LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Avantio Fierza Patria NIM. 2310817310001

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 : Avantio Fierza Patria
NIM : 2310817310001

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Muhammad Raka Azwar NIM. 2210817210012	Andreyan Rizky Baskara, S.Kom., M.Kom. NIP. 19930703 201903 01 011

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Soal 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 menggunakan Jetpack Compose

MainActivity.kt

```
1  package com.example.scrollablelist
2
3  import android.os.Bundle
4  import androidx.activity.ComponentActivity
5  import androidx.activity.compose.setContent
6  import com.example.scrollablelist.navigation.NavGraph
7  import com.example.scrollablelist.ui.theme.ScrollableListTheme
8
9  class MainActivity : ComponentActivity() {
10
11     private  val  apiKey  =  "9fad0dc9a0338ecf00596875e4cf5645"  //
12 Masukkan API Key di sini
13
14     override fun onCreate(savedInstanceState: Bundle?) {
15         super.onCreate(savedInstanceState)
16         setContent {
17             ScrollableListTheme {
18                 NavGraph()
19             }
20         }
21     }
22 }
```

Tabel 1. 1 Source Code MainActivity.kt Jetpack Compose

ApiService.kt

```
1  package com.example.scrollablelist.data.remote
```

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

Tabel 1. 2 Source Code ApiService.kt Jetpack Compose

DetailScreen.kt

```
1   package com.example.scrollablelist.screens
2
3   import androidx.activity.compose.BackHandler
4   import androidx.compose.foundation.Image
5   import androidx.compose.foundation.layout.*
6   import androidx.compose.foundation.shape.RoundedCornerShape
7   import androidx.compose.material3.*
8   import androidx.compose.runtime.Composable
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.text.font.FontWeight
13  import androidx.compose.ui.unit.dp
14  import androidx.compose.ui.unit.sp
15  import coil.compose.rememberAsyncImagePainter
16  import androidx.compose.material.icons.filled.ArrowBack
17
18  @Composable
19  fun DetailScreen(
20      title: String,
21      imageUrl: String,
22      fullDescription: String,
23      onBack: () -> Unit
24  ) {
25      val imageBaseUrl = "https://image.tmdb.org/t/p/w500"
26      val fullImageUrl = imageBaseUrl + imageUrl
27
28      Scaffold(
29          topBar = {
30              SmallTopAppBar(
31                  title = { Text(text = "Detail Film") },
32                  onBack = onBack
33              )
34          }
35      ) { padding ->
36          Column(
37              modifier = Modifier
38                  .fillMaxSize()
39                  .padding(padding)
40                  .padding(16.dp)
```

```
        ) {
            Image(
                painter                                          =
rememberAsyncImagePainter(fullImageUrl),
                contentDescription = title,
                modifier = Modifier
                    .fillMaxWidth()
                    .height(300.dp)
                    .clip(RoundedCornerShape(16.dp))
            )

            Spacer(modifier = Modifier.height(16.dp))

            Text(
                text = title,
                style = MaterialTheme.typography.headlineSmall,
                fontWeight = FontWeight.Bold,
                fontSize = 24.sp,
                color = Color.Black
            )

            Spacer(modifier = Modifier.height(12.dp))

            Text(
                text = fullDescription,
                style = MaterialTheme.typography.bodyLarge,
                fontSize = 18.sp,
                color = Color.DarkGray
            )
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SmallTopAppBar(
    title: @Composable () -> Unit,
    onBack: () -> Unit
) {
    TopAppBar(
        title = title,
        navigationIcon = {
            IconButton(onClick = onBack) {
                Icon(
                    imageVector                                  =
androidx.compose.material.icons.Icons.Default.ArrowBack,
                    contentDescription = "Back"
                )
            }
        }
    )
}


```

```
95
96
```

Tabel 1. 3 Source Code DetailScreen.kt Jetpack Compose

Film.kt

```
1  package com.example.scrollablelist.model
2
3  data class Film(
4      val id: Int,
5      val title: String,
6      val overview: String,
7      val posterPath: String?,
8      val backdropPath: String?,
9      val releaseDate: String?,
10     val homepage: String? = null
11 )
```

