LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

CONNECT TO THE INTERNET

Oleh:

Alysa Armelia **NIM. 2310817120009**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN I
MODUL 5

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

Nama Praktikan : Alysa Armelia
NIM : 2310817120009

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Zulfa Auliya Akbar NIM. 2210817210026	Muti`a Maulida S.Kom M.T.I NIP. 19881027 201903 20 13

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

Soal Praktikum:

1. Lanjutkan aplikasi Android yang sudah dibuat pada Modul 4 dengan menambahkan modifikasi sesuai ketentuan berikut:
    a. Gunakan networking library seperti Retrofit atau Ktor agar aplikasi dapat mengambil data dari remote API. Dalam penggunaan networking library, sertakan generic response untuk status dan error handling pada API dan Flow untuk data stream.
    b. Gunakan KotlinX Serialization sebagai library JSON.
    c. Gunakan library seperti Coil atau Glide untuk image loading.
    d. API yang digunakan pada modul ini bebas, contoh API gratis 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 (misalnya offline-first app, pengaturan dark/light mode, fitur favorite, dll)
    f. Gunakan caching strategy pada Room..
    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

1. MyApiResponse.kt

```
1   package com.example.myapi_test.data.api
2
3   // DTOs that exactly match the JSON response from the API.
4
5   data class BookApiResponse(
6       val data: List<BookData>
7   )
8
9   data class BookData(
10      val attributes: BookAttributes
11  )
12
13  data class BookAttributes(
```

```
    val author: String,
    val cover: String,
    val release_date: String,
    val title: String,
    val wiki: String,
    val summary: String
)
```

Tabel 1. 1 Source Code MyApiResponse

2. MyInstance

```
package com.example.myapi_test.data.api

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object MyInstance {
    private const val BASE_URL = "https://api.potterdb.com/"

    val api: MyService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(MyService::class.java)
    }
}
```

Tabel 1. 2 Source Code MyApiResponse

3. MyService

```
package com.example.myapi_test.data.api

import retrofit2.http.GET

interface MyService {
    @GET("v1/books")
    suspend fun getMessage(): BookApiResponse
}
```

Tabel 1. 3 Source Code MyService

4. BookDao

```
package com.example.myapi_test.data.database.dao

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
```

```
6   import androidx.room.Query
7   import
    com.example.myapi_test.data.database.entity.BookDbEntity
8
9   @Dao
10  interface BookDao {
11      /**
12       * Inserts a list of books into the database. If a book
    with the same
13       * primary key already exists, it will be replaced.
14       */
15      @Insert(onConflict = OnConflictStrategy.REPLACE)
16      suspend fun insertBooks(books: List<BookDbEntity>)
17
18      /**
19       * Retrieves all books from the database, ordered by
    title.
20       * @return A list of all BookDbEntity objects.
21       */
22      @Query("SELECT * FROM books ORDER BY title ASC")
23      suspend fun getAllBooks(): List<BookDbEntity>
24
25      /**
26       * Deletes all books from the database.
27       */
28      @Query("DELETE FROM books")
29      suspend fun clearAllBooks()
30  }
```

Tabel 1. 4 Source Code BookDao

5. BookDbEntity

```
1   package com.example.myapi_test.data.database.entity
2
3   import androidx.room.Entity
4   import androidx.room.PrimaryKey
5
6   /**
7    * Defines the schema for the 'books' table in the Room
    database.
8    * Each instance of this class represents a row in the
    table.
9    */
10  @Entity(tableName = "books")
11  data class BookDbEntity(
12      // We use the title as the primary key, assuming it's
    unique.
13      // For real-world apps, a unique ID from the API is
    preferable.
14      @PrimaryKey
15      val title: String,
```

```
16      val author: String,
17      val cover: String,
18      val releaseDate: String,
19      val summary: String,
20      val wiki: String
21  )
```

Tabel 1. 5 Source Code BookDbEntity

6. AppDatabase

```
1   package com.example.myapi_test.data.database
2
3   import android.content.Context
4   import androidx.room.Database
5   import androidx.room.Room
6   import androidx.room.RoomDatabase
7   import com.example.myapi_test.data.database.dao.BookDao
8   import
    com.example.myapi_test.data.database.entity.BookDbEntity
9
10  @Database(entities = [BookDbEntity::class], version = 1,
    exportSchema = false)
11  abstract class AppDatabase : RoomDatabase() {
12
13      abstract fun bookDao(): BookDao
14
15      companion object {
16          // Volatile ensures that the INSTANCE is always up-
    to-date and the same for all execution threads.
17          @Volatile
18          private var INSTANCE: AppDatabase? = null
19
20          fun getDatabase(context: Context): AppDatabase {
21              // Return the existing instance if it exists,
    otherwise create a new one.
22              return INSTANCE ?: synchronized(this) {
23                  val instance = Room.databaseBuilder(
24                      context.applicationContext,
25                      AppDatabase::class.java,
26                      "book_database"
27                  )
28                      .fallbackToDestructiveMigration() //
    Strategy for handling version changes
29                      .build()
30                  INSTANCE = instance
31                  instance
32              }
33          }
34      }
35  }
```

Tabel 1. 6 Source Code AppDatabase

7. BookMapper.kt

```
1   package com.example.myapi_test.data.mappers
2
3   import com.example.myapi_test.data.api.BookData
4   import com.example.myapi_test.domain.model.Book
5   import com.example.myapi_test.presentation.model.BookUi
6   import com.example.myapi_test.data.api.BookAttributes
7   import
    com.example.myapi_test.data.database.entity.BookDbEntity
8   /**
9    * Converts the data layer's BookData (DTO) into a domain
    layer Book entity.
10   * This is a key part of separating the data and domain
    layers.
11   */
12  fun BookData.toDomain(): Book {
13      return Book(
14          title = this.attributes.title,
15          author = this.attributes.author,
16          cover = this.attributes.cover,
17          releaseDate = this.attributes.release_date,
18          summary = this.attributes.summary,
19          wiki = this.attributes.wiki
20      )
21  }
22  fun Book.toUiModel(): BookUi {
23      return BookUi(
24          title = this.title,
25          author = this.author,
26          cover = this.cover,
27          release_date = this.releaseDate,
28          summary = this.summary,
29          wiki = this.wiki
30      )
31  }
32  /**
33   * Converts the network DTO (BookAttributes) to a database
    entity (BookDbEntity).
34   */
35  fun BookAttributes.toDbEntity(): BookDbEntity {
36      return BookDbEntity(
37          title = this.title,
38          author = this.author,
39          cover = this.cover,
40          releaseDate = this.release_date,
41          summary = this.summary,
42          wiki = this.wiki
43      )
44  }
45
46  /**
```

```
 * Converts a database entity (BookDbEntity) to a domain model (Book).
 */
fun BookDbEntity.toDomain(): Book {
    return Book(
        title = this.title,
        author = this.author,
        cover = this.cover,
        releaseDate = this.releaseDate,
        summary = this.summary,
        wiki = this.wiki
    )
}
```

Tabel 1. 7 Source Code BookMapper

8. BookRepositoryImpl

```
package com.example.myapi_test.data.repository

import android.content.Context
import android.util.Log
import com.example.myapi_test.data.api.MyInstance
import com.example.myapi_test.data.database.AppDatabase
import com.example.myapi_test.data.mappers.toDbEntity
import com.example.myapi_test.data.mappers.toDomain
import com.example.myapi_test.domain.model.Book
import com.example.myapi_test.domain.repository.BookRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext

/**
 * Implements the BookRepository. It now manages two data sources:
 * 1. Remote: MyInstance (Retrofit API)
 * 2. Local: AppDatabase (Room DAO)
 */
class BookRepositoryImpl(context: Context) : BookRepository {

    private val apiService = MyInstance.api
    private val bookDao = AppDatabase.getDatabase(context).bookDao()

    override suspend fun getBooks(): Result<List<Book>> {
        return withContext(Dispatchers.IO) {
            try {
                // 1. Fetch fresh data from the API
                Log.d("BookRepository", "Fetching books from API...")
                val remoteBooks =
```

```
    apiService.getMessage().data
30
31                  // 2. Clear old data from the database
32                  bookDao.clearAllBooks()
33
34                  // 3. Map network DTOs to database entities
    and insert them
35                  val dbEntities = remoteBooks.map {
    it.attributes.toDbEntity() }
36                  bookDao.insertBooks(dbEntities)
37                  Log.d("BookRepository", "Successfully
    inserted ${dbEntities.size} books into database.")
38
39              } catch (e: Exception) {
40                  // 4. If the network call fails, log the
    error.
41                  // The function will proceed to load data
    from the cache.
42                  Log.e("BookRepository", "Failed to fetch
    from API. Loading from cache.", e)
43              }
44
45              // 5. Always return data from the database
    (Single Source of Truth)
46              // If the network call succeeded, this is the
    fresh data.
47              // If it failed, this is the old, cached data.
48              try {
49                  val cachedBooks = bookDao.getAllBooks()
50                  Log.d("BookRepository", "Loaded
    ${cachedBooks.size} books from database.")
51                  Result.success(cachedBooks.map {
    it.toDomain() })
52              } catch (e: Exception) {
53                  Log.e("BookRepository", "Failed to read from
    database.", e)
54                  Result.failure(e)
55              }
56          }
57      }
58  }
```

Tabel 1. 8 Source Code BookRepositoryImpl

9. Book

```
1   package com.example.myapi_test.domain.model
2
3   /**
4    * Represents the core business object (Entity). It is a
    pure data class,
5    * independent of any data source or UI implementation
```

```
    details.
6    */
7   data class Book(
8       val title: String,
9       val author: String,
10      val cover: String,
11      val releaseDate: String,
12      val summary: String,
13      val wiki: String
14  )
```

Tabel 1. 9 Source Code Book

10. BookRepository

```
1   package com.example.myapi_test.domain.repository
2
3   import com.example.myapi_test.domain.model.Book
4
5   /**
6    * Defines a contract for data operations that the Data
    layer must implement.
7    * The domain layer uses this interface to access data,
    keeping it decoupled
8    * from the data source's implementation (e.g., Retrofit,
    Room).
9    */
10  interface BookRepository {
11      /**
12       * Fetches a list of books from the data source.
13       * @return A Result wrapper containing either the list
    of books on success or an exception on failure.
14       */
15      suspend fun getBooks(): Result<List<Book>>
16  }
```

