LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

CONNECT TO THE INTERNET

Oleh:

Muhammad Raihan NIM. 2310817110008

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE
MODUL 5

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

Nama Praktikan : Muhammad Raihan
NIM : 2310817110008

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

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 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

AppDatabase.kt

Tabel 1. Source Code AppDatabase.kt

```
1  package com.pemrogramanmobile.apicompose.data.local
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, exportSchema = false)
9  abstract class AppDatabase : RoomDatabase() {
10     abstract fun movieDao(): MovieDao
11
```

```
12      companion object {
13          @Volatile
14          private var INSTANCE: AppDatabase? = null
15
16          fun      getDatabase(context:      Context):
    AppDatabase {
17              return INSTANCE ?: synchronized(this) {
18                  val instance = Room.databaseBuilder(
19                      context.applicationContext,
20                      AppDatabase::class.java,
21                      "movie_database"
22                  )

23  .fallbackToDestructiveMigration()
24                      .build()
25                  INSTANCE = instance
26                  instance
27              }
28          }
29      }
30  }
```

MovieDao.kt

Tabel 2. Source Code MovieDao.kt

```
1   package com.pemrogramanmobile.apicompose.data.local
2
3   import androidx.room.Dao
4   import androidx.room.Insert
5   import androidx.room.OnConflictStrategy
6   import androidx.room.Query
7   import kotlinx.coroutines.flow.Flow
8
9   @Dao
10  interface MovieDao {
11      @Insert(onConflict = OnConflictStrategy.REPLACE)
12      suspend         fun         insertMovies(movies:
    List<MovieEntity>)
13
14      @Query("SELECT * FROM movies ORDER BY voteAverage
    DESC")
15      fun getAllMovies(): Flow<List<MovieEntity>>
16
17      @Query("SELECT * FROM movies WHERE id = :movieId")
18      fun          getMovieById(movieId:          Int):
    Flow<MovieEntity?>
19
```

```
20      @Query("DELETE FROM movies")
21      suspend fun deleteAllMovies()
22
23      @Query("SELECT COUNT(*) FROM movies")
24      suspend fun getMovieCount(): Int
25
26      @Query("SELECT lastRefreshed FROM movies ORDER BY
    lastRefreshed DESC LIMIT 1")
27      suspend fun getLastRefreshTimestamp(): Long?
28  }
```

MovieEntity.kt

Tabel 3. Source Code MovieEntity.kt

```
1   package com.pemrogramanmobile.apicompose.data.local
2
3   import androidx.room.Entity
4   import androidx.room.PrimaryKey
5
6   @Entity(tableName = "movies")
7   data class MovieEntity(
8       @PrimaryKey val id: Int,
9       val title: String,
10      val overview: String,
11      val posterPath: String?,
12      val releaseDate: String?,
13      val voteAverage: Double,
14      val         lastRefreshed:         Long         =
    System.currentTimeMillis(),
15      val homepage: String? = null
16  )
```

MovieApiModel.kt

Tabel 4. Source Code MovieApiModel.kt

```
1   package com.pemrogramanmobile.apicompose.data.remote
2
3   import kotlinx.serialization.SerialName
4   import kotlinx.serialization.Serializable
5
6   @Serializable
7   data class MovieListResponse(
8       @SerialName("page") val page: Int,
9       @SerialName("results")        val        results:
    List<MovieApiModel>,
10      @SerialName("total_pages") val totalPages: Int,
```

```
11      @SerialName("total_results") val totalResults:
        Int
12  )
13
14  @Serializable
15  data class MovieApiModel(
16      @SerialName("id") val id: Int,
17      @SerialName("title") val title: String,
18      @SerialName("overview") val overview: String,
19      @SerialName("poster_path")    val    posterPath:
        String?,
20      @SerialName("release_date")   val   releaseDate:
        String?,
21      @SerialName("vote_average")   val   voteAverage:
        Double,
22      @SerialName("homepage") val homepage: String? =
        null
23  )
```

RetrofitClient.kt

Tabel 5. Source Code RetrofitClient.kt

```
1   package com.pemrogramanmobile.apicompose.data.remote
2
3   import
    com.jakewharton.retrofit2.converter.kotlinx.serializ
    ation.asConverterFactory
4   import kotlinx.serialization.json.Json
5   import okhttp3.MediaType.Companion.toMediaType
6   import okhttp3.OkHttpClient
7   import okhttp3.logging.HttpLoggingInterceptor
8   import retrofit2.Retrofit
9
10  object RetrofitClient {
11      private      const      val      BASE_URL      =
    "https://api.themoviedb.org/3/"
12      private      const      val      API_KEY      =
    "91a08b3c95e46f3b968e7dfdfc6c81b0"
13
14      private val json = Json {
15          ignoreUnknownKeys = true
16          coerceInputValues = true
17      }
18
19      private      val      loggingInterceptor      =
    HttpLoggingInterceptor().apply {
20          level = HttpLoggingInterceptor.Level.BODY
```

```
21      }
22
23      private val httpClient = OkHttpClient.Builder()
24          .addInterceptor(loggingInterceptor)
25          .addInterceptor { chain ->
26              val originalRequest = chain.request()
27              val originalHttpUrl = originalRequest.url
28              val url = originalHttpUrl.newBuilder()
29                  .addQueryParameter("api_key",
    API_KEY)
30                  .build()
31              val           requestBuilder           =
    originalRequest.newBuilder().url(url)
32              val request = requestBuilder.build()
33              chain.proceed(request)
            }
34          .build()
35
36      val instance: TmdbApiService by lazy {
37          Retrofit.Builder()
38              .baseUrl(BASE_URL)
39              .client(httpClient)

40  .addConverterFactory(json.asConverterFactory("applic
41  ation/json".toMediaType()))
42              .build()
43              .create(TmdbApiService::class.java)
44      }
45  }
```

TmdbApiService.kt

Tabel 6. Source Code TmdbApiService.kt

```
1   package com.pemrogramanmobile.apicompose.data.remote
2
3   import retrofit2.http.GET
4   import retrofit2.http.Path
5   import retrofit2.http.Query
6
7   interface TmdbApiService {
8       @GET("movie/popular")
9       suspend fun getPopularMovies(
10          @Query("page") page: Int = 1
11      ): MovieListResponse
12
13       @GET("movie/{movie_id}")
14       suspend fun getMovieDetails(
```

```
15          @Path("movie_id") movieId: Int
16      ): MovieApiModel
17  }
```

MovieRepositoryImpl.kt

Tabel 7. Source Code MovieRepositoryImpl.kt

```
1   package
    com.pemrogramanmobile.apicompose.data.repository
2
3   import android.content.Context
4   import
    com.pemrogramanmobile.apicompose.data.local.MovieDao
5   import
    com.pemrogramanmobile.apicompose.data.remote.TmdbApi
    Service
6   import
    com.pemrogramanmobile.apicompose.data.toDomainMovie
7   import
    com.pemrogramanmobile.apicompose.data.toDomainMovies
    FromEntities
8   import
    com.pemrogramanmobile.apicompose.data.toMovieEntitie
    s
9   import
    com.pemrogramanmobile.apicompose.data.toMovieEntity
10  import
    com.pemrogramanmobile.apicompose.domain.model.Movie
11  import
    com.pemrogramanmobile.apicompose.domain.repository.M
    ovieRepository
12  import
    com.pemrogramanmobile.apicompose.util.NetworkResult
13  import kotlinx.coroutines.flow.Flow
14  import kotlinx.coroutines.flow.firstOrNull
15  import kotlinx.coroutines.flow.flow
16  import kotlinx.coroutines.flow.map
17  import java.util.concurrent.TimeUnit
18
19  class MovieRepositoryImpl(
20      private val tmdbApiService: TmdbApiService,
21      private val movieDao: MovieDao,
22      private val context: Context
23  ) : MovieRepository {
24      private       val       CACHE_EXPIRY_MS       =
    TimeUnit.HOURS.toMillis(1)
25
```

```
26      override fun getPopularMovies(): Flow<NetworkResult<List<Movie>>> = flow {
27          val lastRefreshTime = movieDao.getLastRefreshTimestamp() ?: 0L
28          val isCacheStale = (System.currentTimeMillis() - lastRefreshTime) > CACHE_EXPIRY_MS
29          val movieCount = movieDao.getMovieCount()
30          val isCacheEmpty = movieCount == 0
31
32          if (!isCacheEmpty && !isCacheStale) {
33              val cachedDataFlow: Flow<List<Movie>> = movieDao.getAllMovies().map { entities ->
34  entities.toDomainMoviesFromEntities()
35              }
36              val cachedData = cachedDataFlow.firstOrNull()
37              if (cachedData != null && cachedData.isNotEmpty()) {
38  emit(NetworkResult.Success(cachedData))
39              }
40          }
41
42          if (isCacheEmpty || isCacheStale) {
43              try {
44                  val movieListResponse = tmdbApiService.getPopularMovies()
45                  val currentTime = System.currentTimeMillis()
46                  val movieEntities = movieListResponse.results.toMovieEntities(currentTime)
47
48                  movieDao.deleteAllMovies()
49                  movieDao.insertMovies(movieEntities)
50
51                  val freshDataFlow: Flow<List<Movie>> = movieDao.getAllMovies().map { entities ->
52  entities.toDomainMoviesFromEntities()
53                  }
54                  val freshDataFromDb = freshDataFlow.firstOrNull()
55
56                  if (freshDataFromDb != null) {
```

```
57  emit(NetworkResult.Success(freshDataFromDb))
58                  } else {
59                      emit(NetworkResult.Error("Gagal
    mengambil  data  dari  database  setelah  refresh.",
    null))
60                  }
61              } catch (e: Exception) {
62                  val                 fallbackDataFlow:
    Flow<List<Movie>>  =  movieDao.getAllMovies().map  {
    entities ->

63  entities.toDomainMoviesFromEntities()
64                  }
65                  val        currentCachedData        =
    fallbackDataFlow.firstOrNull()
66                  emit(NetworkResult.Error("Gagal
    memuat    data    dari    jaringan:    ${e.message}",
    currentCachedData))
67              }
68          }
69      }
70
71      override   fun   getMovieDetails(movieId:   Int):
    Flow<NetworkResult<Movie>> = flow {
72          val            cachedMovieEntity            =
    movieDao.getMovieById(movieId).firstOrNull()
73          if (cachedMovieEntity != null) {

74  emit(NetworkResult.Success(cachedMovieEntity.toDomai
    nMovie()))
75          }
76
77          try {
78              val            movieApiModel            =
    tmdbApiService.getMovieDetails(movieId)
79              val             movieEntity             =
    movieApiModel.toMovieEntity(System.currentTimeMillis
    ())
80

81  movieDao.insertMovies(listOf(movieEntity))
82

83  emit(NetworkResult.Success(movieApiModel.toDomainMov
    ie()))
84          } catch (e: Exception) {
```

```
85              if (cachedMovieEntity == null) {
86                  emit(NetworkResult.Error("Gagal
    memuat detail film: ${e.message}", null))
87              } else {
88                  emit(NetworkResult.Error("Gagal
    menyegarkan     detail     film:     ${e.message}",
    cachedMovieEntity.toDomainMovie()))
89              }
90          }
91      }
92  }
```

