperimentalMaterial3Api::class)
   @Composable
   fun MovieListItem(
       movie: Movie,
       movieViewModel: MovieViewModel,
       sharedViewModel: SharedMovieViewModel,
       navController: NavController
   ) {
       val context = LocalContext.current

       Card(
           modifier = Modifier
               .fillMaxWidth()
               .padding(vertical = 8.dp, horizontal = 16.dp),
           elevation = CardDefaults.cardElevation(4.dp)
       ) {
           Row(modifier = Modifier.padding(16.dp)) {
               AsyncImage(
                   model =
   "https://image.tmdb.org/t/p/w500${movie.posterPath}",
                   contentDescription = movie.title,
                   modifier = Modifier
                       .size(width = 100.dp, height = 150.dp)
                       .clip(RoundedCornerShape(8.dp)),
                   contentScale = ContentScale.Crop
               )

               Spacer(modifier = Modifier.width(16.dp))
```

```
            Column(modifier = Modifier.weight(1f)) {
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    verticalAlignment = Alignment.CenterVertically,
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
                    Text(
                        text = movie.title,
                        style = MaterialTheme.typography.titleMedium,
                        fontWeight = FontWeight.Bold,
                        modifier = Modifier.weight(1f),
                        maxLines = 2,
                        overflow = TextOverflow.Ellipsis
                    )
                    IconButton(onClick = { movieViewModel.toggleBookmark(movie) }) {
                        Icon(
                            imageVector = if (movie.isBookmarked) Icons.Filled.Bookmark else Icons.Outlined.BookmarkBorder,
                            contentDescription = "Bookmark",
                            tint = MaterialTheme.colorScheme.primary
                        )
                    }
                }

                // Tahun Rilis
                Text(
                    text = movie.year?.substring(0, 4) ?: "N/A",
                    style = MaterialTheme.typography.bodySmall
                )

                Spacer(modifier = Modifier.height(8.dp))

                // Plot
                Text(
                    text = movie.plot ?: "No overview available.",
                    style = MaterialTheme.typography.bodySmall,
                    maxLines = 3,
                    overflow = TextOverflow.Ellipsis
                )

                Spacer(modifier = Modifier.weight(1f))

                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.End
                ) {
                    Button(onClick = {
                        val query = Uri.encode(movie.title)
```

```
                        val intent = Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.imdb.com/find?q=$query"))
                        context.startActivity(intent)
                    }, modifier = Modifier.padding(end = 8.dp))
{
                        Text("IMDB")
                    }

                    Button(onClick              =              {
                        sharedViewModel.selectMovie(movie)

navController.navigate(Screen.Detail.route)
                    })                                          {
                        Text("Detail")
                    }
                }
            }
        }
    }
}
```

Tabel 12 Source Code Soal 1

Ui.theme / MovieListScreen

```
1   package                                com.example.modul5.ui.theme
2
3   import                       androidx.compose.foundation.layout.*
4   import                androidx.compose.foundation.lazy.LazyColumn
5   import                     androidx.compose.foundation.lazy.items
6   import                               androidx.compose.material3.*
7   import                          androidx.compose.runtime.Composable
8   import                      androidx.compose.runtime.collectAsState
9   import                            androidx.compose.runtime.getValue
    import                                androidx.compose.ui.Alignment
    import                                 androidx.compose.ui.Modifier
    import                              androidx.navigation.NavController
    import                com.example.modul5.viewmodel.MovieViewModel
    import         com.example.modul5.viewmodel.SharedMovieViewModel

    @OptIn(ExperimentalMaterial3Api::class)
    @Composable
    fun                                              MovieListScreen(
        navController:                                 NavController,
        movieViewModel:                               MovieViewModel,
        sharedViewModel:                          SharedMovieViewModel
    )                                                                {
        val   movieList   by   movieViewModel.movies.collectAsState()

        Scaffold(
            topBar                          =                       {
                TopAppBar(title   =   {   Text("KDrama   Cinema")   })
            }
        )               {               paddingValues               ->
            if                 (movieList.isEmpty())                 {
                Box(
```

```
                modifier                                                =
Modifier.fillMaxSize().padding(paddingValues),
                contentAlignment       =       Alignment.Center
            )                                                           {
                CircularProgressIndicator()
            }
        }                           else                                {
            LazyColumn(modifier                                         =
Modifier.padding(paddingValues))                                        {
                items(movieList)        {        movie        ->
                    MovieListItem(
                        movie               =               movie,
                        movieViewModel      =     movieViewModel,
                        sharedViewModel     =    sharedViewModel,
                        navController       =      navController
                    )
                }
            }
        }
    }
}
```

Tabel 13 Source Code Soal 1

Viewmodel / MovieViewModel