Tabel 1. 10 Source Code BookRepository

11. GetBooksUseCase

```
1   package com.example.myapi_test.domain.usecase
2
3   import com.example.myapi_test.domain.model.Book
4   import
    com.example.myapi_test.domain.repository.BookRepository
5
6   /**
7    * A Use Case that encapsulates the business logic for
    fetching the list of books.
8    * It depends on the BookRepository interface, not its
    implementation.
```

```
 */
class GetBooksUseCase(private val bookRepository:
BookRepository) {

    /**
     * Executes the use case. The 'invoke' operator allows
this class to be called
     * like a function, e.g., getBooksUseCase().
     */
    suspend operator fun invoke(): Result<List<Book>> {
        return bookRepository.getBooks()
    }
}
```

Tabel 1. 11 Source Code GetBooksUseCase

12. BookUi

```
package com.example.myapi_test.presentation.model

import android.os.Parcelable
import kotlinx.parcelize.Parcelize

/**
 * A data model specifically for the UI (Presentation
Layer).
 * It implements Parcelable to be passed between Android
components like Fragments.
 */
@Parcelize
data class BookUi(
    val author: String,
    val cover: String,
    val release_date: String,
    val title: String,
    val wiki: String,
    val summary: String
) : Parcelable
```

Tabel 1. 12 Source Code BookUi

13. DetailFragment

```
package com.example.myapi_test.presentation.ui

import android.os.Build
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.app.AppCompatActivity
```

```
9  import androidx.fragment.app.Fragment
10 import com.bumptech.glide.Glide
11 import com.example.myapi_test.R
12 import
   com.example.myapi_test.databinding.DetailFragmentBinding
13 import com.example.myapi_test.presentation.model.BookUi
14
15 class DetailFragment : Fragment() {
16
17     private var _binding: DetailFragmentBinding? = null
18     private val binding get() = _binding!!
19
20     private var currentBook: BookUi? = null
21
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         arguments?.let {
25             currentBook = if (Build.VERSION.SDK_INT >=
   Build.VERSION_CODES.TIRAMISU) {
26                 it.getParcelable(ARG_BOOK,
   BookUi::class.java)
27             } else {
28                 @Suppress("DEPRECATION")
29                 it.getParcelable(ARG_BOOK)
30             }
31         }
32     }
33
34     override fun onCreateView(
35         inflater: LayoutInflater, container: ViewGroup?,
36         savedInstanceState: Bundle?
37     ): View {
38         _binding = DetailFragmentBinding.inflate(inflater,
   container, false)
39         return binding.root
40     }
41
42     override fun onViewCreated(view: View,
   savedInstanceState: Bundle?) {
43         super.onViewCreated(view, savedInstanceState)
44
45         val toolbar: androidx.appcompat.widget.Toolbar =
   view.findViewById(R.id.detailToolbar)
46         val activity = requireActivity() as
   AppCompatActivity
47         activity.setSupportActionBar(toolbar)
48 activity.supportActionBar?.setDisplayHomeAsUpEnabled(true)
49 activity.supportActionBar?.setDisplayShowHomeEnabled(true)
50
51         toolbar.setNavigationOnClickListener {
52             // Handle back press
53             activity.onBackPressed()
```

```
54          }
55          currentBook?.let { book ->
56              binding.detailTitle.text = book.title
57              binding.detailDescription.text = book.summary
58
59              if (book.cover.isNotBlank()) {
60                  Glide.with(this)
61                      .load(book.cover)
62                      .into(binding.detailImage)
63              } else {
64  binding.detailImage.setImageResource(R.drawable.ic_launcher
    _background)
65              }
66          }
67      }
68
69      override fun onDestroyView() {
70          super.onDestroyView()
71          _binding = null
72      }
73
74      companion object {
75          private const val ARG_BOOK = "book_ui_parcel"
76
77          @JvmStatic
78          fun newInstance(book: BookUi): DetailFragment {
79              return DetailFragment().apply {
80                  arguments = Bundle().apply {
81                      putParcelable(ARG_BOOK, book)
82                  }
83              }
84          }
85      }
86  }
```

Tabel 1. 13 Source Code DetailFragment

14. HomeFragment

```
1   package com.example.myapi_test.presentation.ui
2
3   import android.content.Intent
4   import android.net.Uri
5   import android.os.Bundle
6   import android.util.Log
7   import android.view.LayoutInflater
8   import android.view.View
9   import android.view.ViewGroup
10  import android.widget.Toast
11  import androidx.fragment.app.Fragment
12  import androidx.lifecycle.ViewModelProvider
13  import androidx.recyclerview.widget.LinearLayoutManager
```

```
14  import com.example.myapi_test.R
15  import
    com.example.myapi_test.databinding.HomeFragmentBinding
16  import com.example.myapi_test.presentation.model.BookUi
17  import
    com.example.myapi_test.presentation.viewmodel.BookViewModel
18
19  class HomeFragment : Fragment() {
20
21      private var _binding: HomeFragmentBinding? = null
22      private val binding get() = _binding!!
23
24      private lateinit var bookViewModel: BookViewModel
25      private lateinit var myAdapter: MyAdapter
26
27      override fun onCreateView(
28          inflater: LayoutInflater, container: ViewGroup?,
29          savedInstanceState: Bundle?
30      ): View {
31          _binding = HomeFragmentBinding.inflate(inflater,
    container, false)
32          return binding.root
33      }
34
35      override fun onViewCreated(view: View,
    savedInstanceState: Bundle?) {
36          super.onViewCreated(view, savedInstanceState)
37
38          // Share the ViewModel with the hosting Activity
39          bookViewModel =
    ViewModelProvider(requireActivity()).get(BookViewModel::cla
    ss.java)
40
41          setupRecyclerView()
42          observeViewModel()
43
44          // Fetch data only if the list is empty
45          if
    (bookViewModel.booksLiveData.value.isNullOrEmpty()) {
46              bookViewModel.fetchBooks()
47          }
48      }
49
50      private fun setupRecyclerView() {
51          myAdapter = MyAdapter(
52              mutableListOf(),
53              onDetailButtonClicked = { book ->
    navigateToDetail(book) },
54              onInfoButtonClicked = { book ->
    openWikiLink(book) },
55              onItemRootClicked = { book ->
    navigateToDetail(book) }
```

```
56          )
57
58          binding.rvCharacter.apply {
59              layoutManager =
    LinearLayoutManager(requireContext())
60              adapter = myAdapter
61          }
62      }
63
64      private fun observeViewModel() {
65          // Observe the list of books and update the adapter
66      bookViewModel.booksLiveData.observe(viewLifecycleOwner) {
    books ->
67              books?.let { myAdapter.updateBooks(it) }
68          }
69
70          // The MainActivity handles observing isLoading and
    errorMessage for global UI feedback
71      }
72
73      private fun navigateToDetail(book: BookUi) {
74          Log.d("HomeFragment", "Navigating to detail for:
    ${book.title}")
75          val detailFragment =
    DetailFragment.newInstance(book)
76          parentFragmentManager.beginTransaction()
77              .replace(R.id.fragmentContainer,
    detailFragment)
78              .addToBackStack(null)
79              .commit()
80      }
81
82      private fun openWikiLink(book: BookUi) {
83          Log.d("HomeFragment", "Info button clicked for:
    ${book.title}")
84          if (book.wiki.isNotBlank()) {
85              try {
86                  val intent = Intent(Intent.ACTION_VIEW,
    Uri.parse(book.wiki))
87                  startActivity(intent)
88              } catch (e: Exception) {
89                  Log.e("HomeFragment", "Could not open URL:
    ${book.wiki}", e)
90                  Toast.makeText(requireContext(), "Could not
    open link.", Toast.LENGTH_SHORT).show()
91              }
92          } else {
93              Toast.makeText(requireContext(), "No info link
    available.", Toast.LENGTH_SHORT).show()
94          }
95      }
96
```

```
97      override fun onDestroyView() {
98          super.onDestroyView()
99          _binding = null
100     }
101 }
```

Tabel 1. 14 Source Code HomeFragment

15. MainActivity

```
1   package com.example.myapi_test.presentation.ui
2
3   import android.os.Bundle
4   import android.view.View
5   import android.widget.Toast
6   import androidx.appcompat.app.AppCompatActivity
7   import
    androidx.core.splashscreen.SplashScreen.Companion.installSp
    lashScreen // <-- Import this
8   import androidx.lifecycle.ViewModelProvider
9   import com.example.myapi_test.R
10  import
    com.example.myapi_test.databinding.ActivityMainBinding
11  import
    com.example.myapi_test.presentation.viewmodel.BookViewModel
12  import
    com.example.myapi_test.presentation.viewmodel.BookViewModel
    Factory
13
14  class MainActivity : AppCompatActivity() {
15
16      private lateinit var binding: ActivityMainBinding
17      private lateinit var bookViewModel: BookViewModel
18
19      override fun onCreate(savedInstanceState: Bundle?) {
20          // 1. Handle the splash screen transition. Must be
    called before super.onCreate().
21          val splashScreen = installSplashScreen()
22          super.onCreate(savedInstanceState)
23
24          // Standard view binding setup
25          binding =
    ActivityMainBinding.inflate(layoutInflater)
26          setContentView(binding.root)
27
28          // Initialize ViewModel using the factory
29          val viewModelFactory =
    BookViewModelFactory(application)
30          bookViewModel = ViewModelProvider(this,
    viewModelFactory)[BookViewModel::class.java]
31
32          // 2. Keep the splash screen visible until the
```

```
    initial data is ready.
33          // This links the splash screen's duration to your
    app's actual loading state.
34          splashScreen.setKeepOnScreenCondition {
35              bookViewModel.isLoadingLiveData.value == true
36          }
37
38          // Add the main fragment only if the activity is
    newly created
39          if (savedInstanceState == null) {
40              supportFragmentManager.beginTransaction()
41                  .replace(R.id.fragmentContainer,
    HomeFragment())
42                  .commitNow()
43          }
44
45          observeViewModel()
46      }
47
48      /**
49       * Sets up observers for global UI states like loading
    indicators and error messages.
50       */
51      private fun observeViewModel() {
52          // Observe loading state to show/hide the
    ProgressBar
53          bookViewModel.isLoadingLiveData.observe(this) {
    isLoading ->
54              binding.progressBar.visibility = if (isLoading)
    View.VISIBLE else View.GONE
55          }
56
57          // Observe errors to show a Toast message
58          bookViewModel.errorLiveData.observe(this) {
    errorMessage ->
59              errorMessage?.let {
60                  Toast.makeText(this, it,
    Toast.LENGTH_LONG).show()
61              }
62          }
63      }
64  }
```