Mappers.kt

Tabel 8. Source Code Mappers.kt

```
1   package com.pemrogramanmobile.apicompose.data
2
3   import
    com.pemrogramanmobile.apicompose.data.local.MovieEnt
    ity
4   import
    com.pemrogramanmobile.apicompose.data.remote.MovieAp
    iModel
5   import
    com.pemrogramanmobile.apicompose.domain.model.Movie
6
7   fun MovieApiModel.toDomainMovie(): Movie {
8       return Movie(
9           id = this.id,
10          title = this.title,
11          overview = this.overview,
12          posterPath = this.posterPath,
13          releaseDate = this.releaseDate,
14          voteAverage = this.voteAverage,
15          homepage = this.homepage
16      )
17  }
18
19  fun   MovieApiModel.toMovieEntity(lastRefreshedTime:
    Long): MovieEntity {
20      return MovieEntity(
21          id = this.id,
22          title = this.title,
23          overview = this.overview,
24          posterPath = this.posterPath,
25          releaseDate = this.releaseDate,
26          voteAverage = this.voteAverage,
```

```
27          lastRefreshed = lastRefreshedTime,
28          homepage = this.homepage
29      )
30  }
31
32  fun MovieEntity.toDomainMovie(): Movie {
33      return Movie(
34          id = this.id,
35          title = this.title,
36          overview = this.overview,
37          posterPath = this.posterPath,
38          releaseDate = this.releaseDate,
39          voteAverage = this.voteAverage,
40          homepage = this.homepage
41      )
42  }
43
44  fun             List<MovieApiModel>.toDomainMovies():
    List<Movie> {
45      return this.map { it.toDomainMovie() }
46  }
47
48  fun  List<MovieEntity>.toDomainMoviesFromEntities():
    List<Movie> {
49      return this.map { it.toDomainMovie() }
50  }
51
52  fun
    List<MovieApiModel>.toMovieEntities(lastRefreshedTim
    e: Long): List<MovieEntity> {
53      return                  this.map                {
    it.toMovieEntity(lastRefreshedTime) }
54  }
```

Movie.kt

Tabel 9. Source Code Movie.kt

```
1   package
    com.pemrogramanmobile.apicompose.domain.model
2
3   data class Movie(
4       val id: Int,
5       val title: String,
6       val overview: String,
7       val posterPath: String?,
8       val releaseDate: String?,
9       val voteAverage: Double,
```

```
10      val homepage: String? = null
11  )
```

MovieRepository.kt

Tabel 10. Source Code MovieRepository.kt

```
1   package com.pemrogramanmobile.apicompose.domain.repository
2
3   import com.pemrogramanmobile.apicompose.domain.model.Movie
4   import com.pemrogramanmobile.apicompose.util.NetworkResult
5   import kotlinx.coroutines.flow.Flow
6
7   interface MovieRepository {
8       fun getPopularMovies(): Flow<NetworkResult<List<Movie>>>
9       fun getMovieDetails(movieId: Int): Flow<NetworkResult<Movie>>
10  }
```

GetMovieDetailsUseCase.kt

Tabel 11. Source Code GetMovieDetailsUseCase.kt

```
1   package com.pemrogramanmobile.apicompose.domain.usecase
2
3   import com.pemrogramanmobile.apicompose.domain.model.Movie
4   import com.pemrogramanmobile.apicompose.domain.repository.MovieRepository
5   import com.pemrogramanmobile.apicompose.util.NetworkResult
6   import kotlinx.coroutines.flow.Flow
7
8   class GetMovieDetailsUseCase(private val movieRepository: MovieRepository) {
9       operator fun invoke(movieId: Int): Flow<NetworkResult<Movie>> {
10          return movieRepository.getMovieDetails(movieId)
11      }
12  }
```

GetPopularMoviesUseCase.kt

Tabel 12. Source Code GetPopularMoviesUseCase.kt

```
1  package com.pemrogramanmobile.apicompose.domain.usecase
2
3  import com.pemrogramanmobile.apicompose.domain.model.Movie
4  import com.pemrogramanmobile.apicompose.domain.repository.MovieRepository
5  import com.pemrogramanmobile.apicompose.util.NetworkResult
6  import kotlinx.coroutines.flow.Flow
7
8  class GetPopularMoviesUseCase(private val movieRepository: MovieRepository) {
9      operator fun invoke(): Flow<NetworkResult<List<Movie>>> {
10         return movieRepository.getPopularMovies()
11     }
12 }
```

MovieDetailScreen.kt

Tabel 13. Source Code MovieDetailScreen.kt

```
1  package com.pemrogramanmobile.apicompose.ui.screen.moviedetail
2
3  import android.content.ActivityNotFoundException
4  import android.content.Intent
5  import android.net.Uri
6  import android.widget.Toast
7  import androidx.compose.foundation.layout.Arrangement
8  import androidx.compose.foundation.layout.Box
9  import androidx.compose.foundation.layout.Column
10 import androidx.compose.foundation.layout.Row
11 import androidx.compose.foundation.layout.Spacer
12 import androidx.compose.foundation.layout.aspectRatio
13 import androidx.compose.foundation.layout.fillMaxSize
14 import androidx.compose.foundation.layout.fillMaxWidth
15 import androidx.compose.foundation.layout.height
16 import androidx.compose.foundation.layout.padding
```

```
17  import androidx.compose.foundation.layout.size
18  import
    androidx.compose.foundation.rememberScrollState
19  import
    androidx.compose.foundation.shape.RoundedCornerShape
20  import androidx.compose.foundation.verticalScroll
21  import androidx.compose.material.icons.Icons
22  import
    androidx.compose.material.icons.automirrored.filled.
    ArrowBack
23  import androidx.compose.material.icons.filled.Search
24  import androidx.compose.material3.Button
25  import androidx.compose.material3.ButtonDefaults
26  import
    androidx.compose.material3.CircularProgressIndicator
27  import
    androidx.compose.material3.ExperimentalMaterial3Api
28  import androidx.compose.material3.Icon
29  import androidx.compose.material3.IconButton
30  import androidx.compose.material3.MaterialTheme
31  import androidx.compose.material3.Scaffold
32  import androidx.compose.material3.SnackbarDuration
33  import androidx.compose.material3.SnackbarHost
34  import androidx.compose.material3.SnackbarHostState
35  import androidx.compose.material3.Text
36  import androidx.compose.material3.TopAppBar
37  import androidx.compose.material3.TopAppBarDefaults
38  import androidx.compose.runtime.Composable
39  import androidx.compose.runtime.LaunchedEffect
40  import androidx.compose.runtime.getValue
41  import androidx.compose.runtime.remember
42  import androidx.compose.ui.Alignment
43  import androidx.compose.ui.Modifier
44  import androidx.compose.ui.draw.clip
45  import androidx.compose.ui.layout.ContentScale
46  import androidx.compose.ui.platform.LocalContext
47  import androidx.compose.ui.res.painterResource
48  import androidx.compose.ui.res.stringResource
49  import androidx.compose.ui.text.font.FontWeight
50  import androidx.compose.ui.text.style.TextAlign
51  import androidx.compose.ui.unit.dp
52  import
    androidx.lifecycle.compose.collectAsStateWithLifecyc
    le
53  import
    androidx.lifecycle.viewmodel.compose.viewModel
54  import androidx.navigation.NavController
```

```
55  import coil.compose.AsyncImage
56  import coil.request.ImageRequest
57  import com.pemrogramanmobile.apicompose.R
58  import
    com.pemrogramanmobile.apicompose.domain.model.Movie
59  import
    com.pemrogramanmobile.apicompose.ui.viewmodel.MovieD
    etailViewModel
60
61  @OptIn(ExperimentalMaterial3Api::class)
62  @Composable
63  fun MovieDetailScreen(
64      navController: NavController,
65      viewModel:         MovieDetailViewModel         =
    viewModel(factory = MovieDetailViewModel.Factory)
66  ) {
67      val                   uiState                   by
    viewModel.uiState.collectAsStateWithLifecycle()
68      val     snackbarHostState     =     remember     {
    SnackbarHostState() }
69
70      LaunchedEffect(uiState.snackbarMessage) {
71          uiState.snackbarMessage?.let { message ->
72              snackbarHostState.showSnackbar(message,
    duration = SnackbarDuration.Short)
73              viewModel.clearSnackbarMessage()
74          }
75      }
76
77      Scaffold(
78          snackbarHost                =                {
    SnackbarHost(snackbarHostState) },
79          topBar = {
80              TopAppBar(
81                  title  =  {  Text(uiState.movie?.title
    ?: stringResource(R.string.detail_film)) },
82                  navigationIcon = {
83                      IconButton(onClick      =      {
    navController.navigateUp() }) {

84  Icon(Icons.AutoMirrored.Filled.ArrowBack,
    contentDescription = "Kembali")
85                      }
                    },
86                  colors                                =
    TopAppBarDefaults.topAppBarColors(
```

```
87                      containerColor                  =
    MaterialTheme.colorScheme.primary,
88                      titleContentColor               =
    MaterialTheme.colorScheme.onPrimary,
89                      navigationIconContentColor      =
    MaterialTheme.colorScheme.onPrimary
90                  )
91              )
92          }
93      ) { paddingValues ->
94          Box(
95              modifier = Modifier
96                  .fillMaxSize()
97                  .padding(paddingValues)
98          ) {
99              when {
100                 uiState.isLoading -> {

    CircularProgressIndicator(modifier                  =
    Modifier.align(Alignment.Center))
101                 }
102                 uiState.error != null -> {
103                     Column(
104                         modifier                    =
    Modifier.align(Alignment.Center).padding(16.dp),
105                         horizontalAlignment         =
    Alignment.CenterHorizontally,
106                         verticalArrangement         =
    Arrangement.Center
107                     ) {
108                         Text(
109                             text       =       "Oops!
    ${uiState.error}",
110                             style                   =
    MaterialTheme.typography.bodyLarge,
111                             color                   =
    MaterialTheme.colorScheme.error,
112                             textAlign               =
    TextAlign.Center
113                         )
114                         Spacer(modifier             =
    Modifier.height(8.dp))
115                         Button(onClick      =      {
    viewModel.retryFetch() }) {
116
117 Text(stringResource(R.string.try_again))
118                         }
```

```
119                     }
120                 }
121                 uiState.movie != null -> {
122                     MovieDetailContent(movie       =
    uiState.movie!!)
123                 }
124                 else -> {
125                     Text(
126                         text                       =
    stringResource(R.string.no_detail),
127                         modifier                   =
    Modifier.align(Alignment.Center).padding(16.dp),
128                         style                      =
    MaterialTheme.typography.bodyMedium,
129                         textAlign = TextAlign.Center
130                     )
131                 }
132             }
133         }
134     }
135 }
136
137 @Composable
138 fun MovieDetailContent(movie: Movie) {
139     val context = LocalContext.current
140     Column(
141         modifier = Modifier
142             .fillMaxSize()
143             .verticalScroll(rememberScrollState())
144             .padding(16.dp)
145     ) {
146         AsyncImage(
147             model                                  =
    ImageRequest.Builder(LocalContext.current)
148                 .data(if (movie.posterPath != null)
    "https://image.tmdb.org/t/p/w500${movie.posterPath}"
    else null)
149                 .crossfade(true)
150                 .build(),
151             placeholder   =   painterResource(id   =
    R.drawable.ic_placeholder_image),
152             error     =     painterResource(id     =
    R.drawable.ic_broken_image),
153             contentDescription       =       "Poster
    ${movie.title}",
154             modifier = Modifier
155                 .fillMaxWidth()
```

```
156                 .aspectRatio(2f / 3f)
157                 .clip(RoundedCornerShape(12.dp))

158 .align(Alignment.CenterHorizontally),
159             contentScale = ContentScale.Crop
160         )
161
162         Spacer(modifier = Modifier.height(16.dp))
163
164         Text(
165             text = movie.title,
166             style                                   =
    MaterialTheme.typography.headlineMedium,
167             fontWeight = FontWeight.Bold,
168             textAlign = TextAlign.Center,
169             modifier = Modifier.fillMaxWidth()
170         )
171
172         Spacer(modifier = Modifier.height(8.dp))
173
174         Row(
175             modifier = Modifier.fillMaxWidth(),
176             horizontalArrangement                   =
    Arrangement.SpaceBetween,
177             verticalAlignment                       =
    Alignment.CenterVertically
178         ) {
179             Text(
180                 text = "Rilis: ${movie.releaseDate ?:
    "Tidak diketahui"}",
181                 style                               =
    MaterialTheme.typography.bodyMedium,
182                 fontWeight = FontWeight.SemiBold
183             )
184             Text(
185                 text            =            "Rating:
    ${String.format("%.1f", movie.voteAverage)}/10",
186                 style                               =
    MaterialTheme.typography.bodyMedium,
187                 fontWeight = FontWeight.SemiBold
188             )
189         }
190
191         Spacer(modifier = Modifier.height(16.dp))
192
193         Text(
```

```
194             text                                    =
    stringResource(R.string.overview),
195             style                                   =
    MaterialTheme.typography.titleMedium,
196             fontWeight = FontWeight.Bold
197         )
198         Spacer(modifier = Modifier.height(4.dp))
199         Text(
200             text = movie.overview,
201             style                                   =
    MaterialTheme.typography.bodyMedium,
202             textAlign = TextAlign.Justify
203         )
204
205         if (!movie.homepage.isNullOrBlank()) {
206             Spacer(modifier                         =
    Modifier.height(24.dp))
207             Button(
208                 onClick = {
209                     val           intent            =
    Intent(Intent.ACTION_VIEW,
    Uri.parse(movie.homepage)).apply {
210
    setPackage("com.android.chrome")
211                     }
212                     try {

213 context.startActivity(intent)
214                     }           catch           (e:
    ActivityNotFoundException) {
215                         val     fallbackIntent     =
    Intent(Intent.ACTION_VIEW,
    Uri.parse(movie.homepage))
216                         try {

217 context.startActivity(fallbackIntent)
218                         }         catch         (ex:
    ActivityNotFoundException) {
219                             Toast.makeText(context,
    "Tidak     ada     browser     yang     ditemukan!",
    Toast.LENGTH_SHORT).show()
220                         }
221                     }
222                 },
223                 modifier = Modifier.fillMaxWidth(),
224                 shape = RoundedCornerShape(12.dp)
225             ) {
```

```
226                 Icon(Icons.Default.Search,
    contentDescription = "Homepage", modifier =
    Modifier.size(ButtonDefaults.IconSize))

227 Spacer(Modifier.size(ButtonDefaults.IconSpacing))

228 Text(stringResource(R.string.visit_website))
229             }
230         }
231         Spacer(modifier = Modifier.height(16.dp))
232     }
233 }
234
```