Tabel 1. 4 Source Code Film.kt Jetpack Compose

FilmRepository.kt

```
1  package com.example.scrollablelist.data
2
3  import com.example.scrollablelist.model.Film
4  import com.example.scrollablelist.network.TmdbApi
5
6  class FilmRepository(
7      private val api: TmdbApi,
8      private val apiKey: String
9  ) {
10     suspend fun getPopularFilms(): List<Film> {
11         val response = api.getPopularMovies(apiKey)
12
13         return response.results.map { networkFilm ->
14             Film(
15                 id = networkFilm.id,
16                 title = networkFilm.title,
17                 overview = networkFilm.overview,
18                 posterPath = networkFilm.posterPath,
19                 backdropPath = networkFilm.backdropPath,
20                 releaseDate = networkFilm.releaseDate,
21                 homepage = null
22             )
23         }
24     }
25 }
```

Tabel 1. 5 Source Code FilmRepository.kt Jetpack Compose

FilmViewModel.kt

```
1  package com.example.scrollablelist.ui
2
3  import androidx.lifecycle.ViewModel
4  import androidx.lifecycle.viewModelScope
5  import com.example.scrollablelist.data.FilmRepository
6  import com.example.scrollablelist.model.Film
```

```
7  import kotlinx.coroutines.flow.MutableStateFlow
8  import kotlinx.coroutines.flow.StateFlow
9  import kotlinx.coroutines.flow.asStateFlow
10 import kotlinx.coroutines.launch
11 import timber.log.Timber
26
27 class FilmViewModel(
28     private val repository: FilmRepository
29 ) : ViewModel() {
30
31     private             val             _filmList             =
32 MutableStateFlow<List<Film>>(emptyList())
33     val         filmList:         StateFlow<List<Film>>         =
34 _filmList.asStateFlow()
35
36     private val _event = MutableStateFlow<Event?>(null)
37     val event: StateFlow<Event?> = _event.asStateFlow()
38
39     init {
40         loadFilmsFromApi()
41     }
42
43     private fun loadFilmsFromApi() {
44         viewModelScope.launch {
45             try {
46                 val films = repository.getPopularFilms()
47                 _filmList.value = films as List<Film>
48             } catch (e: Exception) {
49                 Timber.e("Gagal    memuat    film    dari    API:
50 ${e.message}")
51             }
52         }
53     }
54
55     fun onItemClicked(film: Film) {
56         _event.value = Event.NavigateToDetail(film)
57     }
58
59     fun onWebButtonClicked(url: String) {
60         _event.value = Event.OpenWebUrl(url)
61     }
62
63     fun clearEvent() {
64         _event.value = null
65     }
66
67     sealed class Event {
68         data class NavigateToDetail(val film: Film) : Event()
69         data class OpenWebUrl(val url: String) : Event()
70     }
71 }
```

Tabel 1. 6 Source Code FilmViewModel.kt Jetpack Compose

FilmViewModelFactory.kt

```
1  package com.example.scrollablelist.ui
2
3  import androidx.lifecycle.ViewModel
4  import androidx.lifecycle.ViewModelProvider
5  import com.example.scrollablelist.data.FilmRepository
6
7  class FilmViewModelFactory(
8      private val repository: FilmRepository
9  ) : ViewModelProvider.Factory {
10
11     override fun <T : ViewModel> create(modelClass: Class<T>):
12 T {
13
14 if(modelClass.isAssignableFrom(FilmViewModel::class.java)) {
15             @Suppress("UNCHECKED_CAST")
16             return FilmViewModel(repository) as T
17         }
18         throw    IllegalArgumentException("Unknown    ViewModel
19 class")
20     }
21 }
```