Tabel 1. 15 Source Code MainActivity

16. MyAdapter

```
1   package com.example.myapi_test.presentation.ui
2
3   import android.util.Log
4   import android.view.LayoutInflater
5   import android.view.ViewGroup
```

```
6   import androidx.recyclerview.widget.RecyclerView
7   import com.bumptech.glide.Glide
8   import com.example.myapi_test.R
9   import com.example.myapi_test.databinding.ItemListBinding
10  import com.example.myapi_test.presentation.model.BookUi
11
12  class MyAdapter(
13      private var books: MutableList<BookUi>,
14      private val onDetailButtonClicked: (book: BookUi) ->
    Unit,
15      private val onInfoButtonClicked: (book: BookUi) ->
    Unit,
16      private val onItemRootClicked: (book: BookUi) -> Unit
17  ) : RecyclerView.Adapter<MyAdapter.BookViewHolder>() {
18
19      inner class BookViewHolder(val binding:
    ItemListBinding) :
20          RecyclerView.ViewHolder(binding.root) {
21
22          init {
23              // Listener for the "Detail" button
24              binding.buttonInfo.setOnClickListener {
25                  val position = adapterPosition
26                  if (position != RecyclerView.NO_POSITION) {
27                      onDetailButtonClicked(books[position])
28                  }
29              }
30
31              // Listener for the "Info" button
32              binding.buttonDetail.setOnClickListener {
33                  val position = adapterPosition
34                  if (position != RecyclerView.NO_POSITION) {
35                      onInfoButtonClicked(books[position])
36                  }
37              }
38
39              // Listener for the entire item click
40              binding.root.setOnClickListener {
41                  val position = adapterPosition
42                  if (position != RecyclerView.NO_POSITION) {
43                      onItemRootClicked(books[position])
44                  }
45              }
46          }
47
48          fun bind(book: BookUi) {
49              binding.apply {
50                  textViewName.text = book.title
51                  textViewYear.text = book.release_date
52                  textViewAuthor.text = book.author
53
54                  Glide.with(itemView.context)
```

```
55                      .load(book.cover)
56                      .into(imageView)
57              }
58          }
59      }
60
61      override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): BookViewHolder {
62          val binding =
    ItemListBinding.inflate(LayoutInflater.from(parent.context)
    , parent, false)
63          return BookViewHolder(binding)
64      }
65
66      override fun getItemCount() = books.size
67
68      override fun onBindViewHolder(holder: BookViewHolder,
    position: Int) {
69          holder.bind(books[position])
70      }
71
72      fun updateBooks(newBooks: List<BookUi>) {
73          this.books.clear()
74          this.books.addAll(newBooks)
75          notifyDataSetChanged() // For production apps,
    consider using DiffUtil for better performance
76      }
77  }
```

Tabel 1. 16 Source Code MyAdapter

17. BookViewModel

```
1   package com.example.myapi_test.presentation.viewmodel
2
3   import android.util.Log
4   import android.app.Application
5   import androidx.lifecycle.AndroidViewModel
6   import androidx.lifecycle.LiveData
7   import androidx.lifecycle.MutableLiveData
8   import androidx.lifecycle.ViewModel
9   import androidx.lifecycle.viewModelScope
10  import com.example.myapi_test.data.mappers.toUiModel
11  import
    com.example.myapi_test.data.repository.BookRepositoryImpl
12  import
    com.example.myapi_test.domain.usecase.GetBooksUseCase
13  import com.example.myapi_test.presentation.model.BookUi
14  import kotlinx.coroutines.launch
15
16  class BookViewModel(application: Application) :
```

```
    AndroidViewModel(application) {
17      // --- Dependencies ---
18      // In a real app, use Dependency Injection (e.g., Hilt)
    to provide these dependencies.
19      // We manually create the instances here for
    simplicity.
20      private val bookRepository =
    BookRepositoryImpl(application.applicationContext)
21      private val getBooksUseCase =
    GetBooksUseCase(bookRepository)
22
23      // --- LiveData ---
24      private val _books = MutableLiveData<List<BookUi>>()
25      val booksLiveData: LiveData<List<BookUi>> get() =
    _books
26
27      private val _isLoading = MutableLiveData<Boolean>()
28      val isLoadingLiveData: LiveData<Boolean> get() =
    _isLoading
29
30      private val _errorMessage = MutableLiveData<String>()
31      val errorLiveData: LiveData<String> get() =
    _errorMessage
32
33      private val TAG = "BookViewModel"
34
35      /**
36       * Fetches the list of books by executing the use case.
37       */
38      fun fetchBooks() {
39          _isLoading.value = true
40          viewModelScope.launch {
41              // Execute the use case to get the result
42              val result = getBooksUseCase()
43
44              // Handle the result: onSuccess or onFailure
45              result.onSuccess { domainBooks ->
46                  // Map domain models to UI models before
    posting to LiveData
47                  _books.postValue(domainBooks.map {
    it.toUiModel() })
48                  Log.d(TAG, "Successfully fetched
    ${domainBooks.size} books.")
49              }.onFailure { exception ->
50                  _errorMessage.postValue("Failed to fetch
    books: ${exception.message}")
51                  Log.e(TAG, "Error fetching books",
    exception)
52              }
53              // Update loading state regardless of outcome
54              _isLoading.postValue(false)
55          }
```

```
56      }
57  }
```

Tabel 1. 17 Source Code BookViewModel

18. BookViewModelFactory

```
1   package com.example.myapi_test.presentation.viewmodel
2
3   import android.app.Application
4   import androidx.lifecycle.ViewModel
5   import androidx.lifecycle.ViewModelProvider
6
7   /**
8    * A factory class for creating BookViewModel instances.
9    * It's required because our BookViewModel has a
    constructor with parameters (Application).
10   */
11  class BookViewModelFactory(private val application:
    Application) : ViewModelProvider.Factory {
12
13      /**
14       * Creates a new instance of the given `Class`.
15       * @param modelClass a `Class` whose instance is
    requested
16       * @return a newly created ViewModel
17       */
18      override fun <T : ViewModel> create(modelClass:
    Class<T>): T {
19          // Check if the requested ViewModel class is our
    BookViewModel
20          if
    (modelClass.isAssignableFrom(BookViewModel::class.java)) {
21              // If it is, create and return an instance of
    it, passing the application context.
22              // The unchecked cast is safe because of the
    isAssignableFrom check.
23              @Suppress("UNCHECKED_CAST")
24              return BookViewModel(application) as T
25          }
26          // If it's a different ViewModel, throw an
    exception.
27          throw IllegalArgumentException("Unknown ViewModel
    class: ${modelClass.name}")
28      }
29  }
```

Tabel 1. 18 Source Code BookViewModelFactory

19. detail_fragment.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       tools:context=".presentation.ui.DetailFragment">
8
9       <!-- This Toolbar is specific to the DetailFragment for
    the back button and title -->
10      <androidx.appcompat.widget.Toolbar
11          android:id="@+id/detailToolbar"
12          android:layout_width="match_parent"
13          android:layout_height="?attr/actionBarSize"
14          android:background="?attr/colorPrimary"
15  android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar
    "
16          app:layout_constraintTop_toTopOf="parent"
17          app:titleTextColor="@android:color/white" />
18
19      <!-- NestedScrollView makes the content below the
    toolbar scrollable -->
20      <androidx.core.widget.NestedScrollView
21          android:layout_width="0dp"
22          android:layout_height="0dp"
23  app:layout_constraintTop_toBottomOf="@id/detailToolbar"
24          app:layout_constraintBottom_toBottomOf="parent"
25          app:layout_constraintStart_toStartOf="parent"
26          app:layout_constraintEnd_toEndOf="parent">
27
28          <!-- A LinearLayout organizes the image and text
    vertically -->
29          <LinearLayout
30              android:layout_width="match_parent"
31              android:layout_height="wrap_content"
32              android:orientation="vertical">
33
34              <!-- The ImageView with a fixed height of 600dp
    as requested -->
35              <ImageView
36                  android:id="@+id/detailImage"
37                  android:layout_width="match_parent"
38                  android:layout_height="600dp"
39                  android:scaleType="centerCrop"
40  android:contentDescription="@string/book_cover_image_large"
41                 tools:src="@tools:sample/backgrounds/scenic"
    />
42
```