MovieItem.kt

Tabel 14. Source Code MovieItem.kt

```
1   package
2   com.pemrogramanmobile.apicompose.ui.screen.movielist
3
4   import android.content.ActivityNotFoundException
5   import android.content.Intent
6   import android.net.Uri
7   import android.widget.Toast
8   import androidx.compose.foundation.layout.*
9   import
10  androidx.compose.foundation.shape.RoundedCornerShape
11  import androidx.compose.material3.*
12  import androidx.compose.runtime.Composable
13  import androidx.compose.ui.Alignment
14  import androidx.compose.ui.Modifier
15  import androidx.compose.ui.draw.clip
16  import androidx.compose.ui.layout.ContentScale
17  import androidx.compose.ui.platform.LocalContext
18  import androidx.compose.ui.res.painterResource
19  import androidx.compose.ui.res.stringResource
20  import androidx.compose.ui.text.font.FontWeight
21  import androidx.compose.ui.text.style.TextAlign
22  import androidx.compose.ui.text.style.TextOverflow
23  import androidx.compose.ui.unit.dp
24  import androidx.compose.ui.unit.sp
25  import coil.compose.AsyncImage
26  import coil.request.ImageRequest
27  import com.pemrogramanmobile.apicompose.R
28  import
29  com.pemrogramanmobile.apicompose.domain.model.Movie
30
```

```
@Composable
fun MovieItem(
    movie: Movie,
    modifier: Modifier = Modifier,
    onDetailsClick: (movieId: Int) -> Unit
) {
    val context = LocalContext.current

    Card(
        modifier = modifier
            .padding(7.dp)
            .fillMaxWidth()
            .wrapContentHeight(),
        shape = MaterialTheme.shapes.medium,
        colors = CardDefaults.cardColors(
            containerColor                                =
MaterialTheme.colorScheme.surface
        ),
        elevation = CardDefaults.cardElevation(
            defaultElevation = 5.dp
        )
    ) {
        Row(
            verticalAlignment = Alignment.Top,
            modifier = Modifier.padding(5.dp)
        ) {
            AsyncImage(
                model                                     =
ImageRequest.Builder(LocalContext.current)
                    .data(if   (movie.posterPath   !=
null)
"https://image.tmdb.org/t/p/w342${movie.posterPath}"
else null)
                    .crossfade(true)
                    .build(),
                placeholder  =  painterResource(id  =
R.drawable.ic_placeholder_image),
                error    =    painterResource(id    =
R.drawable.ic_broken_image),
                contentDescription     =     "Poster
${movie.title}",
                modifier = Modifier
                    .size(width  =  120.dp,  height  =
150.dp)
                    .padding(8.dp)

.align(Alignment.CenterVertically)
```

```
78
79  .clip(RoundedCornerShape(15.dp)),
80                  contentScale                            =
81  ContentScale.FillBounds,
82              )
83              Column(
84                  Modifier
85                      .weight(1f)
86                      .padding(start  =  0.dp,  top  =
87  8.dp, end = 8.dp, bottom = 8.dp)
88              ) {
89                  Row (
90                      modifier                            =
91  Modifier.fillMaxWidth(),
92                      verticalAlignment                   =
93  Alignment.CenterVertically
94                  ) {
95                      Text(
96                          text = movie.title,
97                          fontSize = 20.sp,
98                          fontWeight                      =
99  FontWeight.Bold,
100                         color                           =
101 MaterialTheme.colorScheme.onSurface,
102                         modifier                        =
103 Modifier.weight(1f),
104                         maxLines = 2,
105                         overflow                        =
106 TextOverflow.Ellipsis
107                     )
108                     Spacer(Modifier.width(10.dp))
109                     movie.releaseDate?.let {
110                         if (it.isNotBlank()) {
111                             Text(
112                                 text  =  it.split("-
113 ").firstOrNull() ?: it,
114                                 style               =
115 MaterialTheme.typography.labelMedium,
116                                 fontSize = 15.sp,
117                                 color               =
118 MaterialTheme.colorScheme.onSurface
119                             )
120                         }
121                     }
122                 }
123
124                 Spacer(Modifier.height(8.dp))
```

```
125                 Text(
126                     text                                =
127 stringResource(R.string.overview),
128                     style                               =
129 MaterialTheme.typography.labelMedium,
130                     fontWeight                          =
131 FontWeight.SemiBold,
132                     color                               =
133 MaterialTheme.colorScheme.onSurface
134                 )
135                 Text(
136                     text = movie.overview,
137                     style                               =
138 MaterialTheme.typography.bodySmall,
139                     color                               =
140 MaterialTheme.colorScheme.onSurface,
141                     textAlign = TextAlign.Justify,
142                     maxLines = 3,
143                     overflow = TextOverflow.Ellipsis
144                 )
145                 Spacer(Modifier.height(12.dp))
146
147                 Text(
148                     text          =          "Rating
149 ${String.format("%.1f", movie.voteAverage)} / 10",
150                     style                               =
151 MaterialTheme.typography.bodyMedium,
152                     fontWeight                          =
153 FontWeight.SemiBold,
154                     color                               =
155 MaterialTheme.colorScheme.onSurface,
156                     modifier                            =
157 Modifier.fillMaxWidth()
158                 )
159                 Spacer(Modifier.height(12.dp))
160
161                 Row (
162                     Modifier.fillMaxWidth(),
163                     horizontalArrangement           =
164 Arrangement.End
165                 ) {
166                     Button(
167                         onClick = {
                                movie.homepage?.let    {
    url ->
                                    if
    (url.isNotBlank()) {
```

```
                                        val intent =
Intent(Intent.ACTION_VIEW, Uri.parse(url)).apply {

setPackage("com.android.chrome")
                                        }
                                        try {

context.startActivity(intent)
                                        } catch (e:
ActivityNotFoundException) {
                                            val
fallbackIntent     =     Intent(Intent.ACTION_VIEW,
Uri.parse(url))
                                            try {

context.startActivity(fallbackIntent)
                                            } catch (ex:
ActivityNotFoundException) {

Toast.makeText(context,  "Tidak  ada  browser  yang
ditemukan!", Toast.LENGTH_SHORT).show()
                                            }
                                        }
                                    }
                                }
                            },
                            shape                       =
RoundedCornerShape(12.dp),
                            contentPadding              =
PaddingValues(horizontal = 15.dp),
                            enabled                     =
!movie.homepage.isNullOrBlank()
                        ) {

Text(stringResource(R.string.website),  fontSize  =
13.sp)
                        }
                        Spacer(Modifier.width(10.dp))
                        Button(
                            onClick         =         {
onDetailsClick(movie.id) },
                            shape                       =
RoundedCornerShape(12.dp),
                            contentPadding              =
PaddingValues(horizontal = 15.dp),
                        ) {
```

```
                Text(stringResource(R.string.details),   fontSize   =
                13.sp)
                                }
                            }
                        }
                    }
                }
            }
```