```
1   package                              com.example.modul5.viewmodel
2
3   import                               androidx.lifecycle.ViewModel
4   import                           androidx.lifecycle.viewModelScope
5   import                          com.example.modul5.data.model.Movie
6   import       com.example.modul5.data.repository.MovieRepository
7   import                      kotlinx.coroutines.flow.SharingStarted
8   import                           kotlinx.coroutines.flow.StateFlow
9   import                             kotlinx.coroutines.flow.stateIn
    import                                   kotlinx.coroutines.launch

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

        val  movies:  StateFlow<List<Movie>>  =  repository.movies
            .stateIn(
                scope                =                viewModelScope,
                started   =   SharingStarted.WhileSubscribed(5000L),
                initialValue               =               emptyList()
            )

        val     bookmarkedMovies:      StateFlow<List<Movie>>      =
    repository.bookmarkedMovies
            .stateIn(
                scope                =                viewModelScope,
                started   =   SharingStarted.WhileSubscribed(5000L),
                initialValue               =               emptyList()
            )

        init                                                            {
```

```
        viewModelScope.launch {
            repository.refreshMovies()
        }
    }

    fun toggleBookmark(movie: Movie) {
        viewModelScope.launch {
            val updatedMovie = movie.copy(isBookmarked =
!movie.isBookmarked)
            repository.updateMovie(updatedMovie)
        }
    }
}
```

Tabel 14 Source Code Soal 1

Viewmodel / MovieViewModelFactory

```
package com.example.modul5.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import com.example.modul5.data.repository.MovieRepository

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

Tabel 15 Source Code Soal 1

Viewmodel / SharedViewModel

```
package com.example.modul5.viewmodel

import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import com.example.modul5.data.model.Movie

class SharedMovieViewModel : ViewModel() {
    var selectedMovie by mutableStateOf<Movie?>(null)
        private set

    fun selectMovie(movie: Movie) {
        selectedMovie = movie
```

```
    }
}
```

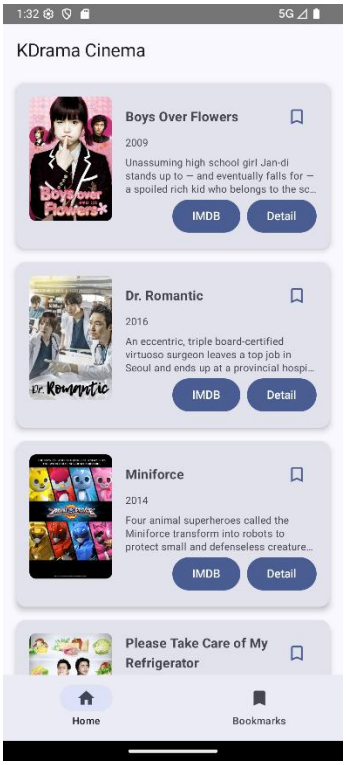
Tabel 16 Source Code Soal 1

MainActivity.kt

```
package com.example.modul5

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.lifecycle.ViewModelProvider
import com.example.modul5.data.local.AppDatabase
import com.example.modul5.data.remote.RetrofitClient
import com.example.modul5.data.repository.MovieRepository
import com.example.modul5.ui.theme.MainScreen
import com.example.modul5.ui.theme.Modul5Theme
import com.example.modul5.viewmodel.MovieViewModel
import com.example.modul5.viewmodel.MovieViewModelFactory
import com.example.modul5.viewmodel.SharedMovieViewModel

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

        val database = AppDatabase.getDatabase(applicationContext)
        val apiService = RetrofitClient.instance
        val repository = MovieRepository(apiService, database.movieDao())
        val factory = MovieViewModelFactory(repository)

        val movieViewModel: MovieViewModel = ViewModelProvider(this, factory)[MovieViewModel::class.java]
        val sharedViewModel: SharedMovieViewModel = ViewModelProvider(this)[SharedMovieViewModel::class.java]

        setContent {
            Modul5Theme {
                MainScreen(
                    movieViewModel = movieViewModel,
                    sharedViewModel = sharedViewModel
                )
            }
        }
    }
}
```