Tabel 1. 7 Source Code FlmViewModelFactory.kt Jetpack Compose

HomeScreen.kt

```
1   package com.example.scrollablelist.screens
2
3   import android.content.Intent
4   import android.net.Uri
5   import androidx.compose.foundation.Image
6   import androidx.compose.foundation.layout.*
7   import androidx.compose.foundation.lazy.LazyColumn
8   import androidx.compose.foundation.lazy.items
9   import androidx.compose.foundation.shape.RoundedCornerShape
10  import androidx.compose.material3.*
11  import androidx.compose.runtime.*
12  import androidx.compose.ui.Modifier
13  import androidx.compose.ui.draw.clip
14  import androidx.compose.ui.graphics.Color
15  import androidx.compose.ui.platform.LocalContext
16  import androidx.compose.ui.text.font.FontWeight
17  import androidx.compose.ui.unit.dp
18  import androidx.compose.ui.unit.sp
19  import androidx.lifecycle.viewmodel.compose.viewModel
20  import androidx.navigation.NavController
21  import coil.compose.rememberAsyncImagePainter
22  import com.example.scrollablelist.data.FilmRepository
23  import com.example.scrollablelist.network.RetrofitInstance
24  import com.example.scrollablelist.ui.FilmViewModel
25  import com.example.scrollablelist.ui.FilmViewModelFactory
26
27  @Composable
28  fun HomeScreen(navController: NavController) {
```

```
    val context = LocalContext.current

    val repository = remember {
        FilmRepository(
            api = RetrofitInstance.api,
            apiKey = "9fad0dc9a0338ecf00596875e4cf5645"
        )
    }
    val viewModel: FilmViewModel = viewModel(factory =
FilmViewModelFactory(repository))

    val films by viewModel.filmList.collectAsState()
    val event by viewModel.event.collectAsState()

    val imageBaseUrl = "https://image.tmdb.org/t/p/w500"

    LaunchedEffect(event) {
        when (event) {
            is FilmViewModel.Event.NavigateToDetail -> {
                val film = (event as
FilmViewModel.Event.NavigateToDetail).film
                navController.navigate(

"detail/${Uri.encode(film.title)}/${Uri.encode(film.posterPath
?: "")}/${Uri.encode(film.overview)}"
                )
                viewModel.clearEvent()
            }
            is FilmViewModel.Event.OpenWebUrl -> {
                val url = (event as
FilmViewModel.Event.OpenWebUrl).url
                val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(url))
                navController.context.startActivity(intent)
                viewModel.clearEvent()
            }
            null -> Unit
        }
    }

    LazyColumn(
        contentPadding = PaddingValues(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        items(films) { film ->
            Card(
                shape = RoundedCornerShape(16.dp),
                colors =
CardDefaults.cardColors(containerColor = Color(0xFF2C2C2C)),
                modifier = Modifier.fillMaxWidth()
            ) {
                Row(
                    modifier = Modifier
                        .padding(16.dp)
```

```
                        .fillMaxWidth()
                ) {
                    Image(
                        painter                                 =
rememberAsyncImagePainter(imageBaseUrl  +  (film.posterPath  ?:
""")),
                        contentDescription = film.title,
                        modifier = Modifier
                            .size(100.dp)
                            .clip(RoundedCornerShape(16.dp))
                    )

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

                    Column(
                        modifier = Modifier.weight(1f)
                    ) {
                        Text(
                            text = film.title,
                            style                               =
MaterialTheme.typography.titleMedium,
                            color = Color.White,
                            fontWeight = FontWeight.Bold,
                            fontSize = 18.sp
                        )
                        Spacer(modifier                         =
Modifier.height(4.dp))
                        Text(
                            text = film.overview,
                            style                               =
MaterialTheme.typography.bodyMedium,
                            color = Color.Gray,
                            maxLines = 3,
                            overflow                            =
androidx.compose.ui.text.style.TextOverflow.Ellipsis
                        )

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

                        Row {
                            Button(
                                onClick          =          {
viewModel.onItemClicked(film) },
                                colors                          =
ButtonDefaults.buttonColors(
                                    containerColor              =
Color(0xFFB3C7F9),
                                    contentColor = Color.Black
                                ),
                                shape                           =
RoundedCornerShape(50),
                                modifier                        =
Modifier.height(40.dp)
```

```
                                ) {
                                    Text(text = "Detail")
                                }
                            }
                        }
                    }
                }
            }
        }
```

Tabel 1. 8 Source Code HomeScreen.kt Jetpack Compose

MainScreen.kt

```
package com.example.scrollablelist.ui

import androidx.compose.foundation.Image
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.unit.dp
import coil.compose.rememberAsyncImagePainter
import com.example.scrollablelist.model.Film

@Composable
fun MainScreen(viewModel: FilmViewModel) {
    // Ganti films ke filmList, dan pastikan collectAsState
dengan tipe eksplisit
    val films = viewModel.filmList.collectAsState(initial =
emptyList())

    Scaffold(
        topBar = {
            SmallTopAppBar(title = { Text("Popular Movies") })
        }
    ) { padding ->
        LazyColumn(
            modifier = Modifier
                .padding(padding)
                .fillMaxSize()
        ) {
            items(films.value) { film ->
                FilmListItem(film)
            }
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
```

```
@Composable
fun SmallTopAppBar(title: @Composable () -> Unit) {
    TopAppBar(title = title)
}

@Composable
fun FilmListItem(film: Film) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .clickable { /* Navigate to details if needed */ }
            .padding(8.dp)
    ) {
        Image(
            painter                                                =
rememberAsyncImagePainter("https://image.tmdb.org/t/p/w500${fil
m.posterPath}"),
            contentDescription = film.title,
            modifier = Modifier.size(80.dp),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.width(16.dp))
        Column(modifier = Modifier.fillMaxWidth()) {
            Text(text      =      film.title,      style      =
MaterialTheme.typography.titleMedium)
            Spacer(modifier = Modifier.height(4.dp))
            Text(
                text = film.overview,
                maxLines = 3,
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}
```