MovieListScreen.kt

Tabel 15. Source Code MovieListScreen.kt

```
1   package
2   com.pemrogramanmobile.apicompose.ui.screen.movielist
3
4   import android.content.res.Configuration
5   import androidx.compose.foundation.layout.*
6   import androidx.compose.foundation.lazy.LazyColumn
7   import androidx.compose.foundation.lazy.items
8   import androidx.compose.material.icons.Icons
9   import
10  androidx.compose.material.icons.filled.Refresh
11  import androidx.compose.material3.*
12  import androidx.compose.runtime.*
13  import androidx.compose.ui.Alignment
14  import androidx.compose.ui.Modifier
15  import androidx.compose.ui.platform.LocalContext
16  import androidx.compose.ui.res.stringResource
17  import androidx.compose.ui.tooling.preview.Preview
18  import androidx.compose.ui.unit.dp
19  import
20  androidx.lifecycle.compose.collectAsStateWithLifecyc
21  le
22  import
23  androidx.lifecycle.viewmodel.compose.viewModel
24  import androidx.navigation.NavController
25  import
26  androidx.navigation.compose.rememberNavController
27  import
28  com.pemrogramanmobile.apicompose.MyApplication
29  import com.pemrogramanmobile.apicompose.R
30  import
31  com.pemrogramanmobile.apicompose.data.remote.Retrofi
32  tClient
33
34
```

```
35  import
36  com.pemrogramanmobile.apicompose.data.repository.Mov
37  ieRepositoryImpl
38  import
39  com.pemrogramanmobile.apicompose.domain.usecase.GetP
40  opularMoviesUseCase
41  import com.pemrogramanmobile.apicompose.ui.Routes
42  import
43  com.pemrogramanmobile.apicompose.ui.theme.ApiCompose
44  Theme
45  import
46  com.pemrogramanmobile.apicompose.ui.viewmodel.MovieV
47  iewModel
48  import
49  com.pemrogramanmobile.apicompose.ui.viewmodel.MovieV
50  iewModelFactory
51
52  @OptIn(ExperimentalMaterial3Api::class)
53  @Composable
54  fun MovieListScreen(navController: NavController) {
55      val title = stringResource(R.string.title)
56
57      val                    context                    =
58  LocalContext.current.applicationContext
59      val movieRepositoryImpl = remember(context) {
60          MovieRepositoryImpl(
61              tmdbApiService                             =
62  RetrofitClient.instance,
63              movieDao       =       (context       as
64  MyApplication).database.movieDao(),
65              context = context
66          )
67      }
68      val           getPopularMoviesUseCase           =
69  remember(movieRepositoryImpl) {
70          GetPopularMoviesUseCase(movieRepositoryImpl)
71      }
72      val movieViewModel: MovieViewModel = viewModel(
73          factory                                        =
74  MovieViewModelFactory(getPopularMoviesUseCase)
75      )
76
77      val                    uiState                    by
78  movieViewModel.uiState.collectAsStateWithLifecycle()
79      val    snackbarHostState    =    remember    {
80  SnackbarHostState() }
81
```

```
82      LaunchedEffect(uiState.snackbarMessage) {
83          uiState.snackbarMessage?.let { message ->
84              snackbarHostState.showSnackbar(
85                  message = message,
86                  duration = SnackbarDuration.Short
87              )
88              movieViewModel.clearSnackbarMessage()
89          }
90      }
91
92      Scaffold(
93          snackbarHost               =               {
94  SnackbarHost(snackbarHostState) },
95          topBar = {
96              TopAppBar(
97                  title = { Text(title) },
98                  colors                                =
99  TopAppBarDefaults.topAppBarColors(
100                     containerColor                    =
101 MaterialTheme.colorScheme.primary,
102                     titleContentColor                 =
103 MaterialTheme.colorScheme.onPrimary
104                 ),
105                 actions = {
106                     IconButton(onClick      =      {
107 movieViewModel.refreshMovies() }) {
108                         Icon(
109                             Icons.Filled.Refresh,
110                             contentDescription        =
111 "Refresh Film",
112                             tint                      =
113 MaterialTheme.colorScheme.onPrimary
114                         )
115                     }
116                 }
117             )
118         }
119     ) { innerPadding ->
120         Box(
121             modifier = Modifier
122                 .fillMaxSize()
123                 .padding(innerPadding)
124         ) {
125             if          (uiState.isLoading          &&
126 uiState.movies.isEmpty()) {
127                 CircularProgressIndicator(modifier =
128 Modifier.align(Alignment.Center))
```

```
129             }
130             else if (uiState.error != null &&
131 uiState.movies.isEmpty()) {
132                 Column(
133                     modifier                        =
134 Modifier.align(Alignment.Center).padding(16.dp),
135                     horizontalAlignment             =
136 Alignment.CenterHorizontally,
137                     verticalArrangement             =
138 Arrangement.Center
139                 ) {
140                     Text(
141                         text         =         "Oops!
142 ${uiState.error}",
143                         style                       =
144 MaterialTheme.typography.bodyLarge,
145                         color                       =
146 MaterialTheme.colorScheme.error
147                     )
148                     Spacer(modifier                 =
149 Modifier.height(8.dp))
150                     Button(onClick        =        {
151 movieViewModel.refreshMovies() }) {
152
153 Text(stringResource(R.string.try_again))
154                     }
155                 }
156             }
157             else if (uiState.movies.isNotEmpty()) {
158                 LazyColumn(
159                     modifier                        =
160 Modifier.fillMaxSize(),
                    contentPadding                  =
    PaddingValues(horizontal = 16.dp, vertical = 8.dp)
                ) {
                    items(uiState.movies,  key  =  {
    movie -> movie.id }) { movie ->
                        MovieItem(
                            movie = movie,
                            modifier                =
    Modifier.fillMaxWidth(),
                            onDetailsClick    =    {
    movieId ->

    navController.navigate(Routes.navigateById(movieId))
                            }
                        )
```

```
                        Spacer(modifier =
Modifier.height(8.dp))
                    }
                }
            }
            else if (!uiState.isLoading &&
uiState.error == null) {
                Text(
                    text =
stringResource(R.string.no_film_found),
                    modifier =
Modifier.align(Alignment.Center).padding(16.dp),
                    style =
MaterialTheme.typography.bodyMedium
                )
            }

            if (uiState.isLoading &&
uiState.movies.isNotEmpty()) {
                LinearProgressIndicator(modifier =
Modifier.fillMaxWidth().align(Alignment.TopCenter))
            }
        }
    }
}

@Preview(showBackground = true, uiMode =
Configuration.UI_MODE_NIGHT_NO)
@Composable
fun MovieListScreenLightPreview() {
    ApiComposeTheme {
        Surface {
            MovieListScreen(navController =
rememberNavController())
        }
    }
}

@Preview(showBackground = true, uiMode =
Configuration.UI_MODE_NIGHT_YES)
@Composable
fun MovieListScreenDarkPreview() {
    ApiComposeTheme {
        Surface {
            MovieListScreen(navController =
rememberNavController())
        }
```