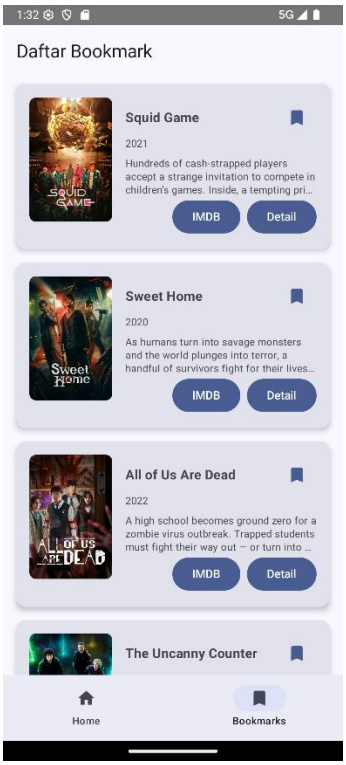
Tabel 17 Source Code Soal 1

B. Output Program



Gambar 1 Screenshot Hasil Jawaban Soal 1

Gambar 2 Screenshot Hasil Jawaban Soal 1



Gambar 3 Screenshot Hasil Jawaban Soal 1

C. Pembahasan

1. Lapisan Data (Data Layer)

Lapisan ini bertanggung jawab atas penyediaan dan manajemen data aplikasi dari berbagai sumber.

1.1. Model Data (data/model)

- Berkas: Movie.kt (Definisi Entitas Film)

Berkas ini mendefinisikan struktur data untuk sebuah film, yang berfungsi sebagai model untuk data dari API dan juga sebagai skema tabel untuk database lokal.

- **Baris 5: package com.example.modul5.data.model** Mendeklarasikan bahwa berkas ini merupakan bagian dari paket com.example.modul5.data.model.
- **Baris 7-10: import ...** Mengimpor kelas-kelas yang diperlukan, seperti Entity dan PrimaryKey dari Room, serta SerialName dan Serializable dari KotlinX Serialization.
- **Baris 12: @Serializable** Anotasi ini menandakan bahwa kelas Movie dapat diubah (diserialisasi) menjadi format lain seperti JSON dan sebaliknya (deserialisasi). Hal ini krusial untuk memproses respons dari API.
- **Baris 13: @Entity(tableName = "movies")** Anotasi dari pustaka Room yang menginstruksikan bahwa kelas Movie ini merepresentasikan sebuah tabel dalam database dengan nama movies.
- **Baris 14: data class Movie(...)** Mendeklarasikan Movie sebagai sebuah data class. Ini adalah cara ringkas di Kotlin untuk membuat kelas yang tujuan utamanya adalah menyimpan data.
- **Baris 15: @PrimaryKey** Menetapkan properti id yang didefinisikan pada baris berikutnya sebagai kunci utama (*primary key*) dari tabel movies.

- **Baris 16: val id: Int** Mendefinisikan properti id dengan tipe data Int untuk menyimpan identifikasi unik setiap film.
- **Baris 18: @SerialName("poster_path")** Memetakan atribut poster_path dalam data JSON dari API ke properti posterPath di dalam kelas ini.
- **Baris 19: val posterPath: String** Mendefinisikan properti posterPath dengan tipe String untuk menyimpan path ke gambar poster film.
- **Baris 21: val title: String** Mendefinisikan properti untuk menyimpan judul film.
- **Baris 23: val overview: String** Mendefinisikan properti untuk menyimpan sinopsis atau deskripsi film.
- **Baris 25: var isBookmarked: Boolean = false** Mendefinisikan properti isBookmarked dengan nilai awal false. Properti ini digunakan untuk melacak apakah sebuah film telah ditandai oleh pengguna.

1.2. Sumber Data Jarak Jauh (data/remote)

- Berkas: ApiService.kt (Definisi Layanan API)

Berkas ini mendefinisikan metode-metode untuk berinteraksi dengan endpoint dari The Movie Database (TMDb) API.

- **Baris 5: package com.example.modul5.data.remote** Mendeklarasikan keanggotaan berkas dalam paket data.remote.
- **Baris 7-9: import ...** Mengimpor kelas MovieApiResponse sebagai tipe data kembalian dan anotasi GET serta Query dari pustaka Retrofit.
- **Baris 11: interface ApiService** Mendeklarasikan ApiService sebagai sebuah *interface*. Retrofit akan menggunakan *interface* ini untuk menghasilkan implementasi kode pemanggil jaringan.
- **Baris 14: @GET("movie/now_playing")** Anotasi Retrofit yang menandakan bahwa metode getNowPlayingMovies akan melakukan *request* HTTP GET ke *endpoint* movie/now_playing relatif terhadap URL dasar.
- **Baris 15: suspend fun getNowPlayingMovies(...)** Mendefinisikan fungsi getNowPlayingMovies sebagai suspend, yang berarti fungsi ini dapat dijalankan dalam sebuah *coroutine* tanpa memblokir *thread*.
- **Baris 16: @Query("api_key") apiKey: String = ...** Anotasi Retrofit yang akan menambahkan parameter kueri api_key ke dalam URL *request*. Nilai *default* untuk kunci API juga ditetapkan di sini.
- **Baris 17:): MovieApiResponse** Menetapkan bahwa fungsi ini akan mengembalikan sebuah objek MovieApiResponse yang telah diproses dari respons JSON.