Tabel 1. 9 Source Code MainScreen.kt Jetpack Compose

Mappers.kt

```
package com.example.scrollablelist.model

fun NetworkFilm.toFilm(): Film {
    return Film(
        id = id,
        title = title,
        overview = overview,
        posterPath = posterPath,
        backdropPath = backdropPath,
        releaseDate = releaseDate,
        homepage = null
    )
}
```

Tabel 1. 10 Source Code Mappers.kt Jetpack Compose

MovieResponse.kt

```
1   package com.example.scrollablelist.model
2
3   fun NetworkFilm.toFilm(): Film {
4       return Film(
5           id = id,
6           title = title,
7           overview = overview,
8           posterPath = posterPath,
9           backdropPath = backdropPath,
10          releaseDate = releaseDate,
11          homepage = null
12      )
13  }
```

Tabel 1. 11 Source Code MovieResponse.kt Jetpack Compose

MyApplication.kt

```
1   package com.example.scrollablelist
2
3   import android.app.Application
4   import
5   com.example.scrollablelist.data.FilmRepository
6   import com.example.scrollablelist.network.TmdbApi
7   import
8   com.jakewharton.retrofit2.converter.kotlinx.serial
9   ization.asConverterFactory
10  import kotlinx.serialization.json.Json
11  import okhttp3.MediaType.Companion.toMediaType
12  import okhttp3.OkHttpClient
13  import retrofit2.Retrofit
14  import timber.log.Timber
15
16  class MyApplication : Application() {
17
18      lateinit var filmRepository: FilmRepository
19          private set
20
21      lateinit         var         preferenceManager:
22  PreferenceManager
23          private set
24
25      override fun onCreate() {
26          super.onCreate()
27
28          Timber.plant(Timber.DebugTree())
29
30          preferenceManager = PreferenceManager(this)
31
32          val              contentType              =
33  "application/json".toMediaType()
34          val json = Json { ignoreUnknownKeys = true
35  }
36
37          val              okHttpClient             =
38  OkHttpClient.Builder().build()
```

```
39
40          val retrofit = Retrofit.Builder()
41
42  .baseUrl("https://api.themoviedb.org/3/")
43              .client(okHttpClient)
44
45  .addConverterFactory(json.asConverterFactory(conte
46  ntType))
47              .build()
48
49          val                    api                    =
50  retrofit.create(TmdbApi::class.java)
51
52          val                  apiKey                   =
53  "9fad0dc9a0338ecf00596875e4cf5645"
54
55          filmRepository     =     FilmRepository(api,
56  apiKey)
57      }
58  }
```

Tabel 1. 12 Source Code MyApplication.kt Jetpack Compose

Navigation.kt

```
1   package com.example.scrollablelist.navigation
2
3   import androidx.compose.runtime.Composable
4   import androidx.navigation.NavHostController
5   import androidx.navigation.compose.NavHost
6   import androidx.navigation.compose.composable
7   import
8   androidx.navigation.compose.rememberNavControll
9   er
10  import
11  com.example.scrollablelist.screens.DetailScreen
12  import
13  com.example.scrollablelist.screens.HomeScreen
14
15  @Composable
16  fun NavGraph(navController: NavHostController =
17  rememberNavController()) {
18      NavHost(
19          navController = navController,
20          startDestination = "home"
21      ) {
22          composable("home") {
23              HomeScreen(navController)
24          }
25
26  composable("detail/{title}/{imageUrl}/{fullDesc
27  ription}") { backStackEntry ->
28              val             title             =
29  backStackEntry.arguments?.getString("title")  ?:
30  ""
31
```

```
            val            imageUrl            =
backStackEntry.arguments?.getString("imageUrl")
?: ""
            val        fullDescription        =
backStackEntry.arguments?.getString("fullDescri
ption") ?: ""

            DetailScreen(
                title = title,
                imageUrl = imageUrl,
                fullDescription               =
fullDescription,
                onBack           =           {
navController.popBackStack() }
            )
        }
    }
}

```

Tabel 1. 13 Source Code Navigation.kt Jetpack Compose

NetworkFilm.kt

```
package com.example.scrollablelist.navigation
package com.example.scrollablelist.model

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable
import
kotlinx.serialization.InternalSerializationApi
import kotlin.annotation.AnnotationTarget.CLASS

@OptIn(InternalSerializationApi::class)
@Serializable
data class NetworkFilm(
    val id: Int,
    val title: String,
    val overview: String,
    @SerialName("poster_path")  val  posterPath:
String?,
    @SerialName("backdrop_path")            val
backdropPath: String?,
    @SerialName("release_date") val releaseDate:
String?
)
```