	} }

MovieDetailViewModel.kt

Tabel 16. Source Code MovieDetailViewModel.kt

1	package com.pemrogramanmobile.apicompose.ui.viewmodel
2	
3	import androidx.lifecycle.SavedStateHandle
4	import androidx.lifecycle.ViewModel
5	import androidx.lifecycle.ViewModelProvider
6	import androidx.lifecycle.createSavedStateHandle
7	import androidx.lifecycle.viewModelScope
8	import androidx.lifecycle.viewmodel.CreationExtras
9	import com.pemrogramanmobile.apicompose.MyApplication
10	import com.pemrogramanmobile.apicompose.data.remote.RetrofitClient
11	import com.pemrogramanmobile.apicompose.data.repository.MovieRepositoryImpl
12	import com.pemrogramanmobile.apicompose.domain.model.Movie
13	import com.pemrogramanmobile.apicompose.domain.usecase.GetMovieDetailsUseCase
14	import com.pemrogramanmobile.apicompose.util.NetworkResult
15	import kotlinx.coroutines.flow.MutableStateFlow
16	import kotlinx.coroutines.flow.StateFlow
17	import kotlinx.coroutines.flow.asStateFlow
18	import kotlinx.coroutines.flow.launchIn
19	import kotlinx.coroutines.flow.onEach
20	import kotlinx.coroutines.flow.update
21	
22	data class MovieDetailUiState(
23	val movie: Movie? = null,
24	val isLoading: Boolean = false,
25	val error: String? = null,
26	val snackbarMessage: String? = null
27	)
28	
29	class MovieDetailViewModel(
30	private val getMovieDetailsUseCase: GetMovieDetailsUseCase,

```
    private val savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(MovieDetailUiState(isLoading = true))
    val uiState: StateFlow<MovieDetailUiState> = _uiState.asStateFlow()

    private val movieId: Int = savedStateHandle.get<String>("movieId")?.toIntOrNull() ?: 0

    init {
        if (movieId != 0) {
            fetchMovieDetails(movieId)
        } else {
            _uiState.update {
                it.copy(isLoading = false, error = "Movie ID tidak valid.")
            }
        }
    }

    private fun fetchMovieDetails(id: Int) {
        _uiState.update { it.copy(isLoading = true, error = null, snackbarMessage = null) }

        getMovieDetailsUseCase(id)
            .onEach { result ->
                _uiState.update { currentState ->
                    when (result) {
                        is NetworkResult.Success -> {
                            currentState.copy(
                                movie = result.data,
                                isLoading = false,
                                error = null
                            )
                        }
                        is NetworkResult.Error -> {
                            val errorMessage = result.message ?: "Terjadi kesalahan tidak diketahui"
                            if (currentState.movie != null) {
                                currentState.copy(
                                    isLoading = false,
```

```
68                                          snackbarMessage
    = errorMessage
69                                      )
70                                  } else {
71                                      currentState.copy(
72                                          movie = null,
73                                          isLoading      =
    false,
74                                          error          =
    errorMessage
75                                      )
76                                  }
77                              }
78                          }
79                      }
80                  }
81                  .launchIn(viewModelScope)
82      }
83
84      fun clearSnackbarMessage() {
85          _uiState.update { it.copy(snackbarMessage =
    null) }
86      }
87
88      fun retryFetch() {
89          if (movieId != 0) {
90              fetchMovieDetails(movieId)
91          }
92      }
93
94      companion object {
95          val  Factory:  ViewModelProvider.Factory  =
    object : ViewModelProvider.Factory {
96              @Suppress("UNCHECKED_CAST")
97              override fun <T : ViewModel> create(
98                  modelClass: Class<T>,
99                  extras: CreationExtras
100             ): T {
101                 val           application           =
    checkNotNull(extras[ViewModelProvider.AndroidViewMod
    elFactory.APPLICATION_KEY]) as MyApplication
102                 val        savedStateHandle        =
    extras.createSavedStateHandle()
103
104                 val        movieRepository         =
    MovieRepositoryImpl(
105                     RetrofitClient.instance,
```

```
106                     application.database.movieDao(),
107                     application.applicationContext
108                 )
109                 val     getMovieDetailsUseCase     =
    GetMovieDetailsUseCase(movieRepository)
110
111                 return MovieDetailViewModel(
112                     getMovieDetailsUseCase,
113                     savedStateHandle
114                 ) as T
115             }
116         }
117     }
118 }
```

MovieViewModel.kt

Tabel 17. Source Code MovieViewModel.kt

```
1   package
    com.pemrogramanmobile.apicompose.ui.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.ViewModelProvider
5   import androidx.lifecycle.viewModelScope
6   import
    com.pemrogramanmobile.apicompose.domain.model.Movie
7   import
    com.pemrogramanmobile.apicompose.domain.usecase.GetP
    opularMoviesUseCase
8   import
    com.pemrogramanmobile.apicompose.util.NetworkResult
9   import kotlinx.coroutines.flow.MutableStateFlow
10  import kotlinx.coroutines.flow.StateFlow
11  import kotlinx.coroutines.flow.asStateFlow
12  import kotlinx.coroutines.flow.launchIn
13  import kotlinx.coroutines.flow.onEach
14  import kotlinx.coroutines.flow.update
15
16  data class MovieListUiState(
17      val movies: List<Movie> = emptyList(),
18      val isLoading: Boolean = false,
19      val error: String? = null,
20      val snackbarMessage: String? = null
21  )
22
23  class MovieViewModel(
```

```
24      private val getPopularMoviesUseCase: GetPopularMoviesUseCase
25  ) : ViewModel() {
26
27      private val _uiState = MutableStateFlow(MovieListUiState(isLoading = true))
28      val uiState: StateFlow<MovieListUiState> = _uiState.asStateFlow()
29
30      init {
31          fetchPopularMovies()
32      }
33
34      private fun fetchPopularMovies(forceRefresh: Boolean = false) {
35          if (forceRefresh || _uiState.value.movies.isEmpty()) {
36              _uiState.update { it.copy(isLoading = true, error = null, snackbarMessage = null) }
37          }
38
39          getPopularMoviesUseCase()
40              .onEach { result ->
41                  _uiState.update { currentState ->
42                      when (result) {
43                          is NetworkResult.Success -> {
44                              currentState.copy(
45                                  movies = result.data ?: emptyList(),
46                                  isLoading = false,
47                                  error = null,
48                                  snackbarMessage = null
49                              )
50                          }
51                          is NetworkResult.Error -> {
52                              val errorMessage = result.message ?: "Terjadi kesalahan tidak diketahui"
53                              if (result.data.isNullOrEmpty()) {
54                                  currentState.copy(
55                                      movies = emptyList(),
56                                      isLoading = false,
57                                      error = errorMessage,
```

```
58                                          snackbarMessage
    = null
59                                      )
60                                  } else {
61                                      currentState.copy(
62                                          movies         =
    result.data,
63                                          isLoading      =
    false,
64                                          error = null,
65                                          snackbarMessage
    = errorMessage
66                                      )
67                                  }
68                              }
69                          }
70                      }
71                  }
72                  .launchIn(viewModelScope)
73      }
74
75      fun clearSnackbarMessage() {
76          _uiState.update { it.copy(snackbarMessage =
    null) }
77      }
78
79      fun refreshMovies() {
80          fetchPopularMovies(forceRefresh = true)
81      }
82  }
83
84  class MovieViewModelFactory(
85      private       val       getPopularMoviesUseCase:
    GetPopularMoviesUseCase
86  ) : ViewModelProvider.Factory {
87      override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
88          if
    (modelClass.isAssignableFrom(MovieViewModel::class.j
    ava)) {
89              @Suppress("UNCHECKED_CAST")
90              return
    MovieViewModel(getPopularMoviesUseCase) as T
91          }
92          throw      IllegalArgumentException("Unknown
    ViewModel class")
93      }
```

```
94  }
```

AppNavigation.kt

Tabel 18. Source Code AppNavigation.kt

```
1   package com.pemrogramanmobile.apicompose.ui
2
3   import androidx.compose.runtime.Composable
4   import androidx.navigation.NavType
5   import androidx.navigation.compose.NavHost
6   import androidx.navigation.compose.composable
7   import
    androidx.navigation.compose.rememberNavController
8   import androidx.navigation.navArgument
9   import
    com.pemrogramanmobile.apicompose.ui.screen.moviedeta
    il.MovieDetailScreen
10  import
    com.pemrogramanmobile.apicompose.ui.screen.movielist
    .MovieListScreen
11
12  @Composable
13  fun AppNavigation() {
14      val navController = rememberNavController()
15      NavHost(navController      =      navController,
    startDestination = Routes.MOVIE_LIST) {
16          composable(Routes.MOVIE_LIST) {
17              MovieListScreen(navController          =
    navController)
18          }
19          composable(
20              route = Routes.MOVIE_DETAIL,
21              arguments                              =
    listOf(navArgument("movieId")      {      type      =
    NavType.StringType })
22          ) {
23              MovieDetailScreen(navController        =
    navController)
24          }
25      }
26  }
```

Routes.kt

Tabel 19. Source Code Routes.kt

```
1   package com.pemrogramanmobile.apicompose.ui
2
```

```
3   object Routes {
4       const val MOVIE_LIST = "movieList"
5       const val MOVIE_DETAIL = "movieDetail/{movieId}"
6
7       fun navigateById(id: Int): String{
8           return "movieDetail/${id}"
9       }
10  }
```

AppSettings.kt

Tabel 20. Source Code AppSettings.kt

```
1   package com.pemrogramanmobile.apicompose.util
2
3   import android.content.Context
4   import android.content.SharedPreferences
5
6   object AppSettings {
7       private const val PREFS_NAME =
    "app_settings_prefs"
8       private const val KEY_LAST_CACHE_TIMESTAMP_MOVIES
    = "last_cache_timestamp_movies"
9       private const val KEY_UI_THEME = "ui_theme"
10
11      private fun getPrefs(context: Context):
    SharedPreferences {
12          return
    context.getSharedPreferences(PREFS_NAME,
    Context.MODE_PRIVATE)
13      }
14
15      fun getLastMoviesCacheTimestamp(context:
    Context): Long {
16          return
    getPrefs(context).getLong(KEY_LAST_CACHE_TIMESTAMP_M
    OVIES, 0L)
17      }
18
19      fun setLastMoviesCacheTimestamp(context:
    Context, timestamp: Long) {

20  getPrefs(context).edit().putLong(KEY_LAST_CACHE_TIME
    STAMP_MOVIES, timestamp).apply()
21      }
22
23      fun getTheme(context: Context): String {
```

```
24          return
    getPrefs(context).getString(KEY_UI_THEME,  "system")
    ?: "system"
25      }
26
27      fun setTheme(context: Context, theme: String) {

28  getPrefs(context).edit().putString(KEY_UI_THEME,
    theme).apply()
29      }
30  }
```

NetworkResult.kt

Tabel 21. Source Code NetworkResult.kt

```
1   package com.pemrogramanmobile.apicompose.util
2
3   sealed class NetworkResult<T>(
4       val data: T? = null,
5       val message: String? = null
6   ) {
7       class        Success<T>(data:        T)        :
    NetworkResult<T>(data)
8       class Error<T>(message: String, data: T? = null)
    : NetworkResult<T>(data, message)
9   }
```

MainActivity.kt

Tabel 22. Source Code MainActivity.kt

```
1   package com.pemrogramanmobile.apicompose
2
3   import android.os.Bundle
4   import androidx.activity.ComponentActivity
5   import androidx.activity.compose.setContent
6   import
    androidx.compose.foundation.layout.fillMaxSize
7   import androidx.compose.material3.MaterialTheme
8   import androidx.compose.material3.Surface
9   import androidx.compose.ui.Modifier
10  import
    com.pemrogramanmobile.apicompose.ui.AppNavigation
11  import
    com.pemrogramanmobile.apicompose.ui.theme.ApiCompose
    Theme
12
13  class MainActivity : ComponentActivity() {
```