- Berkas: RetrofitClient.kt (Konfigurasi Klien HTTP)

Berkas ini bertanggung jawab untuk membuat dan mengonfigurasi satu instance (singleton) dari Retrofit.

- **Baris 11: object RetrofitClient** Mendeklarasikan RetrofitClient sebagai sebuah object. Ini adalah cara Kotlin untuk mengimplementasikan pola desain *singleton*, memastikan hanya ada satu *instance* dari klien Retrofit di seluruh aplikasi.
- **Baris 13: private const val BASE_URL = ...** Mendefinisikan URL dasar dari API sebagai sebuah konstanta privat.
- **Baris 17: private val json = Json { ignoreUnknownKeys = true }** Menginisialisasi sebuah *instance* Json dari pustaka KotlinX Serialization. Konfigurasi ignoreUnknownKeys = true mencegah aplikasi mengalami *crash* jika API mengembalikan atribut JSON yang tidak didefinisikan dalam data class.
- **Baris 20: private val retrofit: Retrofit by lazy { ... }** Mendeklarasikan *instance* Retrofit menggunakan delegasi by lazy. Ini berarti objek Retrofit hanya akan dibuat pada saat pertama kali diakses, bukan saat aplikasi dimulai, sehingga meningkatkan efisiensi.

- **Baris 22: .baseUrl(BASE_URL)** Mengatur URL dasar untuk semua *request* yang dibuat oleh *instance* Retrofit ini.
- **Baris 23: .addConverterFactory(...)** Menambahkan sebuah ConverterFactory. Dalam kasus ini, kotlinx.serialization digunakan untuk secara otomatis mengonversi respons JSON dari API menjadi objek-objek Kotlin.
- **Baris 24: .build()** Membangun *instance* Retrofit dengan konfigurasi yang telah ditetapkan.
- **Baris 27: val instance: ApiService by lazy { ... }** Mengekspos *instance* dari ApiService. Sama seperti Retrofit, ini menggunakan lazy delegation.
- **Baris 28: retrofit.create(ApiService::class.java)** Membuat implementasi konkret dari *interface* ApiService menggunakan *instance* Retrofit yang telah dikonfigurasi.

1.3. Sumber Data Lokal (data/local)

- Berkas: MovieDao.kt (Objek Akses Data Film)

Interface ini mendefinisikan semua operasi database (CRUD - Create, Read, Update, Delete) untuk entitas Movie.

- **Baris 11: @Dao** Anotasi Room yang mengidentifikasi *interface* ini sebagai *Data Access Object*.
- **Baris 12: interface MovieDao** Deklarasi *interface*.
- **Baris 13: @Insert(onConflict = OnConflictStrategy.REPLACE)** Anotasi untuk operasi penyisipan. Strategi konflik REPLACE berarti jika film dengan id yang sama sudah ada, entri lama akan digantikan oleh yang baru.
- **Baris 14: suspend fun insertMovies(movies: List<Movie>)** Fungsi untuk menyisipkan daftar film ke dalam database.
- **Baris 16: @Query("SELECT * FROM movies")** Anotasi yang berisi kueri SQL untuk mengambil semua kolom dari tabel movies.
- **Baris 17: fun getAllMovies(): Flow<List<Movie>>** Fungsi yang mengembalikan data sebagai Flow. Ini memungkinkan UI untuk bereaksi secara otomatis terhadap perubahan data di database.
- **Baris 19: @Query("SELECT * FROM movies WHERE isBookmarked = 1")** Kueri SQL untuk mengambil hanya film-film yang kolom isBookmarked-nya bernilai true (atau 1 dalam terminologi SQL).
- **Baris 20: fun getBookmarkedMovies(): Flow<List<Movie>>** Fungsi yang mengembalikan daftar film yang di-*bookmark* sebagai Flow.
- **Baris 22: @Update** Anotasi untuk operasi pembaruan data pada entitas yang ada.
- **Baris 23: suspend fun updateMovie(movie: Movie)** Fungsi untuk memperbarui satu entitas Movie di dalam database.

- Berkas: AppDatabase.kt (Definisi Database Aplikasi)

Kelas abstrak ini merepresentasikan database Room aplikasi dan merupakan titik akses utama ke data yang disimpan secara lokal.