```
43              <!-- A container for the text content with
    padding for better readability -->
44              <LinearLayout
45                  android:layout_width="match_parent"
46                  android:layout_height="wrap_content"
47                  android:orientation="vertical"
48                  android:padding="16dp">
49
50                  <TextView
51                      android:id="@+id/detailTitle"
52                      android:layout_width="match_parent"
53                      android:layout_height="wrap_content"
54                      android:layout_marginBottom="8dp"
    android:textAppearance="@style/TextAppearance.MaterialCompo
    nents.Headline5"
55                      tools:text="Nama Buku" />
56
57                  <TextView
58                      android:id="@+id/detailDescription"
59                      android:layout_width="match_parent"
60                      android:layout_height="wrap_content"
61                      android:layout_marginTop="8dp"
62                      android:lineSpacingMultiplier="1.2"
    android:textAppearance="@style/TextAppearance.MaterialCompo
    nents.Body1"
63
64                      tools:text="Deskripsi lengkap tempat
    destinasi." />
65              </LinearLayout>
66
67          </LinearLayout>
68
69      </androidx.core.widget.NestedScrollView>
70
71
72  </androidx.constraintlayout.widget.ConstraintLayout>
```

Tabel 1. 19 Source Code detail_fragment.xml

20. activity_main.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:id="@+id/main_layout"
6       android:layout_width="match_parent"
7       android:layout_height="match_parent"
8
9       android:background="?android:attr/colorBackground"
10
```

```
11      tools:context=".presentation.ui.MainActivity">
12
13      <FrameLayout
14          android:id="@+id/fragmentContainer"
15          android:layout_width="0dp"
16          android:layout_height="0dp"
17          app:layout_constraintTop_toTopOf="parent"
18          app:layout_constraintBottom_toBottomOf="parent"
19          app:layout_constraintStart_toStartOf="parent"
20          app:layout_constraintEnd_toEndOf="parent" />
21
22      <ProgressBar
23          android:id="@+id/progressBar"
24          style="?android:attr/progressBarStyle"
25          android:layout_width="wrap_content"
26          android:layout_height="wrap_content"
27          android:visibility="gone"
28          app:layout_constraintTop_toTopOf="parent"
29          app:layout_constraintBottom_toBottomOf="parent"
30          app:layout_constraintStart_toStartOf="parent"
31          app:layout_constraintEnd_toEndOf="parent"
32          tools:visibility="visible" />
33
34  </androidx.constraintlayout.widget.ConstraintLayout>
```

Tabel 1. 20 Source Code activity_main

21. home_fragment.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       tools:context=".presentation.ui.HomeFragment">
8
9       <com.google.android.material.appbar.AppBarLayout
10          android:id="@+id/homeAppBarLayout"
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          app:layout_constraintTop_toTopOf="parent"
14          app:layout_constraintStart_toStartOf="parent"
15          app:layout_constraintEnd_toEndOf="parent">
16
17          <androidx.appcompat.widget.Toolbar
18              android:id="@+id/homeToolbar"
19              android:layout_width="match_parent"
20              android:layout_height="?attr/actionBarSize"
21              android:background="?attr/colorPrimary"
22              app:title="@string/app_name"
```

```
23              app:titleTextColor="@android:color/white"
24  app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
25
26      </com.google.android.material.appbar.AppBarLayout>
27
28      <androidx.recyclerview.widget.RecyclerView
29          android:id="@+id/rv_character"
30          android:layout_width="0dp"
31          android:layout_height="0dp"
32          android:padding="4dp"
33          android:clipToPadding="false"
34  app:layoutManager="androidx.recyclerview.widget.LinearLayou
    tManager"
35          app:layout_constraintBottom_toBottomOf="parent"
36          app:layout_constraintEnd_toEndOf="parent"
37          app:layout_constraintStart_toStartOf="parent"
38  app:layout_constraintTop_toBottomOf="@id/homeAppBarLayout"
39          tools:listitem="@layout/item_list" />
40
41  </androidx.constraintlayout.widget.ConstraintLayout>
```

Tabel 1. 21 Source Code home_fragment

22. item_list.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="wrap_content"
7       android:layout_margin="8dp"
8       app:cardBackgroundColor="?attr/colorSurface"
9       app:cardCornerRadius="16dp"
10      app:cardElevation="4dp">
11
12      <androidx.constraintlayout.widget.ConstraintLayout
13          android:layout_width="match_parent"
14          android:layout_height="wrap_content"
15          android:padding="16dp">
16
17          <ImageView
18              android:id="@+id/imageView"
19              android:layout_width="100dp"
20              android:layout_height="150dp"
21  android:contentDescription="@string/book_cover_image"
22              android:scaleType="centerCrop"
23              app:layout_constraintStart_toStartOf="parent"
24              app:layout_constraintTop_toTopOf="parent"
25              tools:src="@tools:sample/avatars" />
26
```

```
27          <TextView
28              android:id="@+id/textViewName"
29              android:layout_width="0dp"
30              android:layout_height="wrap_content"
31              android:layout_marginStart="16dp"
32              android:ellipsize="end"
33              android:maxLines="2"
34  android:textAppearance="@style/TextAppearance.MaterialCompo
    nents.Subtitle1"
35              android:textStyle="bold"
36              app:layout_constraintEnd_toEndOf="parent"
37  app:layout_constraintStart_toEndOf="@id/imageView"
38              app:layout_constraintTop_toTopOf="parent"
39              tools:text="Main Title" />
40
41          <TextView
42              android:id="@+id/textViewAuthor"
43              android:layout_width="0dp"
44              android:layout_height="wrap_content"
45              android:layout_marginTop="4dp"
46              android:ellipsize="end"
47              android:maxLines="1"
48  android:textAppearance="@style/TextAppearance.MaterialCompo
    nents.Body2"
49  app:layout_constraintEnd_toEndOf="@id/textViewName"
50  app:layout_constraintStart_toStartOf="@id/textViewName"
51  app:layout_constraintTop_toBottomOf="@id/textViewName"
52              tools:text="Author" />
53
54          <TextView
55              android:id="@+id/textViewYear"
56              android:layout_width="0dp"
57              android:layout_height="wrap_content"
58              android:layout_marginTop="2dp"
59  android:textAppearance="@style/TextAppearance.MaterialCompo
    nents.Caption"
60  app:layout_constraintEnd_toEndOf="@id/textViewName"
61  app:layout_constraintStart_toStartOf="@id/textViewName"
62  app:layout_constraintTop_toBottomOf="@id/textViewAuthor"
63              tools:text="1990" />
64
65          <LinearLayout
66              android:id="@+id/buttonRow"
67              android:layout_width="0dp"
68              android:layout_height="wrap_content"
69              android:layout_marginTop="8dp"
70              android:gravity="end"
71              android:orientation="horizontal"
72              app:layout_constraintBottom_toBottomOf="parent"
73              app:layout_constraintEnd_toEndOf="parent"
74  app:layout_constraintStart_toStartOf="@id/textViewName"
75  app:layout_constraintTop_toBottomOf="@id/textViewYear"
```

```
76              app:layout_constraintVertical_bias="1.0">
77
78      <com.google.android.material.button.MaterialButton
79                  android:id="@+id/buttonInfo"
80      style="@style/Widget.MaterialComponents.Button.TextButton"
81                  android:layout_width="wrap_content"
82                  android:layout_height="wrap_content"
83                  android:layout_marginEnd="8dp"
84                  android:text="@string/info_button_text"
85                  android:textAllCaps="false" />
86
87      <com.google.android.material.button.MaterialButton
88                  android:id="@+id/buttonDetail"
89      style="@style/Widget.MaterialComponents.Button"
90                  android:layout_width="wrap_content"
91                  android:layout_height="wrap_content"
92                  android:text="@string/detail_button_text"
93                  android:textAllCaps="false" />
94          </LinearLayout>
95
96      </androidx.constraintlayout.widget.ConstraintLayout>
97  </androidx.cardview.widget.CardView>
```

Tabel 1. 22 Source Code item_list

23. nav_graph.xml

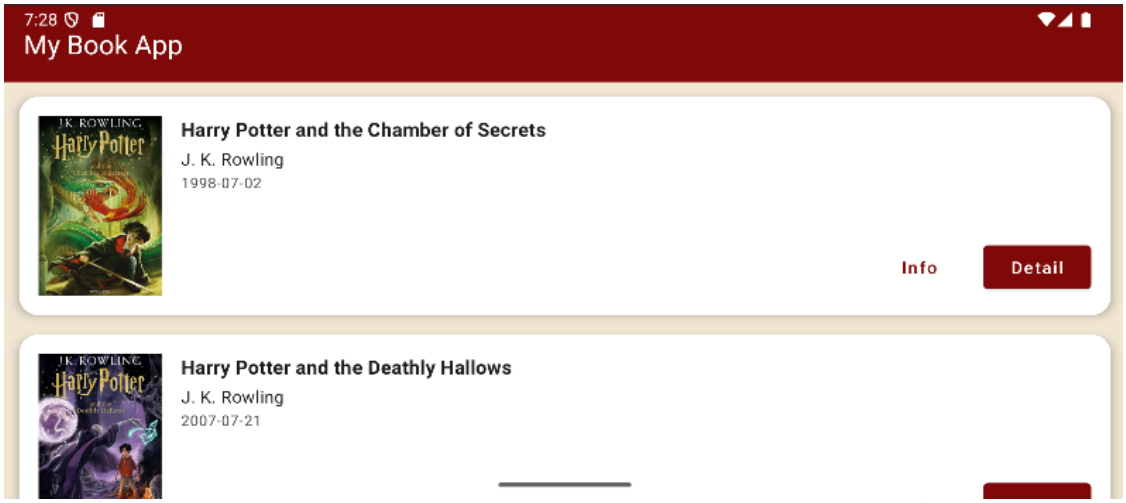
```
1   <?xml version="1.0" encoding="utf-8"?>
2   <navigation
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       android:id="@+id/nav_graph"
5       app:startDestination="@id/HomeFragment">
6
7       <fragment
8           android:id="@+id/HomeFragment"
9   android:name="com.example.myapi_test.presentation.ui.HomeFr
    agment"
10          android:label="HomeFragment" >
11          <action
12  android:id="@+id/action_HomeFragment_to_detailFragment"
13              app:destination="@id/detailFragment" />
14      </fragment>
15
16      <fragment
17          android:id="@+id/detailFragment"
18  android:name="com.example.myapi_test.presentation.ui.Detail
    Fragment"
19          android:label="DetailFragment" >
20          <argument
21              android:name="imageResId"
22              app:argType="integer" />
```

```
23          <argument
24              android:name="nama"
25              app:argType="string" />
26          <argument
27              android:name="deskripsi"
28              app:argType="string" />
29      </fragment>
30
31  </navigation>
```