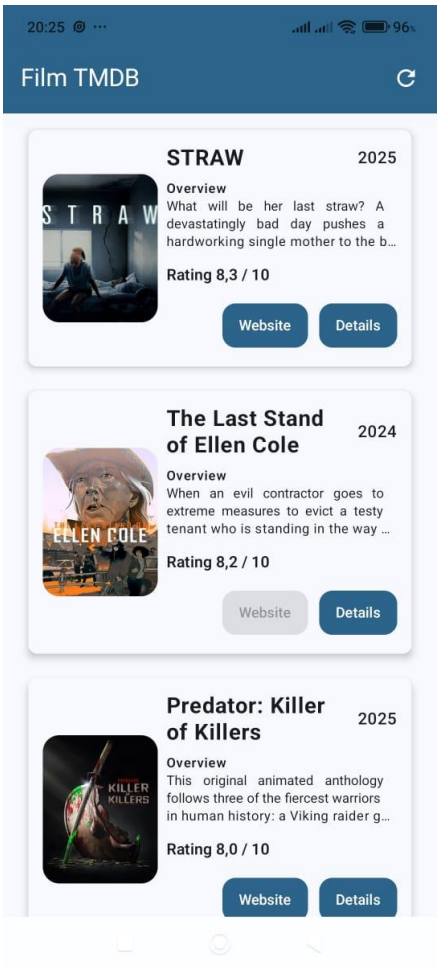
```
14      override fun onCreate(savedInstanceState:
        Bundle?) {
15          super.onCreate(savedInstanceState)
16          setContent {
17              ApiComposeTheme {
18                  Surface(
19                      modifier                    =
        Modifier.fillMaxSize(),
20                      color                       =
        MaterialTheme.colorScheme.background
21                  ) {
22                      AppNavigation()
23                  }
24              }
25          }
26      }
27  }
```

MyApplication.kt

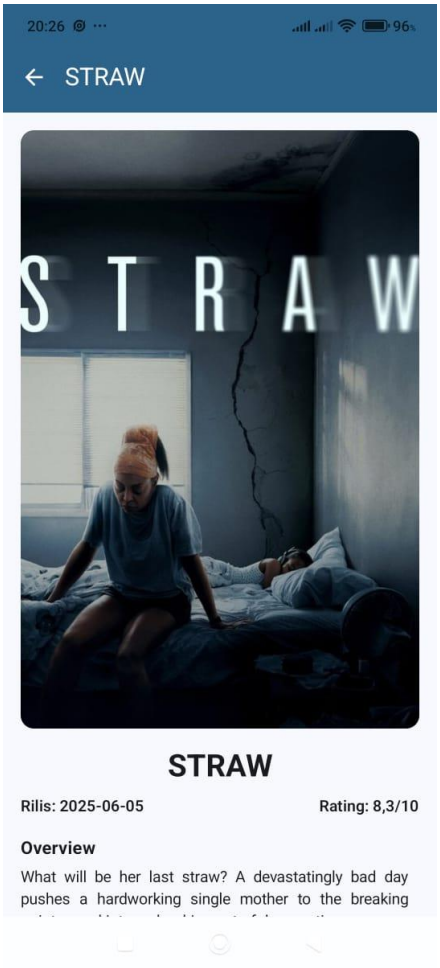
Tabel 23. Source Code MyApplication.kt

```
1   package com.pemrogramanmobile.apicompose
2
3   import android.app.Application
4   import
    com.pemrogramanmobile.apicompose.data.local.AppDatab
    ase
5
6   class MyApplication : Application() {
7       val database: AppDatabase by lazy {
    AppDatabase.getDatabase(this) }
8   }
```

B. Output Program



Gambar 1. Screenshot List Screen



Gambar 2. Screenshot Detail Screen

C. Pembahasan

/data/local/AppDatabase.kt

File ini mendefinisikan kelas AppDatabase yang merupakan inti dari implementasi Room. Kelas ini diannotasi dengan @Database yang memberitahu Room bahwa ini adalah konfigurasi database. entities = [MovieEntity::class] mendaftarkan tabel MovieEntity ke dalam database. version = 1 menandakan versi skema database, yang penting untuk migrasi (meskipun di sini fallbackToDestructiveMigration() digunakan, yang akan membuat ulang database jika versi berubah, menghapus data lama; ini praktis untuk pengembangan tetapi memerlukan strategi migrasi yang lebih hati-hati untuk produksi). exportSchema = false biasanya digunakan untuk menonaktifkan ekspor skema JSON yang bisa berguna untuk histori versi. AppDatabase adalah kelas abstrak yang

mewarisi RoomDatabase dan menyediakan akses ke DAO (Data Access Object) melalui fungsi abstrak movieDao(). Companion object-nya mengimplementasikan pola Singleton untuk memastikan hanya ada satu instance database (INSTANCE) di seluruh aplikasi, yang dibuat menggunakan Room.databaseBuilder(). Ini penting untuk efisiensi dan konsistensi data.

/data/local/MovieDao.kt

MovieDao (Data Access Object) adalah interface yang diannotasi dengan @Dao. Room akan mengimplementasikan metode-metode di dalamnya secara otomatis. File ini mendefinisikan operasi database untuk entitas MovieEntity.

- @Insert(onConflict = OnConflictStrategy.REPLACE): Menyisipkan daftar film. Jika ada film dengan PrimaryKey yang sama, film lama akan diganti. Ini berguna untuk menjaga data tetap up-to-date.
- @Query("SELECT * FROM movies ORDER BY voteAverage DESC") fun getAllMovies(): Flow<List<MovieEntity>>: Mengambil semua film dari tabel movies, diurutkan berdasarkan voteAverage secara menurun. Penggunaan Flow<List<MovieEntity>> sangat penting karena memungkinkan UI untuk secara reaktif mengamati perubahan data di database. Setiap kali data di tabel movies berubah, Flow ini akan memancarkan daftar film yang terbaru.
- @Query("SELECT * FROM movies WHERE id = :movieId") fun getMovieById(movieId: Int): Flow<MovieEntity?>: Mengambil satu film berdasarkan ID-nya, juga menggunakan Flow untuk pembaruan reaktif.
- @Query("DELETE FROM movies") suspend fun deleteAllMovies(): Menghapus semua data film dari tabel. Ini digunakan dalam caching strategy untuk membersihkan cache lama.
- @Query("SELECT COUNT(*) FROM movies") suspend fun getMovieCount(): Int: Mengambil jumlah film yang tersimpan. Ini juga bagian dari logika caching strategy.
- @Query("SELECT lastRefreshed FROM movies ORDER BY lastRefreshed DESC LIMIT 1") suspend fun getLastRefreshTimestamp(): Long?: Mengambil timestamp kapan terakhir kali data film di-refresh. Ini adalah komponen kunci dari caching

strategy yang diimplementasikan, untuk menentukan apakah cache masih valid atau sudah kedaluwarsa.

/data/local/MovieEntity.kt

File ini mendefinisikan MovieEntity, sebuah data class yang diannotasi dengan @Entity(tableName = "movies"). Ini merepresentasikan tabel movies dalam database Room.

- @PrimaryKey val id: Int: Menandakan bahwa id adalah kunci utama tabel.
- Kolom-kolom lain seperti title, overview, posterPath, releaseDate, voteAverage, dan homepage menyimpan detail film.
- val lastRefreshed: Long = System.currentTimeMillis(): Kolom ini sangat penting untuk caching strategy. Setiap kali entitas film disimpan atau diperbarui, kolom ini idealnya akan diisi dengan timestamp saat itu, yang kemudian digunakan oleh MovieDao dan MovieRepositoryImpl untuk menentukan usia cache.

/data/remote/MovieApiModel.kt

File ini berisi dua data class, MovieListResponse dan MovieApiModel, yang keduanya diannotasi dengan @Serializable. Ini menunjukkan penggunaan KotlinX Serialization sebagai library JSON untuk mengonversi respons JSON dari TMDB API menjadi objek Kotlin, dan sebaliknya jika diperlukan.

MovieListResponse: Merepresentasikan struktur respons JSON ketika meminta daftar film populer dari API. Ini mencakup informasi paginasi (page, totalPages, totalResults) dan daftar film (results yang merupakan List<MovieApiModel>).

MovieApiModel: Merepresentasikan satu objek film seperti yang diterima dari TMDB API. @SerialName("poster_path") digunakan untuk memetakan nama field JSON (misalnya poster_path) ke nama properti Kotlin yang berbeda (misalnya posterPath) jika ada perbedaan penamaan. Properti homepage juga disertakan, yang penting untuk detail film.

/data/remote/RetrofitClient.kt

RetrofitClient adalah object (Singleton) yang bertanggung jawab untuk mengkonfigurasi dan menyediakan instance Retrofit. Ini adalah inti dari implementasi networking library Retrofit.

- BASE_URL: URL dasar untuk TMDB API.
- API_KEY: Kunci API Anda untuk TMDB. Catatan: Sebaiknya kunci API tidak di-hardcode seperti ini dalam kode produksi; pertimbangkan untuk menyimpannya di gradle.properties atau menggunakan teknik lain yang lebih aman.
- Json { ignoreUnknownKeys = true; coerceInputValues = true }: Konfigurasi untuk KotlinX Serialization. ignoreUnknownKeys = true membuat parser lebih toleran terhadap field JSON tambahan yang mungkin tidak didefinisikan di MovieApiModel. coerceInputValues = true membantu menangani nilai default jika ada tipe data yang tidak cocok atau null dari API, namun harus digunakan dengan hati-hati.
- HttpLoggingInterceptor: Digunakan untuk mencatat request dan response HTTP, sangat berguna untuk debugging masalah jaringan. Levelnya diatur ke BODY untuk melihat detail penuh.
- OkHttpClient.Builder(): Mengkonfigurasi klien HTTP. Sebuah interceptor ditambahkan untuk secara otomatis menyisipkan api_key sebagai query parameter ke setiap request yang dikirim ke TMDB API.
- Retrofit.Builder(): Membangun instance Retrofit dengan baseUrl, httpClient yang sudah dikonfigurasi, dan yang penting, addConverterFactory(json.asConverterFactory("application/json".toMediaType())). Baris ini mengintegrasikan KotlinX Serialization dengan Retrofit sehingga Retrofit dapat secara otomatis mengonversi respons JSON menjadi objek Kotlin (MovieApiModel, MovieListResponse) dan sebaliknya.
- instance: TmdbApiService by lazy { ... }: Menyediakan instance TmdbApiService secara lazy, artinya instance Retrofit dan service baru akan dibuat saat pertama kali diakses.

/data/remote/TmdbApiService.kt

Ini adalah interface yang mendefinisikan endpoints TMDB API yang akan diakses oleh aplikasi menggunakan Retrofit.