Tabel 1. 14 Source Code NetworkFilm.kt Jetpack Compose

PreferenceManager.kt

```
package com.example.scrollablelist

import android.content.Context
import android.content.SharedPreferences
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableStateFlow
```

```
7
8   class PreferenceManager(context: Context) {
9
10      private val prefs: SharedPreferences =
11
12  context.getSharedPreferences("app_prefs",
13  Context.MODE_PRIVATE)
14
15      private       val       _isDarkMode       =
16  MutableStateFlow(prefs.getBoolean(KEY_DARK_MODE
17  , false))
18      val isDarkMode: Flow<Boolean> = _isDarkMode
19
20      fun setDarkMode(enabled: Boolean) {
21          prefs.edit().putBoolean(KEY_DARK_MODE,
22  enabled).apply()
23          _isDarkMode.value = enabled
24      }
25
26      companion object {
27          private   const   val   KEY_DARK_MODE   =
28  "key_dark_mode"
29      }
30  }
```

Tabel 1. 15 Source Code PreferenceManager.kt Jetpack Compose

Result.kt

```
1   package com.example.scrollablelist
2
3
4   sealed class Result<out T> {
5       object Loading : Result<Nothing>()
6       data   class   Success<T>(val   data:   T)   :
7   Result<T>()
8       data class Error(val exception: Throwable) :
9   Result<Nothing>()
10  }
```

Tabel 1. 16 Source Code Result.kt Jetpack Compose

RetrofitInstance.kt

```
1   package com.example.scrollablelist.network
2
3   import
4   com.jakewharton.retrofit2.converter.kotlinx.ser
5   ialization.asConverterFactory
6   import kotlinx.serialization.json.Json
7   import okhttp3.MediaType.Companion.toMediaType
8   import okhttp3.OkHttpClient
9   import retrofit2.Retrofit
10
11  object RetrofitInstance {
12
13      private val json = Json {
```

```
14          ignoreUnknownKeys = true // biar bisa
15  ignore field yg gak dipakai
16      }
17
18      private val client = OkHttpClient.Builder()
19          .build()
20
21      private val retrofit by lazy {
22          Retrofit.Builder()
23
24  .baseUrl("https://api.themoviedb.org/3/")     //
25  base URL TMDb API
26              .client(client)
27
28  .addConverterFactory(json.asConverterFactory("a
29  pplication/json".toMediaType()))
30              .build()
31      }
32
33      val api: TmdbApi by lazy {
34          retrofit.create(TmdbApi::class.java)
35      }
36  }
```

Tabel 1. 17 Source Code RetrofitInstance.kt Jetpack Compose

SharedPreferenceViewModel.kt

```
1   package com.example.scrollablelist
2
3   import androidx.lifecycle.ViewModel
4   import kotlinx.coroutines.flow.StateFlow
5   import kotlinx.coroutines.flow.SharingStarted
6   import kotlinx.coroutines.flow.stateIn
7   import kotlinx.coroutines.flow.map
8   import kotlinx.coroutines.CoroutineScope
9   import kotlinx.coroutines.Dispatchers
10
11  class   SharedPreferenceViewModel(private   val
12  preferenceManager:     PreferenceManager)     :
13  ViewModel() {
14
15      val   isDarkMode:   StateFlow<Boolean>   =
16  preferenceManager.isDarkMode
17          .stateIn(
18              CoroutineScope(Dispatchers.Main),
19              SharingStarted.Eagerly,
20              false
21          )
22
23      fun toggleDarkMode() {
24          val current = isDarkMode.value
25          preferenceManager.setDarkMode(!current)
26      }
27  }
```

Tabel 1. 18 Source Code SharedPreferenceViewModel.kt Jetpack Compose

SharedPreferenceViewModelFactory.kt

```
1   package com.example.scrollablelist
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5
6   class SharedPreferenceViewModelFactory(
7       private        val        preferenceManager:
8   PreferenceManager
9   ) : ViewModelProvider.Factory {
10
11      override     fun     <T     :     ViewModel>
12  create(modelClass: Class<T>): T {
13          if
14  (modelClass.isAssignableFrom(SharedPreferenceVi
15  ewModel::class.java)) {
16              @Suppress("UNCHECKED_CAST")
17              return
18  SharedPreferenceViewModel(preferenceManager)  as
19  T
20          }
21          throw  IllegalArgumentException("Unknown
22  ViewModel class")
23      }
24  }
```