- **Baris 10: @Database(entities = [Movie::class], version = 1, exportSchema = false)** Anotasi utama untuk mengonfigurasi database. entities mendaftarkan semua kelas entitas (tabel). version digunakan untuk manajemen migrasi. exportSchema dinonaktifkan untuk proyek sederhana ini.
- **Baris 11: abstract class AppDatabase : RoomDatabase()** Kelas ini harus abstrak dan mewarisi RoomDatabase.
- **Baris 13: abstract fun movieDao(): MovieDao** Sebuah metode abstrak yang akan diimplementasikan oleh Room untuk menyediakan akses ke MovieDao.
- **Baris 15: companion object { ... }** Blok ini digunakan untuk mendefinisikan metode dan properti statis, khususnya untuk mengimplementasikan pola *singleton*.
- **Baris 17: @Volatile private var INSTANCE: AppDatabase? = null** Mendeklarasikan variabel INSTANCE yang akan menampung satu-satunya *instance*

dari AppDatabase. Anotasi @Volatile memastikan bahwa perubahan pada variabel ini segera terlihat oleh semua *thread* lain.

- **Baris 19: fun getDatabase(context: Context): AppDatabase** Metode statis untuk mendapatkan *instance* database.
- **Baris 21: return INSTANCE ?: synchronized(this) { ... }** Menggunakan *Elvis operator* (?:) untuk memeriksa apakah INSTANCE sudah ada. Jika belum (null), blok synchronized akan dieksekusi untuk membuat *instance* baru secara aman dari *thread* (*thread-safe*).
- **Baris 22: val instance = Room.databaseBuilder(...)** Memulai proses pembangunan database menggunakan Room.databaseBuilder.
- **Baris 26: .build()** Menyelesaikan pembuatan *instance* database.

1.4. Repositori (data/repository)

- Berkas: MovieRepository.kt (Manajer Sumber Data)

Berkas ini berfungsi sebagai perantara yang mengelola operasi data antara ViewModel dengan sumber data, baik dari jaringan (API) maupun dari database lokal (Room).

- **Baris 7: class MovieRepository(...)** Mendeklarasikan kelas MovieRepository. Konstruktor kelas ini menerima apiService dan movieDao sebagai dependensi, menerapkan prinsip *Dependency Injection*.
- **Baris 8: private val apiService: ApiService** Deklarasi dependensi untuk ApiService, yang digunakan untuk melakukan panggilan ke API.
- **Baris 9: private val movieDao: MovieDao** Deklarasi dependensi untuk MovieDao, yang digunakan untuk berinteraksi dengan database lokal.
- **Baris 12: val allMovies = movieDao.getAllMovies()** Mengekspos sebuah Flow dari MovieDao yang berisi daftar semua film. ViewModel akan mengobservasi Flow ini untuk mendapatkan pembaruan data secara *real-time*.
- **Baris 14: val bookmarkedMovies = movieDao.getBookmarkedMovies()** Mengekspos Flow yang berisi daftar film yang telah ditandai (*bookmarked*).
- **Baris 17: suspend fun updateMovie(movie: Movie)** Mendefinisikan fungsi suspend untuk memperbarui entitas film di database. Ini biasanya dipanggil ketika status isBookmarked diubah.
- **Baris 18: movieDao.updateMovie(movie)** Memanggil fungsi updateMovie dari MovieDao untuk mengeksekusi operasi pembaruan di database.
- **Baris 21: suspend fun refreshMovies()** Mendefinisikan fungsi untuk mengambil data film terbaru dari API.
- **Baris 22: try { ... } catch (e: Exception) { ... }** Menggunakan blok try-catch untuk menangani kemungkinan Exception selama proses pemanggilan jaringan, misalnya ketika tidak ada koneksi internet.
- **Baris 24: val response = apiService.getNowPlayingMovies()** Memanggil fungsi di ApiService untuk mendapatkan daftar film yang sedang tayang dari server.
- **Baris 26: movieDao.insertMovies(response.results)** Setelah berhasil mendapatkan data dari API, data tersebut (response.results) dimasukkan ke dalam database lokal melalui MovieDao.

2. Lapisan ViewModel

Lapisan ini menghubungkan lapisan data dengan antarmuka pengguna, menangani logika bisnis dan manajemen status UI.

- Berkas: MovieViewModel.kt (Logika UI Utama)

Kelas ini bertanggung jawab untuk menyimpan dan mengelola data yang terkait dengan UI serta menangani logika bisnis.

- **Baris 11: class MovieViewModel(private val repository: MovieRepository) : ViewModel()** Mendeklarasikan kelas MovieViewModel yang mewarisi kelas

ViewModel dari Android Jetpack. Kelas ini menerima MovieRepository sebagai dependensinya.