- @GET("movie/popular") suspend fun getPopularMovies(...): Mendefinisikan request GET ke endpoint movie/popular untuk mendapatkan daftar film populer. @Query("page") menambahkan parameter halaman ke request. Fungsi ini adalah suspend fun karena operasi jaringan harus dijalankan secara asinkron menggunakan Kotlin Coroutines. Ia mengembalikan MovieListResponse.
- @GET("movie/{movie_id}") suspend fun getMovieDetails(...): Mendefinisikan request GET ke endpoint movie/{movie_id} untuk mendapatkan detail film tertentu. @Path("movie_id") menggantikan {movie_id} dalam URL dengan nilai parameter movieId. Fungsi ini juga suspend fun dan mengembalikan MovieApiModel.

/data/repository/MovieRepositoryImpl.kt

File ini adalah implementasi konkret dari MovieRepository. Ini adalah komponen kunci yang bertindak sebagai Single Source of Truth untuk data film, mengelola pengambilan data dari API (tmdbApiService) dan cache lokal (movieDao).

- Konstruktor menerima TmdbApiService, MovieDao, dan Context.
- CACHE_EXPIRY_MS = TimeUnit.HOURS.toMillis(1): Mendefinisikan durasi cache selama 1 jam. Ini adalah bagian dari caching strategy.
- getPopularMovies(): Flow<NetworkResult<List<Movie>>>:
  - Menggunakan Flow untuk memancarkan data secara asinkron.
  - Caching Strategy:
    1) Memeriksa timestamp refresh terakhir (lastRefreshTime) dari DAO dan jumlah film di cache (movieCount).
    2) Menghitung apakah cache sudah basi (isCacheStale) berdasarkan CACHE_EXPIRY_MS.
    3) Jika cache tidak kosong dan belum basi, data dari movieDao.getAllMovies() diambil, di-map ke List<Movie>, dan dipancarkan sebagai NetworkResult.Success.

4) Jika cache kosong atau sudah basi, maka:
5) Dilakukan pemanggilan API ke tmdbApiService.getPopularMovies().
6) Data dari API (hasil movieListResponse.results) diubah menjadi List<MovieEntity> menggunakan toMovieEntities(), yang juga menyematkan currentTime sebagai lastRefreshed.
7) Cache lama dihapus dengan movieDao.deleteAllMovies().
8) Data baru dari API disimpan ke Room dengan movieDao.insertMovies().
9) Data segar kemudian diambil dari movieDao.getAllMovies() dan dipancarkan sebagai NetworkResult.Success.
10) Error Handling: Jika terjadi exception saat mengambil data dari jaringan, ia akan mencoba memancarkan data yang mungkin masih ada di cache sebagai fallback (NetworkResult.Error dengan data fallback). Jika bahkan pengambilan dari database setelah refresh gagal, NetworkResult.Error juga dipancarkan.

- Penggunaan NetworkResult: Membungkus hasil operasi (sukses dengan data, atau error dengan pesan dan mungkin data lama/parsial) memberikan cara yang terstruktur untuk menangani status panggilan API dan operasi data di lapisan atas (ViewModel).

- getMovieDetails(movieId: Int): Flow<NetworkResult<Movie>>:
  - Pertama, mencoba mengambil data dari cache (movieDao.getMovieById(movieId)). Jika ada, data tersebut langsung dipancarkan sebagai NetworkResult.Success.
  - Kemudian, selalu mencoba mengambil detail film terbaru dari API (tmdbApiService.getMovieDetails(movieId)).
  - Data dari API diubah menjadi MovieEntity dan disimpan/diperbarui di Room menggunakan movieDao.insertMovies(listOf(movieEntity)). Pembaruan ini penting karena MovieEntity memiliki OnConflictStrategy.REPLACE, jadi data film akan diperbarui jika sudah ada.
  - Data terbaru dari API (yang sudah di-map ke Movie) dipancarkan sebagai NetworkResult.Success.

- Error Handling: Jika API gagal dan tidak ada data di cache, NetworkResult.Error tanpa data dipancarkan. Jika API gagal tetapi ada data di cache, NetworkResult.Error dipancarkan dengan pesan error dan data cache sebelumnya.
- Caching Strategy untuk Detail Film: Strategi di sini adalah cache-first, then network and update cache. Selalu mencoba menyajikan data dari cache terlebih dahulu untuk respons cepat, lalu mengambil data terbaru dari jaringan untuk menyegarkan cache dan UI. Properti homepage yang mungkin tidak ada di daftar populer akan diisi dari sini.

Penjelasan Caching Strategy yang digunakan:

Strategi caching yang digunakan adalah kombinasi dari:

- Time-based Expiry untuk Daftar Film Populer: Data daftar film populer dianggap stale (basi) setelah 1 jam (CACHE_EXPIRY_MS). Jika cache belum basi dan tidak kosong, data dari cache akan disajikan. Jika sudah basi atau cache kosong, data baru akan diambil dari jaringan, cache akan dihapus (deleteAllMovies) dan diisi ulang dengan data baru. Penghapusan semua film sebelum memasukkan yang baru adalah pendekatan sederhana untuk daftar "populer" yang mungkin berubah secara keseluruhan.
- Cache-First, then Network and Update untuk Detail Film: Untuk detail film, aplikasi pertama-tama mencoba memuat dari cache. Kemudian, ia selalu berusaha mengambil versi terbaru dari jaringan. Data yang berhasil diambil dari jaringan akan memperbarui entri di cache. Ini memastikan pengguna melihat data terbaru jika tersedia, sambil tetap memiliki data cache jika jaringan gagal setelah pemuatan awal.

Alasan penggunaan strategi ini:

- Keseimbangan Kinerja dan Keterkinian: Untuk daftar film populer, data tidak harus selalu real-time detik demi detik. Periode kadaluwarsa 1 jam memberikan keseimbangan yang baik antara mengurangi panggilan API berlebih dan menjaga data tetap cukup segar.
- Pengalaman Pengguna Offline/Lambat: Menyajikan data dari cache (jika ada) meningkatkan pengalaman pengguna saat koneksi internet lambat atau tidak tersedia.

- Efisiensi Sumber Daya: Mengurangi jumlah panggilan jaringan menghemat baterai dan penggunaan data pengguna.
- Pembaruan Detail Spesifik: Untuk detail film, penting untuk mendapatkan informasi selengkap mungkin (termasuk homepage), sehingga pemanggilan jaringan setelah pemuatan cache dibenarkan untuk menyegarkan dan melengkapi data.

/data/Mappers.kt

File ini berisi serangkaian fungsi ekstensi untuk memetakan (mengubah) objek dari satu layer/model ke layer/model lain. Ini adalah praktik yang baik untuk menjaga pemisahan antar layer.

- MovieApiModel.toDomainMovie(): Mengubah objek API menjadi objek Domain (Movie).
- MovieApiModel.toMovieEntity(lastRefreshedTime: Long): Mengubah objek API menjadi objek Entity Room (MovieEntity), dengan menyertakan lastRefreshedTime yang penting untuk caching.
- MovieEntity.toDomainMovie(): Mengubah objek Entity Room menjadi objek Domain.
- Fungsi-fungsi untuk mengubah List dari satu tipe ke tipe lain (misalnya, List<MovieApiModel>.toDomainMovies()).

/domain/model/Movie.kt

Ini adalah data class Movie yang merepresentasikan entitas film di domain layer. Model ini yang akan digunakan oleh UI dan Use Case. Model ini lebih sederhana dan hanya berisi data yang relevan untuk logika bisnis dan presentasi, terlepas dari bagaimana data itu disimpan atau darimana asalnya. Field homepage juga ada di sini.

/domain/repository/MovieRepository.kt

Ini adalah interface yang mendefinisikan kontrak untuk MovieRepositoryImpl. Ini adalah bagian dari prinsip Dependency Inversion, di mana domain layer mendefinisikan bagaimana

ia ingin berinteraksi dengan data, tanpa mengetahui detail implementasi (apakah data berasal dari API, database, atau keduanya).

- Metode-metodenya (getPopularMovies, getMovieDetails) mengembalikan Flow<NetworkResult<...>>, yang menunjukkan bahwa mereka menyediakan aliran data asinkron yang juga mencakup informasi status (sukses/error).

/domain/usecase/GetMovieDetailsUseCase.kt
dan /domain/usecase/GetPopularMoviesUseCase.kt

Kedua file ini mendefinisikan use case (atau interactor). Use case merangkum satu unit logika bisnis atau fungsionalitas aplikasi.

- Masing-masing mengambil MovieRepository sebagai dependensi.
- Menggunakan operator fun invoke memungkinkan instance use case dipanggil seolah-olah itu adalah fungsi. Contoh: getPopularMoviesUseCase().
- Mereka hanya mendelegasikan panggilan ke metode yang sesuai di movieRepository. Dalam aplikasi yang lebih kompleks, use case mungkin berisi logika tambahan.

/ui/screen/moviedetail/MovieDetailScreen.kt

Ini adalah Composable yang bertanggung jawab untuk menampilkan detail satu film.

- Menggunakan viewModel(factory = MovieDetailViewModel.Factory) untuk mendapatkan instance MovieDetailViewModel.
- uiState by viewModel.uiState.collectAsStateWithLifecycle(): Mengamati StateFlow dari ViewModel untuk memperbarui UI secara reaktif.
- SnackbarHostState dan LaunchedEffect digunakan untuk menampilkan pesan Snackbar dari ViewModel.
- Scaffold dengan TopAppBar yang menampilkan judul film dan tombol kembali.
- Logika kondisional (when) untuk menampilkan CircularProgressIndicator saat isLoading, pesan error jika uiState.error != null, konten detail jika uiState.movie != null, atau pesan default jika tidak ada data.
- MovieDetailContent: Composable terpisah untuk menampilkan konten detail:

- AsyncImage (dari library Coil) digunakan untuk memuat dan menampilkan gambar poster film dari URL (https://image.tmdb.org/t/p/w500${movie.posterPath}). Ini menangani pemuatan gambar secara asinkron, caching gambar, dan menampilkan placeholder atau gambar error.
- Menampilkan judul, tanggal rilis, rating, overview.
- Tombol "Kunjungi Website" akan muncul jika movie.homepage ada dan tidak kosong. Tombol ini menggunakan Intent(Intent.ACTION_VIEW, Uri.parse(movie.homepage)) untuk membuka URL di browser. Ada logika fallback jika Chrome tidak terinstal.

/ui/screen/movielist/MovieItem.kt

Composable ini mendefinisikan tampilan untuk satu item film dalam daftar.