Tabel 1. 19 Source Code SharedPreferenceViewModelFactory.kt Jetpack Compose

TMDBApi.kt

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

Tabel 1. 20 Source Code TMDBApi.kt Jetpack Compose

SettingsScreem.kt

```
1   package com.example.scrollablelist.screens
2
3   import androidx.compose.material3.Switch
4   import androidx.compose.material3.Text
5   import androidx.compose.foundation.layout.Row
6   import
7   androidx.compose.foundation.layout.padding
8   import androidx.compose.runtime.Composable
```

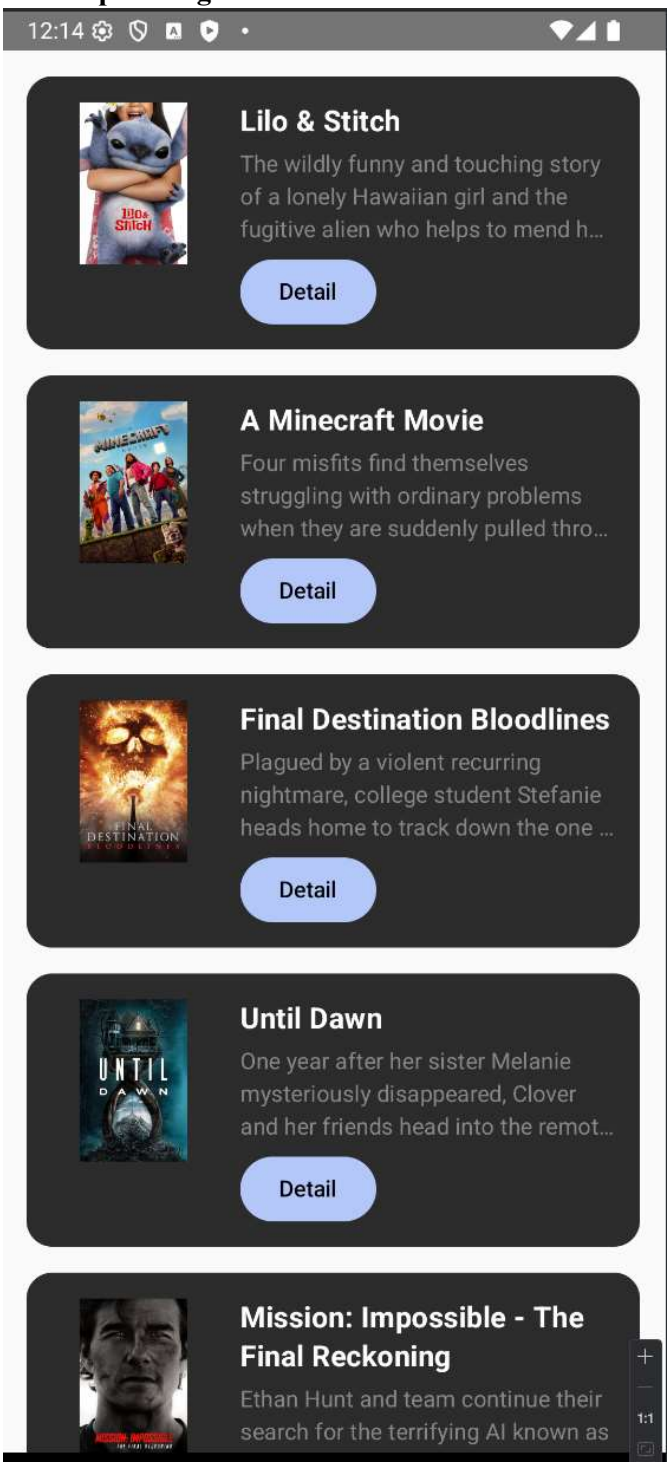
```
import androidx.compose.runtime.collectAsState
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import
com.example.scrollablelist.SharedPreferenceView
Model

@Composable
fun                    SettingsScreen(viewModel:
SharedPreferenceViewModel) {
    val               isDarkMode               =
viewModel.isDarkMode.collectAsState()

    Row(modifier = Modifier.padding(16.dp)) {
        Text(text = "Dark Mode")
        Switch(
            checked = isDarkMode.value,
            onCheckedChange         =         {
viewModel.toggleDarkMode() },
            modifier = Modifier.padding(start =
8.dp)
        )
    }
}
```

Tabel 1. 21 Source Code SettingScreen.kt Jetpack Compose

B. Output Program



Gambar 1. 1 Screenshot Tampilan List Menggunakan API

C. Pembahasan

MainActivity.kt Jetpack Compose:

- `MainActivity` adalah titik awal aplikasi.
- `setContent` digunakan untuk menampilkan UI berbasis **Jetpack Compose**.
- `ScrollableListTheme { NavGraph() }` memastikan bahwa seluruh UI mengikuti tema tertentu dan menggunakan sistem navigasi terpisah (`NavGraph`) untuk berpindah antar halaman (screen).
- `apiKey` disimpan secara lokal, sebaiknya diletakkan di `BuildConfig` atau `local.properties` untuk keamanan.