- **Baris 14: val movies: Flow<List<Movie>> = repository.allMovies** Mendeklarasikan properti publik movies yang mendapatkan datanya dari repository.allMovies. UI akan mengobservasi Flow ini.
- **Baris 16: val bookmarkedMovies: Flow<List<Movie>> = repository.bookmarkedMovies** Mendeklarasikan properti publik bookmarkedMovies yang mendapatkan datanya dari repository.bookmarkedMovies.
- **Baris 18: init { ... }** Blok init dieksekusi secara otomatis ketika sebuah instance MovieViewModel dibuat untuk pertama kalinya.
- **Baris 21: viewModelScope.launch { ... }** Meluncurkan sebuah *coroutine* baru dalam viewModelScope. *Coroutine* yang diluncurkan dalam *scope* ini secara otomatis akan dibatalkan jika ViewModel dihancurkan, sehingga mencegah kebocoran memori (*memory leak*).
- **Baris 22: repository.refreshMovies()** Memanggil fungsi refreshMovies dari repository untuk memastikan data termuat dari API saat aplikasi dimulai.
- **Baris 26: fun toggleBookmark(movie: Movie)** Mendefinisikan fungsi publik yang dipanggil oleh UI ketika pengguna menekan tombol *bookmark*.
- **Baris 27: viewModelScope.launch { ... }** Meluncurkan *coroutine* untuk menjalankan operasi database di *background thread*.
- **Baris 29: val updatedMovie = movie.copy(isBookmarked = !movie.isBookmarked)** Membuat salinan objek movie dengan nilai isBookmarked yang dibalik (dari true menjadi false atau sebaliknya). Ini adalah praktik yang baik untuk menjaga *immutability*.
- **Baris 31: repository.updateMovie(updatedMovie)** Memanggil repository untuk memperbarui film dengan data yang telah diubah di dalam database.

- Berkas: SharedViewModel.kt (ViewModel untuk Data Bersama)

ViewModel ini dirancang untuk memfasilitasi komunikasi dan berbagi data antar Composable yang tidak memiliki hubungan induk-anak secara langsung.

- **Baris 11: class SharedViewModel : ViewModel()** Deklarasi kelas SharedViewModel yang mewarisi ViewModel.
- **Baris 14: var selectedMovie by mutableStateOf<Movie?>(null)** Mendeklarasikan properti selectedMovie menggunakan mutableStateOf. Ini berarti setiap perubahan pada nilai selectedMovie akan secara otomatis memicu rekomposisi pada *Composable* mana pun yang membaca nilai ini. Nilai awalnya adalah null.
- **Baris 15: private set** Menjadikan *setter* dari properti selectedMovie privat. Ini berarti hanya SharedViewModel itu sendiri yang dapat mengubah nilainya, sementara kelas lain hanya dapat membacanya. Ini adalah praktik enkapsulasi yang baik.
- **Baris 18: fun selectMovie(movie: Movie)** Sebuah fungsi publik yang dipanggil untuk memperbarui nilai selectedMovie.

- Berkas: MovieViewModelFactory.kt (Pabrik ViewModel)

Kelas ini bertanggung jawab untuk membuat instance dari MovieViewModel, khususnya untuk menyediakan dependensi MovieRepository yang dibutuhkan oleh konstruktornya.

- **Baris 8: class MovieViewModelFactory(...) : ViewModelProvider.Factory** Mendeklarasikan kelas MovieViewModelFactory yang mengimplementasikan *interface* ViewModelProvider.Factory.
- **Baris 11: override fun <T : ViewModel> create(modelClass: Class<T>): T** Meng-override metode create, yang merupakan metode inti dari *factory*. Metode ini akan dipanggil oleh sistem ketika sebuah ViewModel perlu dibuat.

- **Baris 13: if (modelClass.isAssignableFrom(MovieViewModel::class.java))** Memeriksa apakah kelas ViewModel yang diminta adalah MovieViewModel atau turunannya.
- **Baris 15: return MovieViewModel(repository) as T** Jika pemeriksaan berhasil, sebuah *instance* baru MovieViewModel dibuat dengan meneruskan repository sebagai argumen, kemudian di-*cast* ke tipe generik T.
- **Baris 18: throw IllegalArgumentException("Unknown ViewModel class")** Jika kelas yang diminta bukan MovieViewModel, sebuah pengecualian dilemparkan untuk menandakan kesalahan konfigurasi.

3. Lapisan Antarmuka Pengguna (UI Layer)

Lapisan ini bertanggung jawab untuk menampilkan data di layar dan menangani interaksi pengguna.

3.1. Navigasi (navigation)

- Berkas: Screen.kt (Definisi Rute Navigasi)

Berkas ini mendefinisikan semua kemungkinan tujuan navigasi dalam aplikasi secara terpusat.