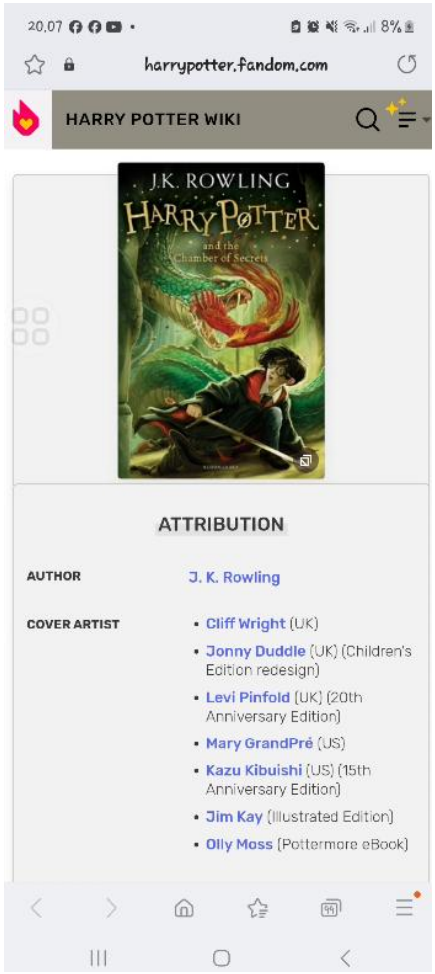
Tabel 1. 23 Source Code nav_graph

B. Output Program

Gambar 1 Screenshot Hasil Jawaban Soal 1



Gambar 2 Screenshot Hasil Jawaban Soal 1



Gambar 3 Screenshot tombol Detail

Gambar 4 Screenshot tombol Info

C. Pembahasan

1. MyApiResponse.kt

Pada file ini terutama bagian `com.example.myapi_test.data.api` berguna dalam struktur proyek Android agar bisa mengelompokkan class yang berhubungan saat kita melakukan pemanggilan `API` eksternal. Dimana file ini berfungsi sebagai Data Transfer Object (DTO), yaitu class-class yang mencerminkan struktur data `JSON` yang diterima dari `API` untuk diolah dengan mudah dalam aplikasi Android.

Dimulai dari bagian pertama dari kode mendefinisikan class `BookApiResponse` sebagai class utama mewakili respons `JSON` dari `API`. Di dalamnya terdapat properti `data` yang berupa daftar (list) dari objek `BookData`. Nah, artinya respons JSON dari API berisi array `data buku` dan setiap item di dalam array tersebut nantinya dipetakan menjadi objek `BookData`. Kemudian, class `BookData` berguna untuk merepresentasikan setiap entri data buku satu persatu dalam daftar dengan class ini hanya mempunyai satu properti bernama attributes berisi objek dari tipe `BookAttributes`. Jadi, seluruh informasi buku (seperti judul, penulis, tanggal rilis, dll) disimpan di dalam objek `attributes`.

Selanjutnya, ada class `BookAttributes` berguna untuk menyimpan detail lengkap dari sebuah buku. Disini terdapat enam properti yang manjadi informasi buku yaitu, `author` (nama penulis), `cover` (URL gambar sampul), `release_date` (tanggal rilis), `title` (judul buku), `wiki` (tautan ke halaman wiki buku), dan `summary` (ringkasan isi buku). Dimana semua properti tipe datanya String sesuai dengan tipe data pada `JSON API` yang digunakan.

Kesimpulan dari file ini bertujuan untuk memetakan struktur `JSON` dari `API` menuju bagian objek Kotlin agar bisa digunakan lebih mudah dalam proses parsing dan manipulasi data di aplikasi Android yang biasanya dibantu dengan library seperti Retrofit atau Moshi untuk aplikasi saya buat ini.

2. MyInstance

Pada file ini bagian paket `com.example.myapi_test.data.api` berguna untuk menginisialisasi dan menyediakan instance Retrofit nantinya saya gunakan saat melakukan permintaan ke `API`. `Retrofit` merupakan library populer di Android yang menangani komunikasi `HTTP` dengan cara sederhana dan efisien. Di file ini juga berperan sebagai singleton object berarti hanya akan dibuat satu instance Retrofit yang digunakan secara bersama-sama di seluruh aplikasi.

Bagian pertama dari file ini kita perlu mendefinisikan object `MyInstance`. Dimana `object` adalah cara agar bisa mendefinisikan singleton, yaitu objek yang

hanya memiliki satu instance. Di dalam objek ini, terdapat sebuah konstanta `BASE_URL` yang menyimpan alamat dasar dari `API` nantinya diakses yang mana di sini `https://api.potterdb.com/`. Dari alamat ini atau `URL` utama nantinya menjadi dasar semua endpoint `API` yang digunakan aplikasi ini.

Kemudian, ada properti api bertipe `MyService` yang dideklarasikan dengan kata kunci `val` dan menggunakan `by lazy`. Nah, `by lazy` berarti instance ini hanya akan dibuat satu kali saat pertama kali kita gunakan. Hal ini untuk efisiensi memori dan performa karena `Retrofit` tidak dibuat sebelum benar-benar dibutuhkan. Dalam blok `lazy` ini dibuatlah instance `Retrofit` menggunakan `Retrofit.Builder()`. Dengan bagian pertamanya itu ada `baseUrl(BASE_URL)` agar menetapkan `URL` dasar untuk semua permintaan `API`. Lalu, `addConverterFactory(GsonConverterFactory.create())` digunakan agar `Retrofit` bisa mengubah (convert) respons `JSON` dari `API` menjadi objek Kotlin menggunakan library `Gson`. Nah, setelah konfigurasi selesai bagian `build()` nantinya membangun instance `Retrofit` dan ada `create(MyService::class.java)` berfungsi saat membuat implementasi interface `MyService` berisi definisi endpoint-endpoint dari `API`.

Kesimpulan dari file ini untuk menyediakan cara yang paling rapi dan lebih efisien membuat dan mengakses instance `Retrofit` di seluruh aplikasi, sehingga kita sebagai pengembang aplikasi tidak perlu mengatur ulang koneksi `API` di setiap tempat yang berbeda.

3. MyService

Pada file ini bagian paket `com.example.myapi_test.data.api` berfungsi sebagai interface `Retrofit` yang mendefinisikan endpoint dari `API` ingin digunakan dalam aplikasi di buat. Dengan `Retrofit` yang menggunakan antarmuka seperti ini agar bisa memetakan `HTTP` request menjadi fungsi Kotlin, sehingga memudahkan komunikasi antara aplikasi dan server.

Kemudian, disini ada `import` terhadap anotasi `@GET` dari library `Retrofit`. Dari anotasi ini digunakan untuk menunjukkan bahwa fungsi di bawahnya akan melakukan permintaan `HTTP GET` menuju endpoint tertentu di `API`.

Lalu, didefinisikan sebuah interface bernama `MyService`. Dimana interface ini bertindak sebagai kontrak berisi deklarasi fungsi-fungsi mewakili permintaan dari `HTTP`. Di dalamnya terdapat satu fungsi yaitu `getMessage()` nantinya mengakses endpoint `"v1/books"` dari `API`. Nah, fungsi ini diberi anotasi `@GET("v1/books")` artinya saat fungsi ini dipanggil dengan `Retrofit` nantinya melakukan `HTTP GET` ke `URL` `https://api.potterdb.com/v1/books` (mengacu `BASE_URL` dari `MyInstance.kt`).

Selanjutnya, ada fungsi `getMessage()` ditandai dengan kata kunci `suspend` yang menunjukkan bahwa ini itu adalah `suspend function` dan harus dipanggil dari dalam coroutine. Hal ini menjadi bagian untuk pemrograman asinkron (non-blokir), sehingga operasi jaringan bisa dilakukan tanpa perlu membekukan UI aplikasi. Dengan fungsi ini akan mengembalikan objek `BookApiResponse` yaitu, model data yang telah didefinisikan sebelumnya untuk mencerminkan struktur `JSON` dari respons `API`.

Kesimpulan dari file `MyService.kt` bertugas menjembatani antara `Retrofit` dengan endpoint yang tersedia di `API` eksternal. Dengan hanya menuliskan fungsi yang disertai anotasi `HTTP`, maka saya sebagai pengembang aplikasi bisa melakukan permintaan `API` tanpa harus menulis kode jaringan secara manual.

4. BookDao

Pada file ini ada `com.example.myapi_test.data.database.dao` merupakan bagian dari struktur arsitektur aplikasi Android berguna untuk menangani akses menuju database lokal menggunakan `Room`. Dengan file inilah bisa mendefinisikan sebuah Data Access Object (DAO) yang digunakan oleh `Room` agar bisa mengakses data ke dalam database secara efisien dan lebih aman.

Bagian pertama dari kode mengimpor anotasi dan class-class penting dari Room seperti `@Dao`, `@Insert`, `@Query`, dan `OnConflictStrategy`, serta untuk mengimpor `BookDbEntity`, yaitu entitas (struktur tabel) yang merepresentasikan data buku di dalam database. Kemudian, didefinisikan sebuah interface `BookDao` sebagai tanda anotasi `@Dao` yang menunjukkan bahwa interface ini nantinya digunakan `Room` untuk meng-generate implementasi kode database secara otomatis. Juga, interface ini berisi tiga fungsi utama yang menangani operasi dasar dari tabel buku di database.

Selanjutnya, fungsi ada `insertBooks()` berguna untuk menyisipkan daftar buku menuju dalam database. Dimana fungsi ini menggunakan anotasi `@Insert` dengan `OnConflictStrategy.REPLACE` artinya jika ada entri dengan primary key yang sama, maka data lama nantinya digantikan dengan yang baru. Nah, fungsi ini bersifat `suspend`, sehingga dijalankan dalam coroutine agar tidak bisa memblokir thread utama.

Kemudian, ada fungsi `getAllBooks()` yang diberi anotasi `@Query("SELECT * FROM books ORDER BY title ASC")`. Nah, fungsi ini digunakan untuk mengambil semua data buku dari tabel books dan mengurutkannya sesuai judul urutannya dari A sampai Z. Hasilnya muncul daftar dari objek `BookDbEntity`. Karena ini juga suspend, maka pemanggilannya harus berada dalam coroutine.

Berikutnya, ada fungsi ketiga `clearAllBooks()` menggunakan anotasi `@Query` untuk menjalankan perintah SQL `DELETE FROM books`. Hal ini nantinya menghapus seluruh isi tabel books di database. Oleh karena itu, fungsi ini berguna misalnya saat saya ingin merefresh data dengan yang baru dari `API`.