ApiService.kt

- Menggunakan **Retrofit** untuk mengambil data dari endpoint `movie/popular` di TMDB.
- Fungsi `suspend` artinya dapat dipanggil dalam coroutine (asynchronous).
- Menggunakan query `api_key` untuk otorisasi.
- Return type adalah `MovieResponse`, yaitu response model JSON dari TMDB.

Film.kt Jetpack Compose :

- `Film` adalah model yang digunakan dalam **lapisan domain** (digunakan oleh UI dan repository).
- Disusun agar bersih dari detail jaringan (network response), memudahkan testing dan pemeliharaan.
- Dipisahkan dari model response agar modular dan tidak terikat pada struktur response TMDB.

FilmRepository.kt Jetpack Compose :

- Menghubungkan data dari `api` ke dalam model `Film`.
- `api.getPopularMovies(apiKey)` adalah panggilan ke Retrofit.
- Kemudian, hasilnya dikonversi dari response model ke domain model `Film`.

DetailScreen.kt

- UI detail film, menampilkan **judul**, **gambar**, dan **deskripsi** film.
- Gunakan **Coil** (`rememberAsyncImagePainter`) untuk memuat gambar dari URL TMDB.
- Menggunakan `Scaffold` dan `SmallTopAppBar` untuk UI yang konsisten dan material-based.
- `onBack` digunakan agar tombol back bisa kembali ke screen sebelumnya dengan `BackHandler`.

FilmViewModel.kt

Fungsi Utama:

- Mengelola **state film** menggunakan `StateFlow` agar Compose bisa observe data secara reaktif.
- Menangani navigasi dan aksi user dengan event handler berbasis sealed class (`Event`).

Penjelasan Teknis:

- `_filmList` menyimpan daftar film dan diamati dari UI menggunakan `collectAsState()`.
- `_event` digunakan sebagai kanal komunikasi satu arah (event-driven) ke UI, seperti `NavigateToDetail` atau `OpenWebUrl`.
- `viewModelScope.launch {}` memungkinkan coroutine berjalan lifecycle-aware (tidak memory leak).
- `loadFilmsFromApi()` memanggil repository dan menangani error log dengan `Timber`.
-

FilmViewModelFactory.kt
Fungsi Utama:
- Membuat instance dari `FilmViewModel` dengan repository sebagai dependensi eksternal (dependency injection manual).

Penjelasan Teknis:
- Diperlukan karena `FilmViewModel` tidak memiliki constructor kosong.
- Digunakan di HomeScreen untuk membuat ViewModel secara benar.

HomeScreen.kt
Fungsi Utama:
- Menampilkan daftar film populer.
- Meng-handle klik tombol Detail dan Web dengan Event dari ViewModel.
- Menginisialisasi repository dan ViewModel (biasanya bisa dikelola oleh DI seperti Hilt).

Penjelasan Teknis:
- `LaunchedEffect(event)` digunakan untuk menjalankan navigasi satu kali saat event terjadi.
- Navigasi dilakukan dengan `navController.navigate(...)`.
- Gambar dimuat menggunakan `Coil (rememberAsyncImagePainter)`.
- List film ditampilkan menggunakan `LazyColumn`.

MainScreen.kt
Fungsi Utama:
- Menampilkan daftar film seperti `HomeScreen`, namun tidak memuat tombol dan event detail.
- Tampaknya merupakan screen awal atau versi sederhana dari tampilan utama.

Penjelasan Teknis:
- Menggunakan `LazyColumn` untuk menampilkan list.
- Tidak ada aksi klik di `FilmListItem` (komentar placeholder).
- Gunakan `rememberAsyncImagePainter` untuk gambar (via Coil).

Mappers.kt
Fungsi Utama:
- Mengonversi dari `NetworkFilm` (model response API) ke Film (domain model).

Penjelasan Teknis:
- Memisahkan model dari API dengan model domain merupakan praktik Clean Architecture agar layer UI dan repository tidak terikat dengan detail dari API.
- Digunakan di `FilmRepository`.