- **Baris 7: sealed class Screen(val route: String, val title: String)** Menggunakan sealed class untuk membatasi hierarki kelas. Ini memastikan bahwa semua tujuan navigasi yang mungkin harus didefinisikan di dalam berkas ini, mencegah kesalahan pengetikan rute di tempat lain.
- **Baris 9: object MovieList : Screen("movie_list", "Movies")** Mendefinisikan layar daftar film sebagai sebuah object tunggal. route adalah pengenal unik untuk navigasi, dan title dapat digunakan untuk UI (misalnya, di *app bar*).
- **Baris 11 & 13: object Detail, object Bookmark** Mendefinisikan layar Detail dan Bookmark dengan cara yang sama.

- Berkas: AppNavigation.kt (Grafik Navigasi Utama)

Composable ini mengatur NavHost tingkat atas.

- **Baris 12: val navController = rememberNavController()** Membuat dan mengingat sebuah NavController yang akan bertahan selama *Composable* berada dalam komposisi.
- **Baris 15: NavHost(navController = navController, startDestination = Screen.MovieList.route)** Menginisialisasi NavHost yang merupakan kontainer untuk semua tujuan navigasi. startDestination menetapkan layar mana yang akan ditampilkan pertama kali.
- **Baris 17: composable(Screen.MovieList.route) { ... }** Mendefinisikan sebuah tujuan dalam grafik navigasi. Blok *lambda* ini akan dieksekusi ketika navController bernavigasi ke rute Screen.MovieList.route.
- **Baris 20: MainScreen(...)** Menempatkan MainScreen sebagai konten untuk rute awal, dengan meneruskan semua ViewModel dan NavController yang diperlukan.

3.2. Komponen dan Layar UI (ui/theme)

- Berkas: MovieListItem.kt (Item dalam Daftar)

Composable ini mendefinisikan tampilan untuk satu baris item film.

- **Baris 19: @Composable fun MovieListItem(...)** Mendeklarasikan fungsi *Composable* MovieListItem yang menerima tiga parameter: objek movie untuk ditampilkan, dan dua fungsi *lambda* (onItemClick, onBookmarkClick) sebagai *event handler*.
- **Baris 25: Card(...)** Menggunakan Card Composable untuk membungkus item, memberikan elevasi dan batas yang jelas.
- **Baris 29: .clickable { onItemClick(movie) }** Menambahkan *modifier* clickable ke Card, sehingga seluruh area kartu dapat merespons klik dan akan memicu fungsi onItemClick.

- **Baris 31: Row(...)** Menyusun elemen-elemen di dalamnya (gambar dan teks) secara horizontal.
- **Baris 33: AsyncImage(...)** Menggunakan *Composable* AsyncImage dari pustaka Coil untuk memuat dan menampilkan gambar poster dari URL secara asinkron.
- **Baris 40: Column(...)** Menyusun elemen-elemen di dalamnya (judul dan sinopsis) secara vertikal.
- **Baris 43: .weight(1f)** Memberikan bobot pada Column agar mengisi sisa ruang horizontal yang tersedia di dalam Row, mendorong ikon *bookmark* ke tepi kanan.
- **Baris 45-51: Text(...)** Menampilkan judul dan sinopsis film, dengan jumlah baris sinopsis dibatasi maksimal 3.
- **Baris 53: IconButton(onClick = { onBookmarkClick(movie) })** Membuat sebuah tombol ikon yang akan memicu fungsi onBookmarkClick ketika ditekan.
- **Baris 56: imageVector = if (movie.isBookmarked) ... else ...** Secara kondisional memilih ikon yang akan ditampilkan: Icons.Filled.Bookmark (terisi) jika isBookmarked bernilai true, dan Icons.Outlined.BookmarkBorder (garis tepi) jika false.

- Berkas: MovieListScreen.kt (Layar Daftar Film)

Composable ini bertanggung jawab untuk menampilkan daftar film utama.

- **Baris 16: @Composable fun MovieListScreen(...)** Deklarasi fungsi *Composable*.
- **Baris 22: val movies by movieViewModel.movies.collectAsState(...)** Mengonversi Flow dari ViewModel menjadi State. UI akan secara otomatis diperbarui setiap kali Flow memancarkan daftar baru.
- **Baris 25: LazyColumn { ... }** Menggunakan LazyColumn yang efisien untuk menampilkan daftar vertikal, karena hanya me-render item yang terlihat di layar.
- **Baris 27: items(movies, key = { it.id }) { movie -> ... }** Fungsi items dari LazyColumn. Parameter key yang unik untuk setiap item sangat penting untuk optimisasi performa.
- **Baris 32: sharedViewModel.selectMovie(selectedMovie)** Menyimpan film yang dipilih ke dalam SharedViewModel saat item diklik.
- **Baris 34: navController.navigate(Screen.Detail.route)** Menginstruksikan NavController untuk bernavigasi ke rute layar detail.
- **Baris 38: movieViewModel.toggleBookmark(movieToBookmark)** Memanggil fungsi di ViewModel untuk mengubah status *bookmark*.