Kesimpulan dari file `BookDao.kt` merupakan komponen penting dalam sistem database lokal menggunakan `Room` yang menyediakan antarmuka untuk melakukan penyimpanan, pembacaan, dan penghapusan data buku secara terstruktur dan aman di dalam aplikasi ini.

5. BookDbEntity

Pada paket `com.example.myapi_test.data.database.entity` berguna untuk mendefinisikan struktur tabel dalam database lokal yang digunakan oleh library `Room` di Android. Dimana file ini adalah entitas database yang mewakili setiap baris (record) dalam tabel `books`. Nah, `Room` akan menggunakan class data seperti ini agar bisa memetakan data antara database `SQLite` dan objek `Kotlin` secara otomatis.

Dengan class ini diberi anotasi `@Entity(tableName = "books")` artinya `Room` nantinya membuat atau menggunakan tabel bernama books di database dan setiap objek dari class ini nantinya direpresentasikan sebagai satu baris dalam tabel tersebut. Nama tabel bisa disesuaikan dan dalam project aplikasi ini disesuaikan menjadi `"books"` agar sesuai dengan konteks data yang ingin disimpan.

Kemudian, nama class `BookDbEntity` menggunakan bentuk data (data class) karena class ini hanya bertugas menyimpan data dan menyediakan fungsi-fungsi dasar seperti `equals()`, `hashCode()`, dan `toString()` yang otomatis dibuat oleh Kotlin. Di dalam class ini terdapat enam property seperti, `title`, `author`, `cover`, `releaseDate`, `summary`, dan `wiki` yang seluruhnya bertipe data `String`. Berikutnya, properti `title` diberi anotasi `@PrimaryKey` yang menjadikannya kunci utama (primary key) dari tabel `books`. Hal ini berarti nilai `title` harus unik di setiap baris tabel. Dengan saya tambahkan komentar dalam kode ini juga menjelaskan bahwa penggunaan judul sebagai primary key hanya untuk keperluan sederhana atau percobaan dalam aplikasi nyata, biasanya digunakan `ID` unik yang berasal dari API agar lebih bagus dan terhindar dari duplikasi data.

Kesimpulan dari file `BookDbEntity.kt` adalah representasi dari satu entri buku dalam database `Room` dan menjadi jembatan antara data lokal dan logika aplikasi ini. Kita lihat lagi dari struktur ini memungkinkan developer bisa menyimpan data hasil dari `API` (di sini buku dari `API Harry Potter`) secara lokal, sehingga dapat diakses walaupun dalam kondisi offline.

6. AppDatabase

Pada file ini ada paket `com.example.myapi_test.data.database` berfungsi untuk mendefinisikan database utama menggunakan `Room` karena library ini dari Android Jetpack yang menyediakan abstraksi di atas `SQLite`. Dimana file ini bagian class database utama tempat seluruh entitas dan DAO dihubungkan, sehingga `Room` bisa membangun dan mengelola database lokal di aplikasi tersebut.

Kemudian, bagian pertama dari anotasi `@Database` menyatakan bahwa class `AppDatabase` merupakan sebuah database dengan di dalam anotasi tersebut terdapat `parameter entities = [BookDbEntity::class]` berarti database ini mempunyai satu tabel, yaitu tabel `books` yang diwakili oleh class `BookDbEntity. version = 1` yang menunjukkan versi pertama dari database ini dan `exportSchema = false` artinya `Room` nantinya tidak bisa mengekspor skema database ke file metadata. Dari fitur inilah biasanya digunakan untuk dokumentasi atau pengujian migrasi.

Selanjutnya, ada class `AppDatabase` adalah abstract class yang mewarisi dari `RoomDatabase`. Karena abstrak, maka kita tidak membuat instance-nya secara langsung, tetapi `Room` nantinya yang mengelolanya. Di dalam class ini terdapat fungsi abstrak `bookDao()` yang mengembalikan objek `BookDao`. Dari fungsi ini adalah titik masuk utama untuk mengakses data buku melalui operasi yang telah didefinisikan di `BookDao.kt`.Bagian berikutnya ada `companion object` berguna untuk mengelola instance tunggal (singleton) dari database yang di dalamnya terdapat properti `INSTANCE` bersifat `@Volatile` berarti nilainya akan selalu diperbarui dan terlihat oleh semua thread. Hal ini penting agar bisa menghindari pembuatan banyak instance database dalam aplikasi yang sama dapat menyebabkan crash.

Lalu, ada fungsi `getDatabase(context: Context): AppDatabase` berguna untuk mendapatkan instance database. Apabila `INSTANCE` belum dibuat, maka fungsi ini nanti yang membuatnya menggunakan `Room.databaseBuilder(...)`. Dimana pembuatan instance ini dibungkus dalam `synchronized(this)` agar aman terhadap akses dari banyak thread secara bersamaan. Terdapat konfigurasi yang diberikan meliputi

`context.applicationContext` referensi class `AppDatabase` dan nama database `"book_database"`. Selain itu, `fallbackToDestructiveMigration()` digunakan sebagai strategi migrasi versi database yang mana database lama nanti dihapus dan dibuat ulang jika versi berubah dengan biasanya hanya dipergunakan saat adanya pengembangan karena akan menghapus data lama.

Kesimpulan dari file `AppDatabase.kt` merupakan pondasi utama agar bisa pengelolaan database lokal menggunakan `Room` yang menghubungkan entitas data (`BookDbEntity`) dengan akses datanya (`BookDao`) dan menyediakan cara yang lebih aman serta efisien untuk membangun instance database dalam aplikasi Android yang sedang kita buat.

7. BookMapper.kt

Pada file ini ada paket `com.example.myapi_test.data.mappers` berperan penting dalam bagaimana cara pemrosesan dan konversi data antar layer (data layer, domain layer, dan presentation layer) di bagian arsitektur aplikasi Android yang terstruktur. Ada fungsi-fungsi di file ini dikenal sebagai `mapper` yang bertugas mengubah satu jenis objek data menjadi jenis objek lain yang lebih sesuai dengan konteks penggunaannya (misalnya dari format `API` ke model domain, atau dari database ke tampilan `UI`).

Kemudian, bagian pertama dari file ini ada fungsi `BookData.toDomain()`. Dimana fungsi ini berguna untuk mengubah objek `BookData` karena hasil deserialisasi dari `API` (DTO atau *Data Transfer Object*) menjadi objek `Book` yang merupakan gambaran dari domain layer. Di sini pula ada konversi karena penting agar domain layer tidak selalu bergantung langsung pada struktur data dari `API` dan bisa bekerja dengan bentuk data lebih stabil.

Selanjutnya, terdapat fungsi `Book.toUiModel()` bertugas untuk mengubah objek `Book` dari domain layer menjadi `BookUi`, yaitu model yang digunakan oleh presentation layer (biasanya digunakan oleh `ViewModel` atau langsung oleh `UI`). Hal ini memungkinkan pemisahan antara logika aplikasi (domain)

dan tampilan (UI), sehingga setiap lapisan bisa berubah tanpa mempengaruhi yang lain secara langsung.

Berikutnya, ada fungsi BookAttributes.toDbEntity() berguna saat ingin mengubah BookAttributes yang merupakan bagian dari respons API menjadi BookDbEntity, yaitu entitas untuk disimpan menuju database Room. Dimana fungsi ini penting dalam proses caching data dari API ke database lokal agar bisa digunakan kembali tanpa perlu koneksi internet lagi.

Lalu, fungsi BookDbEntity.toDomain() berguna untuk mengubah data dari database (BookDbEntity) menjadi objek domain (Book). Dimana fungsi ini digunakan saat kita ingin mengambil data dari database dan mau memprosesnya di dalam domain layer tanpa perlu melibatkan struktur data database secara langsung.

Kesimpulan dari file BookMapper.kt berfungsi sebagai jembatan untuk konversi data antar lapisan aplikasi sampai ke data dari API, database, dan domain dipisahkan dengan jelas. Yang mana saat kita praktikkan di aplikasi ini mendukung prinsip separation of concerns dan meningkatkan fleksibilitas, kemudahan dalam membaca kodenya, serta maintainability dari kode aplikasi secara keseluruhan.

8. BookRepositoryImpl

Pada file ini ada com.example.myapi_test.data.repository merupakan implementasi dari interface BookRepository berguna saat didefinisikan dalam domain layer. Dimana BookRepositoryImpl bertugas untuk mengelola pengambilan data buku dari dua sumber remote (API) dan lokal (database Room) bertujuannya agar bisa menyediakan data yang konsisten untuk lapisan-lapisan lain dalam arsitektur aplikasi, seperti ViewModel dan UI. Jadi, di file ini pula sebenarnya

Di file ini pula untuk mengimplementasikan sebuah pola arsitektur dalam pengembangan aplikasi Android, yaitu "Single Source of Truth" bertujuan agar bisa membuat aplikasi tetap fungsional baik saat online maupun offline dengan mengandalkan database lokal sebagai sumber data utama bagi UI aplikasi ini.

Lalu, ada deklarasi class dan inisialisasi sumber data di baris [19] – [22] dan [57] adalah fondasi dari repositori. Karena class BookRepositoryImpl dibuat

untuk menjalankan fungsi yang didefinisikan dalam interface `BookRepository`. Dari class ini mengelola dua sumber data `apiService` yang digunakan agar mengambil data dari internet melalui `Retrofit` (network), dan `bookDao` yang digunakan untuk berinteraksi dengan database lokal (`Room`). Adanya inisialisasi keduanya di sini, maka repositori siap menjadi perantara antara sumber data remote dan lokal.

Selanjutnya, ada fungsi utama `getBooks()` dan penggunaan coroutine di baris [24] – [56] karena fungsi utama nantinya dipanggil oleh bagian lain dari aplikasi (misalnya, `ViewModel`) agar bisa mendapatkan daftar buku. Dimana fungsi ini ditandai sebagai `suspend` berarti ia adalah fungsi coroutine yang dapat berjalan di latar belakang tanpa memblokir interface pengguna (`UI`). Dengan penggunaan `withContext(Dispatchers.IO)` secara eksplisit memindahkan semua operasi di dalamnya (seperti panggilan `API` dan akses database) ke *thread* IO yang memang dirancang untuk tugas-tugas berat tersebut, sehingga aplikasi tetap masih responsif.