MovieResponse.kt
Penjelasan:

- Fungsi ekstensi untuk mengonversi `NetworkFilm` (response dari API) menjadi `Film` (model domain).
- Ini penting karena kamu memisahkan data yang didapat dari internet (`NetworkFilm`) dengan data yang digunakan di aplikasi (`Film`), sebuah prinsip Clean Architecture.

MyApplication.kt
Isinya:

- Inisialisasi global seperti `FilmRepository` dan `PreferenceManager`.
- Konfigurasi Retrofit menggunakan Kotlin Serialization.
- Setup Timber untuk logging.

Penjelasan:

- File ini menjalankan `Application`, yang hanya dibuat sekali saat app dijalankan.
- `filmRepository` dan `preferenceManager` disiapkan di sini agar bisa digunakan di mana saja, misalnya disuntikkan ke `ViewModel`.
- `Json { ignoreUnknownKeys = true }` membuat deserializer lebih fleksibel terhadap perubahan API.

NetworkFilm.kt
Penjelasan:

- `@Serializable`: Mengaktifkan serialisasi/deserialisasi dengan KotlinX Serialization.
- `@SerialName(...)`: Menghubungkan nama di JSON dengan properti Kotlin.
- Ini adalah representasi data mentah dari TMDb API.

Navigation.kt
Isinya:

- Menentukan flow navigasi antara dua screen: `HomeScreen` dan `DetailScreen`.
- Gunakan `NavHost` dari Jetpack Compose Navigation.

Penjelasan:

- `NavHost` mendefinisikan struktur screen.
- Data dari `HomeScreen` ke `DetailScreen` dikirim lewat path parameter (`detail/{title}/{imageUrl}/{fullDescription}`).
- `Uri.encode(...)` di `HomeScreen` penting agar teks tidak rusak saat dikirim melalui URL.

PreferenceManager.kt
Isinya:

- Membungkus `SharedPreferences` agar bisa digunakan secara reaktif menggunakan `StateFlow`.
- Menyimpan dan membaca mode gelap/terang (dark mode).

Penjelasan:

- `MutableStateFlow` memungkinkan UI bereaksi jika user mengaktifkan atau mematikan dark mode.
- Menyediakan API `setDarkMode(enabled)` dan `isDarkMode: Flow<Boolean>`.

Result.kt
Fungsi:
- Membungkus status dari operasi async (misal: memanggil API).

Penjelasan:
`sealed class Result<out T>` memiliki tiga kemungkinan:
- `Loading`: Proses sedang berlangsung.
- `Success<T>(val data: T)`: Berhasil dan menyimpan hasil.
- `Error(val exception: Throwable)`: Gagal dan menyimpan error-nya.

RetrofitInstance.kt
Fungsi:
- Singleton untuk membuat instance Retrofit dan `TmdbApi`.

Penjelasan:
- `Json { ignoreUnknownKeys = true }`: Menghindari crash jika JSON API punya field yang tidak didefinisikan di model.
- `by lazy`: Menunda inisialisasi sampai digunakan.

SharedPreferenceViewModel.kt
Fungsi:
- Mengelola dan mengekspos status dark mode dari `PreferenceManager`.

Penjelasan:
- `stateIn(...)`: Mengubah Flow jadi StateFlow, sehingga bisa digunakan di Compose dan Live UI update.
- `toggleDarkMode()`: Membalik status dark mode dan menyimpannya.

SharedPreferenceViewModelFactory.kt
Fungsi:
- Membuat instance `SharedPreferenceViewModel` dengan constructor yang membutuhkan parameter (`PreferenceManager`).

Penjelasan:
- Karena `ViewModelProvider` default hanya bisa buat ViewModel tanpa argumen, kamu butuh factory ini.

TmdbApi.kt
Fungsi:
- Interface untuk endpoint Retrofit ke API TMDb.

Penjelasan:
- Endpoint: `GET https://api.themoviedb.org/3/movie/popular?api_key=...`
- Hasil dikembalikan sebagai `MovieResponse` (yang berisi list `NetworkFilm`).

SettingsScreen.kt
Fungsi:
- Tampilan UI untuk mengubah mode gelap/terang dengan Switch.

Penjelasan:
- Menggunakan `collectAsState()` agar UI auto update saat state `isDarkMode` berubah.

- Ketika Switch diklik, `viewModel.toggleDarkMode()` dipanggil.

E. Tautan GIT

https://github.com/gr1ff0m/Pemrograman-Mobile-Praktikum