- Berkas: BookmarkScreen.kt (Layar Daftar Bookmark)

Composable ini khusus menampilkan film yang telah di-bookmark.

- **Baris 15: val bookmarkedMovies by movieViewModel.bookmarkedMovies.collectAsState(...)** Struktur kode sangat mirip dengan MovieListScreen, namun sumber datanya adalah bookmarkedMovies dari ViewModel.
- **Baris 18-33: LazyColumn { ... }** Struktur LazyColumn dan items identik dengan MovieListScreen, hanya saja beroperasi pada daftar bookmarkedMovies.

- Berkas: DetailScreen.kt (Layar Detail Film)

Composable ini menampilkan informasi terperinci dari satu film yang dipilih.

- **Baris 14: @Composable fun DetailScreen(sharedViewModel: SharedViewModel)** Menerima SharedViewModel untuk mendapatkan data film yang akan ditampilkan.
- **Baris 16: val selectedMovie = sharedViewModel.selectedMovie** Membaca properti selectedMovie dari SharedViewModel.
- **Baris 19: if (selectedMovie != null) { ... }** Melakukan pemeriksaan *null-safety* untuk memastikan data film ada sebelum mencoba menampilkannya.
- **Baris 21-30: AsyncImage, Spacer, Text** Menyusun dan menampilkan elemen-elemen UI seperti gambar poster, judul, dan sinopsis dari film yang dipilih.

- Berkas: MainScreen.kt (Kerangka Utama Aplikasi)

Composable ini membangun struktur visual utama aplikasi menggunakan Scaffold.

- **Baris 29: val internalNavController = rememberNavController()** Membuat sebuah NavController lokal yang khusus digunakan untuk navigasi antar item di NavigationBar (Home dan Bookmark).
- **Baris 34: Scaffold(...)** Menggunakan Scaffold untuk mengimplementasikan tata letak dasar Material Design, dengan slot untuk topBar dan bottomBar.
- **Baris 35: topBar = { TopAppBar(...) }** Mendefinisikan TopAppBar (bilah atas) aplikasi.
- **Baris 43: bottomBar = { NavigationBar { ... } }** Mendefinisikan NavigationBar (bilah navigasi bawah).
- **Baris 50: items.forEach { screen -> ... }** Melakukan iterasi untuk setiap screen di dalam daftar items untuk membuat NavigationBarItem.
- **Baris 62: onClick = { ... }** Menentukan aksi ketika item navigasi diklik, yaitu memanggil internalNavController.navigate(screen.route) dengan konfigurasi *back stack* yang benar.
- **Baris 76: NavHost(...)** Menggunakan NavHost untuk menjadi kontainer bagi layar-layar yang dinavigasi oleh internalNavController.
- **Baris 79: modifier = Modifier.padding(innerPadding)** Menerapkan innerPadding yang disediakan oleh Scaffold agar konten tidak tertutup oleh bilah atas dan bawah.
- **Baris 81-89: composable(...)** Mendefinisikan grafik navigasi internal yang memetakan setiap rute ke *Composable* layarnya.

4. Titik Masuk Aplikasi (MainActivity.kt)

- Berkas: MainActivity.kt (Aktivitas Utama)

Ini adalah satu-satunya Activity dan merupakan pintu masuk aplikasi.

- **Baris 14: class MainActivity : ComponentActivity()** Deklarasi kelas MainActivity.
- **Baris 17-25: val database by lazy { ... }, val repository by lazy { ... }, val movieViewModel ... by viewModels { ... }, val sharedViewModel by viewModels()** Menginisialisasi semua dependensi utama aplikasi. by lazy digunakan untuk database dan repository. by viewModels adalah delegasi properti KTX yang menyediakan ViewModel yang terikat pada siklus hidup Activity.
- **Baris 21: MovieViewModelFactory(repository)** Menggunakan MovieViewModelFactory untuk menyediakan repository ke MovieViewModel.
- **Baris 27: override fun onCreate(savedInstanceState: Bundle?)** Metode siklus hidup Activity yang dipanggil saat Activity pertama kali dibuat.
- **Baris 29: setContent { ... }** Fungsi ekstensi yang mendefinisikan konten UI Activity menggunakan Jetpack Compose.
- **Baris 31: Modul5Theme { ... }** Menerapkan tema kustom aplikasi ke seluruh konten di dalamnya.
- **Baris 33: AppNavigation(...)** Memanggil *Composable* navigasi tingkat atas untuk membangun UI dan alur navigasi aplikasi.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/Adiyaus/Praktikum-PemrogramanMobile.git