Kemudian, di baris [26] – [43] sinkronisasi data dengan mengambil dari `API` dan menyimpan ke database yang mana inti dari blok `try` ini adalah proses sinkronisasi. Dimulai aplikasi pertama-tama mencoba mengambil data buku terbaru dari `API`. Jika berhasil nantinya membersihkan semua data lama yang ada di database lokal (`clearAllBooks`) untuk memastikan tidak ada data terduplikat. Setelah itu, data dari `API` diubah formatnya (`map`) agar sesuai struktur tabel database (`toDbEntity`) dan lanjutannya itu dimasukkan menuju database lokal. Adanya proses ini memastikan bahwa database lokal selalu berisi data paling terbaru jika koneksi internet sudah tersedia.

Berikutnya, di baris [39] – [43] untuk penanganan Network Failure karena blok `catch` ini adalah mekanisme penanganan kesalahan (error handling) jika terjadi masalah saat mengambil data dari `API`, misalnya karena tidak ada koneksi internet. Alih-alih membuat aplikasi crash, maka kode ini hanya akan mencatat error tersebut (`Log.e`). Hal terpenting, program tidak berhenti di sini, melainkan nanti melanjutkan lagi eksekusi ke langkah berikutnya. Ini memungkinkan aplikasi tetap berfungsi dengan menggunakan data yang sudah ada di database (`cache`).

Setelah itu, di baris [48] – [55] bagian prinsip "Single Source of Truth" selalu mengembalikan data dari database karena terlepas dari apakah pengambilan data dari API berhasil ataupun gagal, fungsi ini nantinya selalu mengambil data dari database lokal (bookDao.getAllBooks()). Apabila API berhasil, maka data diambil adalah data baru yang baru saja disimpan. Jika API gagal, maka data diambil adalah data lama yang tersimpan sebelumnya (cache). Nah, data dari database ini kemudian diubah formatnya (toDomain) menjadi model data yang akan digunakan oleh lapisan UI dengan dibungkus dalam objek Result.success sebagai penanda keberhasilan operasi secara keseluruhan. Jika bahkan membaca dari database pun gagal, ia akan mengembalikannya ke Result.failure.

9. Book

Pada file ini ada paket com.example.myapi_test.domain.model berguna untuk mendefinisikan class data Book yang berfungsi sebagai model inti (*domain entity*) dalam arsitektur aplikasi. Maksud dari konteks arsitektur bersih (*clean architecture*) file ini mewakili lapisan domain, yaitu pusat dari logika bisnis aplikasi. Dengan model ini tidak memiliki ketergantungan terhadap sumber data (seperti API atau database) maupun terhadap elemen UI berarti model ini bersifat murni dan dapat digunakan di mana saja dalam aplikasi.

Kemudian, di baris [7] – [14] untuk deklarasi data class Book berguna saat mendeklarasikan sebuah class Book menggunakan kata kunci data class. Dimana penggunaan data class di sini sudah pilihan yang tepat karena tujuannya hanya untuk menampung data. Dengan data class, maka kotlin secara otomatis membuatkan fungsi-fungsi standar yang sangat berguna di belakang layar, seperti toString(), equals(), hashCode(), dan copy(). Ini membuat kode lebih ringkas dan bebas dari *boilerplate*. Intinya, baris ini mendefinisikan Book sebagai sebuah objek model data yang sederhana dan efisien.

Selanjutnya, di baris [8] – [13] bagian properti atau atribut yang mendefinisikan sebuah Book. Di setiap properti dideklarasikan dengan val berarti nilainya tidak bisa diubah setelah objek Book dibuat (*immutable*). Nah, ini membuat data lebih aman dan mudah diprediksi. Dengan setiap atribut (title, author,

`cover`, dll.) yang mempunyai tipe data String menunjukkan bahwa semua itu nantinya menyimpan informasi dalam bentuk teks. Jadi, bagian ini menetapkan bahwa setiap objek `Book` dalam aplikasi yang saya buat ini akan selalu menampilkan judul, penulis, sampul, tanggal rilis, ringkasan, dan tautan wiki.

10. BookRepository

Pada file ini ada `com.example.myapi_test.domain.repository` berguna untuk mendefinisikan interface `BookRepository` karena dalam arsitektur aplikasi, interface ini bertindak sebagai jembatan antara lapisan domain (*domain layer*, tempat logika bisnis) dan lapisan data (*data layer*, tempat sumber data seperti `API` atau database). Dimana tujuannya untuk memisahkan logika bisnis dari detail teknis pengambilan data. Dengan lapisan domain hanya peduli apa yang bisa dilakukan (yaitu mendapatkan buku), bukan bagaimana caranya (apakah dari internet atau dari `cache` lokal). Ini membuat kode lebih bersih dan fleksibel.

Selanjutnya, di baris [15] terdapat `suspend fun getBooks(): Result<List<Book>>` merupakan satu-satunya aturan atau fungsi yang didefinisikan dalam BookRepository. Lebih tepatnya di bagian `suspend fun` sebagai penanda bahwa `getBooks` itu fungsi *asynchronous* yang dapat dijeda dan dilanjutkan, dirancang untuk berjalan di latar belakang (menggunakan coroutines) agar tidak mengganggu antarmuka pengguna (`UI`) dengan `getBooks()` menjadi nama fungsi yang jelas agar mendapatkan daftar buku melibatkan : `Result<List<Book>>` adalah tipe data yang akan dikembalikan karena Result ini pembungkus (`wrapper`) yang sangat berguna karena bisa berisi salah satu dari dua kemungkinan yaitu, sebuah daftar objek Book jika berhasil (`Success`) ataupun sebuah `Exception` jika terjadi kegagalan (`Failure`). Jadi, ini cara yang aman dan lebih modern dalam menangani hasil operasi yang bisa saja gagal.

11. GetBooksUseCase

Pada file ini ada `com.example.myapi_test.domain.usecase` berguna untuk mendefinisikan sebuah class `GetBooksUseCase` merupakan bagian dari lapisan domain. Nah, class ini menjadi representasi dari satu kasus penggunaan

atau logika bisnis tunggal yang ada dalam aplikasi kita buat. Lalu, perhatikan bahwa class ini membutuhkan `BookRepository` dalam konstruktornya. Hal ini menjadi poin kunci dimana `GetBooksUseCase` bergantung dengan kontrak (`BookRepository` interface), bukan hanya di implementasi detailnya (`BookRepositoryImpl`). Disini membuat `UseCase` sangat fleksibel dan mudah diuji secara terpisah.

Selanjutnya, di baris [16] – [18] ada `suspend operator fun invoke(): Result<List<Book>>` merupakan inti dari `UseCase` engan penggunaan `suspend operator fun invoke` bagian trik khusus. Disini ada `operator fun invoke` berguna dalam memberi kemungkinan bahwa objek dari class `GetBooksUseCase` untuk dipanggil seolah-olah itu sebuah fungsi. Jadi, daripada menulis getBooksUseCase.execute(), saya bisa langsung menulis `getBooksUseCase()` dan membuat pemanggilan dari `ViewModel` menjadi lebih bersih dan ringkas menggunakan `suspend` sebagai penanda operasi ini bersifat asinkron juga harus dijalankan di dalam sebuah Coroutine, sehingga tidak akan memblokir UI. Nah, isi Fungsi menjadi satu-satunya hal yang perlu dilakukan oleh return `bookRepository.getBooks()` yang mana hanya meneruskan perintah menuju repositori. Tugasnya bukan untuk mengetahui bagaimana cara mendapatkan buku, tetapi hanya untuk meminta buku dari repositori yang telah diberikan API.

12. BookUi

Pada file ini `com.example.myapi_test.presentation.model` berguna saat ingin mendefinisikan class data `BookUi`. Dimana class ini merupakan model data khusus untuk lapisan presentasi (`UI`) berarti data ini sudah dipersiapkan dalam format yang sudah siap ditampilkanke layar user, biasanya hasil transformasi dari domain model menggunakan `mapper`. Dengan adanya model khusus untuk `UI` seperti ini, kita dapat menjaga agar interface ke user tidak langsung bergantung dengan detail dari data atau domain layer.

Selanjutnya, di baris [10] – [18] adalah bagian terpenting dari kode ini karena ada anotasi `@Parcelize` dan implementasi `: Parcelable`. Dimana dalam

Android, jika saya ingin mengirim objek kustom (seperti `BookUi`) dari satu layar (`Activity/Fragment`) menuju layar lainnya dengan objek tersebut harus `"parcelable"`. Lebih jelasnya : `Parcelable` merupakan interface Android yang menandakan bahwa objek dari kelas ini dapat diubah menjadi format yang bisa dikirim dan diterima antar komponen melibatkan `@Parcelize` adalah jalan pintas canggih dari bahasa `Kotlin` karena disini saya harus menulis banyak kode *boilerplate* yang rumit agar bisa mengimplementasikan `Parcelable` secara manual. Adanya `@Parcelize` membuat `Kotlin` secara otomatis membuat semua kode tersebut untuk kita di belakang layar dan membuat kode jauh lebih *clean* yang memudahkan pengelolaan data kompleks dalam navigasi antar layar.

Berikutnya, di baris [12] – [17] perlu mendeklarasikan sebuah data class `BookUi` yang fungsinya adalah sebagai wadah data. Dimana penggunaan nama `BookUi` agar bisa membedakannya dari model `Book` di lapisan domain. Dari class ini secara spesifik disiapkan untuk lapisan presentasi (UI) berarti data di dalamnya mungkin sudah diformat dan ditampilkan. Properti di dalamnya (enam properti utama bertipe String yaitu, `author`, `cover`, `release_date`, `title`, `wiki`, dan `summary`) menjadi atribut-atribut nantinya saya tampilkan langsung di layar kepada user.