- Menggunakan Card untuk tampilan item.
- AsyncImage (dari Coil) digunakan untuk memuat poster film, dengan ukuran yang lebih kecil (w342) dibandingkan layar detail.
- Menampilkan judul, tahun rilis (diekstrak dari releaseDate), overview singkat (dengan maxLines = 3), dan rating.
- Terdapat dua tombol:
  - "Website": Mirip dengan di MovieDetailScreen, membuka movie.homepage jika tersedia.
  - "Details": Memanggil onDetailsClick(movie.id) yang akan menavigasi ke layar detail film.

/ui/screen/movielist/MovieListScreen.kt

Composable utama untuk menampilkan daftar film populer.

- Menginisialisasi MovieRepositoryImpl dan GetPopularMoviesUseCase. Catatan: Idealnya, dependensi seperti ini akan di-provide menggunakan library Dependency Injection seperti Hilt atau Koin, daripada dibuat secara manual di Composable. Namun, untuk proyek yang lebih kecil, pendekatan ini masih bisa diterima.

- Menggunakan viewModel(factory = MovieViewModelFactory(getPopularMoviesUseCase)) untuk mendapatkan MovieViewModel.
- Mengamati uiState dari MovieViewModel.
- Scaffold dengan TopAppBar yang menampilkan judul dan ikon refresh (IconButton yang memanggil movieViewModel.refreshMovies()).
- Logika kondisional untuk menampilkan:
  - CircularProgressIndicator jika isLoading dan daftar film kosong.
  - Pesan error dan tombol "Coba Lagi" jika ada error dan daftar kosong.
  - LazyColumn untuk menampilkan daftar uiState.movies menggunakan MovieItem jika daftar tidak kosong. key = { movie -> movie.id } penting untuk performa LazyColumn.
  - Pesan "Tidak ada film yang ditemukan" jika tidak loading, tidak ada error, dan daftar kosong.
  - LinearProgressIndicator di bagian atas jika isLoading tetapi daftar film sudah ada (menandakan refresh di latar belakang).
- onDetailsClick pada MovieItem menavigasi ke rute "movieDetail/$movieId" menggunakan navController.
- Terdapat Composable Preview (MovieListScreenLightPreview dan MovieListScreenDarkPreview) untuk memudahkan pengembangan UI.

/ui/viewmodel/MovieDetailViewModel.kt

ViewModel untuk MovieDetailScreen.

- MovieDetailUiState: Data class yang merepresentasikan state UI untuk layar detail (data film, status loading, error, pesan snackbar).
- Konstruktor menerima GetMovieDetailsUseCase dan SavedStateHandle. SavedStateHandle digunakan untuk mengambil movieId yang diteruskan melalui argumen navigasi.
- _uiState adalah MutableStateFlow (privat) dan uiState adalah StateFlow (publik, read-only) yang diekspos ke UI.

- init block: Memeriksa apakah movieId valid. Jika ya, panggil fetchMovieDetails(). Jika tidak, set state error.
- fetchMovieDetails(id: Int):
  - Memperbarui _uiState untuk menandakan isLoading = true.
  - Memanggil getMovieDetailsUseCase(id) yang mengembalikan Flow<NetworkResult<Movie>>.
  - Menggunakan .onEach { result -> ... } untuk memproses setiap emisi dari Flow:
    - Jika NetworkResult.Success, perbarui state dengan data film dan set isLoading = false.
    - Jika NetworkResult.Error, perbarui state. Jika sudah ada data film sebelumnya (misalnya dari cache), tampilkan pesan error sebagai snackbarMessage. Jika belum ada data film, tampilkan sebagai error utama di layar.
  - .launchIn(viewModelScope) meluncurkan koleksi Flow dalam viewModelScope, yang akan otomatis dibatalkan saat ViewModel dihancurkan.
- clearSnackbarMessage(): Untuk membersihkan pesan snackbar setelah ditampilkan.
- retryFetch(): Untuk mencoba lagi mengambil data jika terjadi error.
- Companion object Factory: Implementasi ViewModelProvider.Factory untuk membuat instance MovieDetailViewModel dengan dependensi yang diperlukan (GetMovieDetailsUseCase yang dibuat dengan MovieRepositoryImpl). application.database.movieDao() diambil dari MyApplication.

/ui/viewmodel/MovieViewModel.kt

ViewModel untuk MovieListScreen.

- MovieListUiState: Data class untuk state UI daftar film.
- Konstruktor menerima GetPopularMoviesUseCase.
- Sama seperti MovieDetailViewModel, menggunakan MutableStateFlow dan StateFlow untuk uiState.
- init block: Memanggil fetchPopularMovies() saat ViewModel dibuat.
- fetchPopularMovies(forceRefresh: Boolean = false):
  - Jika forceRefresh (misalnya dari tombol refresh) atau daftar film masih kosong, set isLoading = true.

    - Memanggil getPopularMoviesUseCase() dan memproses NetworkResult:
        - NetworkResult.Success: Perbarui state dengan daftar film.
        - NetworkResult.Error: Sama seperti di MovieDetailViewModel, tampilkan error sebagai snackbarMessage jika sudah ada data, atau sebagai error utama jika belum ada.
- clearSnackbarMessage() dan refreshMovies() (yang memanggil fetchPopularMovies(forceRefresh = true)).
- MovieViewModelFactory: Factory sederhana untuk membuat MovieViewModel dengan GetPopularMoviesUseCase.

/ui/AppNavigation.kt

Mengkonfigurasi navigasi dalam aplikasi menggunakan Jetpack Navigation Compose.

- rememberNavController(): Membuat dan mengingat NavController.
- NavHost: Container untuk tujuan navigasi. startDestination = "movieList" menentukan layar awal.
- composable("movieList") { ... }: Mendefinisikan rute untuk MovieListScreen.
- composable(route = "movieDetail/{movieId}", arguments = listOf(navArgument("movieId") { type = NavType.StringType })) { ... }: Mendefinisikan rute untuk MovieDetailScreen. {movieId} adalah argumen yang akan diteruskan. MovieDetailViewModel akan mengambilnya menggunakan SavedStateHandle.

/ui/Routes.kt

File ini mendefinisikan sebuah object bernama Routes. Dalam Kotlin, object adalah cara untuk mendeklarasikan sebuah Singleton, yang berarti hanya akan ada satu instance dari kelas Routes di seluruh aplikasi. Pendekatan ini sangat ideal untuk mengelola konstanta dan utilitas yang bersifat global, seperti definisi rute navigasi.

Tujuan utama dari file ini adalah untuk **sentralisasi dan standardisasi rute navigasi** yang digunakan oleh Jetpack Navigation. Ini adalah praktik terbaik yang sangat dianjurkan untuk menghindari apa yang disebut sebagai *"magic strings"* (string literal yang tersebar di banyak file kode).

Di dalam object Routes, terdapat beberapa komponen:

- const val MOVIE_LIST = "movieList": Mendefinisikan rute untuk layar daftar film sebagai sebuah konstanta. Dengan menggunakan Routes.MOVIE_LIST di seluruh aplikasi (misalnya di AppNavigation.kt), Anda memastikan konsistensi dan mengurangi risiko kesalahan ketik. Jika suatu saat Anda perlu mengubah nama rute ini, Anda hanya perlu mengubahnya di satu tempat ini.
- const val MOVIE_DETAIL = "movieDetail/{movieId}": Mendefinisikan rute untuk layar detail film. Bagian {movieId} adalah *placeholder* untuk argumen yang akan dikirimkan, dalam hal ini adalah ID dari film yang akan ditampilkan. Mendefinisikan struktur rute ini sebagai konstanta juga memastikan konsistensi.
- fun navigateById(id: Int): String: Ini adalah fungsi utilitas yang sangat berguna. Fungsi ini mengambil sebuah id film sebagai parameter Int dan mengembalikan String rute yang sudah lengkap dan siap digunakan untuk navigasi.

/util/AppSettings.kt

File ini menyediakan utilitas untuk bekerja dengan SharedPreferences, yang cocok untuk menyimpan data preferensi atau pengaturan sederhana.

- PREFS_NAME: Nama file SharedPreferences.
- getPrefs(context: Context): Fungsi helper privat untuk mendapatkan instance SharedPreferences.
- getLastMoviesCacheTimestamp dan setLastMoviesCacheTimestamp: Metode untuk menyimpan dan mengambil timestamp cache film. Perlu dicatat bahwa dalam MovieRepositoryImpl yang Anda berikan, timestamp untuk caching data film sebenarnya diambil dari dan disimpan di dalam entitas MovieEntity di Room (lastRefreshed). Fungsi di AppSettings ini mungkin dimaksudkan untuk skenario caching yang berbeda atau sebagai alternatif, tetapi tidak secara aktif digunakan oleh logika caching MovieRepositoryImpl untuk film.
- getTheme dan setTheme: Contoh bagaimana SharedPreferences dapat digunakan untuk menyimpan pengaturan tema UI (misalnya "system", "light", "dark").

/util/NetworkResult.kt

Ini adalah implementasi dari **generic response** untuk menangani status dan hasil dari operasi jaringan (atau operasi asinkron lainnya yang bisa gagal).

- sealed class NetworkResult<T>(val data: T? = null, val message: String? = null): Kelas dasar dengan properti untuk data (jika sukses) dan pesan (jika error).
- class Success<T>(data: T) : NetworkResult<T>(data): Merepresentasikan operasi yang sukses, membawa data hasil.
- class Error<T>(message: String, data: T? = null) : NetworkResult<T>(data, message): Merepresentasikan operasi yang gagal, membawa pesan error dan opsional data lama/parsial (berguna untuk menampilkan data cache saat jaringan gagal).

/MainActivity.kt

Ini adalah *Activity* utama dan titik masuk aplikasi Anda.

- setContent { ... }: Mengatur konten UI menggunakan Jetpack Compose.
- ApiComposeTheme { ... }: Menerapkan tema kustom aplikasi Anda.
- Surface { AppNavigation() }: Surface adalah container dasar di Compose, dan AppNavigation() dipanggil untuk mengatur dan menampilkan layar awal (dan navigasi selanjutnya).

/MyApplication.kt

Kelas MyApplication yang mewarisi android.app.Application.

- val database: AppDatabase by lazy { AppDatabase.getDatabase(this) }: Membuat instance AppDatabase secara *lazy* dan menyediakannya sebagai properti publik. Ini adalah cara sederhana untuk menyediakan instance database secara global di aplikasi. Instance ini kemudian diakses oleh ViewModel factories.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.
https://github.com/raihan2030/Praktikum-Pemrograman-Mobile/tree/main/Praktikum-5