13. DetailFragment

Pada file terdapat class `DetailFragment` bagian sebuah komponen `UI` dalam arsitektur Android bertanggung jawab agar bisa menampilkan detail dari satu buku. Dimana class ini turunan dari `Fragment` berarti bagian dari interface user yang dapat digabungkan ke dalam `Activity` dengan fungsinya untuk menampilkan informasi mendetail berdasarkan data `BookUi` yang dikirimkan dari layar sebelumnya.

Selanjutnya, di file ini juga ada `binding` dan `currentBook` dengan `Fragment` ini menggunakan `ViewBinding` melalui `objek _binding` nantinya properti binding yang diinisialisasi di `onCreateView()` dan dibersihkan pada `onDestroyView()` menghindari memory leak. Dari variabel `currentBook` bertipe `BookUi` akan menyimpan data buku yang dipilih. Dimana

data ini diambil dari argumen `Bundle` di `onCreate()` menggunakan metode `getParcelable()` dengan pengecekan versi Android agar kompatibel dengan API terbaru dan lama. Lalu, `onCreateView()` ini bertanggung jawab untuk meng-inflate layout dari `DetailFragmentBinding`, yaitu tampilan `XML` yang dikaitkan `fragment` ini. Dan view ini kemudian dikembalikan agar bisa ditampilkan ke user.

Kemudian, ada `onViewCreated()` membantu dalam mengatur tampilan setelah layout berhasil dibuat. Di sini peran toolbar nantinya dikonfigurasi agar bisa menampilkan tombol `"Back"` dengan memanggil `activity.setSupportActionBar()`. Dari tombol ini dikaitkan dengan `activity.onBackPressed()` untuk kembali ke layar sebelumnya. Lanjutannya itu jika `currentBook` tidak null, `fragment` akan menampilkan data buku menuju tampilan seperti judul, deskripsi, dan gambar sampul. Apabila `URL` sampul tidak kosong, maka gambar nantinya dimuat menggunakan library `Glide` jika kosong, digunakan gambar default dari resource lokal `API`. Berikutnya, `onDestroyView()` membersihkan `_binding` agar referensi terhadap view dilepas ketika `fragment` dihancurkan. Hal ini terjadi di praktik aplikasi saya buat karena penting dalam pengelolaan lifecycle fragment untuk mencegah memory leak.

Juga, di file ini ada `companion object` menyediakan fungsi `newInstance()` yang digunakan untuk membuat instance baru dari `DetailFragment` dengan menyisipkan objek `BookUi` menuju `Bundle` sebagai `parcelable`. Nah, konstanta `ARG_BOOK` digunakan sebagai key untuk menyimpan dan mengambil data buku dalam bundle ini.

Kesimpulan di file `DetailFragment` adalah implementasi `fragment` yang mengikuti praktik Android modern termasuk penggunaan `ViewBinding`, `Parcelable`, `Fragment Arguments`, dan `Glide`. Dan tujuan dari file ini agar bisa menampilkan detail buku secara dinamis dan aman sambil menjaga struktur kode tetap masih modular, serta terpisah dari logika bisnis dan data (*clean architecture*).

14. HomeFragment

Pada class `HomeFragment` merupakan `fragment` yang bertanggung jawab agar bisa menampilkan daftar buku di halaman utama aplikasi ini. Dimana `fragment` ini menghubungkan `ViewModel` (`BookViewModel`) dengan `UI` melalui `ViewBinding` dan `RecyclerView`. Dari struktur fragment ini mengikuti arsitektur `MVVM` (`Model-View-ViewModel`) yang memisahkan logika data dan tampilan.

Kemudian, ada properti `_binding` adalah objek nullable dari `HomeFragmentBinding` yang digunakan untuk mengakses komponen `UI` secara efisien. Dimana `binding` non-nullable menjamin akses hanya ketika binding sudah dibuat (selama `fragment` aktif). Nah, `binding` diinisialisasi di `onCreateView()` dan dibersihkan di `onDestroyView()` untuk mencegah memory leak pada project sudah dibuat.

Selanjutnya, bagian `onViewCreated()` karena `ViewModel` diinisialisasi menggunakan `ViewModelProvider` yang terikat dengan `requireActivity()`. Hal ini memungkinkan `ViewModel` dibagikan antara `fragment` dan `activity` induk agar menjaga konsistensi data. Peran `ViewModel` disini untuk mengamati data buku yang dikemas dalam `LiveData`. Lalu, fungsi `setupRecyclerView()` saat mengatur tampilan daftar buku menggunakan `LinearLayoutManager` dan `MyAdapter`. Dari `adapter` ini menerima tiga lambda untuk menangani klik pada tombol `detail`, tombol `info` (`Wikipedia`), dan klik di item root semua diarahkan menuju fungsi navigasi atau pembuka link. Berikutnya, ada fungsi `observeViewModel()` mengamati perubahan di `booksLiveData`. Apabila terdapat data baru, maka daftar buku di adapter diperbarui. Observasi ini menjaga agar `UI` tetap sinkron dengan adanya perubahan data di `ViewModel` secara reaktif.

Setelah itu, perlu melakukan pengambilan data awal karena masih di `onViewCreated()` dilakukan pengecekan apakah daftar buku masih kosong. Jika iya, maka `bookViewModel.fetchBooks()` dipanggil agar bisa memuat data

buku dari repository, sehingga menghindari pemanggilan ulang saat konfigurasi ulang fragment (misalnya saat rotasi layar). Juga, ada fungsi navigateToDetail() membantu saat ingin menangani perpindahan menuju layar detail ketika item buku kita klik. Menggunakan parentFragmentManager.beginTransaction(), maka fragment diganti ke DetailFragment yang dikirim argumen BookUi dan transaksi ditambahkan ke back stack agar user bisa kembali dengan cara klik tombol back. Serta, ada fungsi openWikiLink() sebagai percobaan saat membuka URL dari properti wiki buku dengan Intent. Apabila URL valid, nantinya terbuka di browser. Bila tidak valid atau gagal, maka aplikasi dibuat itu menampilkan pesan kesalahan melalui Toast. Nah, hal ini menunjukkan implementasi yang defensif terhadap kemungkinan URL kosong atau format salah. Terakhir, ada fungsi onDestroyView() bertugas mengosongkan _binding agar tidak terjadi memory leak setelah view fragment dihancurkan. Dalam praktik dari project aplikasi yang saya buat ini sangat penting agar fragment tidak menyimpan referensi terhadap view yang sudah tidak diperlukan.

Kesimpulan dari file HomeFragment adalah bagian komponen presentasi utama yang mengelola tampilan daftar buku, merekam semisal ada perubahan data melalui ViewModel, dan menyediakan interaksi user seperti navigasi ke detail dan membuka info tambahan. Dengan melakukan implementasi di project ini sudah mencerminkan arsitektur MVVM yang baik dan menjaga *clean code* dengan pemisahan tanggung jawab antara logika UI dan data.

15. MainActivity

Pada file MainActivity berguna sebagai titik utama aplikasi Android karena aktivitas ini bertanggung jawab dalam mengatur tampilan awal, menginisialisasi ViewModel yang kita punya di project aplikasi, serta memuat fragment utama (HomeFragment). Dengan aktivitas ini juga membantu saya dalam menangani splash screen dan memantau status loading serta error.

Kemudian, di file ini ada bagian `onCreate()`, `installSplashScreen()` dipanggil sebelum `super.onCreate()` agar bisa memastikan *splash screen* compatible dengan `Android 12+` ditampilkan saat kita klik aplikasi dan muncul *splash screen* saat dimulai aplikasinya. Nah, *splash screen* dijaga tetap terlihat melalui `setKeepOnScreenCondition` nantinya tetap bisa menampilkan layar *splash* selama `isLoadingLiveData` bernilai true, yaitu saat data awal masih dimuat oleh `ViewModel`. Lalu, ada `ActivityMainBinding` berguna untuk menghubungkan kode bahas `Kotlin` sudah dibuat dengan layout `XML` (`activity_main.xml`) dengan cara perlu memanggil `ActivityMainBinding.inflate(layoutInflater)` dan memberikan `setContentView(binding.root)` yang mana semua elemen `UI` dalam layout bisa diakses melalui properti binding yang membuat kode lebih aman dan bebas dari adanya kesalahan runtime akibat kesalahan `ID`.

Selanjutnya, ada `ViewModel` lebih tepatnya `BookViewModel` dibuat menggunakan `ViewModelProvider` dan `BookViewModelFactory` dengan penggunaan `factory` diperlukan karena `BookViewModel` membutuhkan parameter (objek Application). Dimana `ViewModel` ini disiapkan di level `activity` agar bisa dibagikan ke `HomeFragment` dan `DetailFragment`. Berikutnya, ada kode `if (savedInstanceState == null)` untuk memastikan bahwa `HomeFragment` hanya bisa dimuat sekali saja saat `activity` pertama kali dibuat oleh programnya. Apabila `activity` dipulihkan ketika kita melakukan rotasi layar atau pembaruan konfigurasi lainnya, `fragment` tidak akan dimuat ulang secara manual karena sistem Android sudah memulihkannya kembali.

Setelah itu, observasi Loading dan Error dengan fungsi `observeViewModel()` yang mengatur dua observer seperti, `isLoadingLiveData` perlu diamati agar bisa menampilkan atau menyembunyikan `ProgressBar` di `UI` sesuai status loading dan `errorLiveData` diamati untuk menampilkan pesan kesalahan dalam bentuk `Toast` jika terjadi error dalam program saat memuat data.\

Kesimpulan file MainActivity adalah bagian pusat kendali utama yang dalam mengelola UI dan menghubungkan ViewModel dengan berbagai elemen tampilan seperti XML . Juga disini mengatur *splash screen* yang adaptif terhadap status loading data dan memberikan feedback error kepada user agar tau permasalahan dari aplikasi yang sedang dijalankan. Dimana penggunaan arsitektur MVVM, ViewBinding, serta fragment manajemen modern menjadikan struktur aplikasi ini *clean architecture* dan modular.

16. MyAdapter

Pada file MyAdapter merupakan adapter yang menghubungkan data buku (BookUi) dengan tampilan daftar (RecyclerView) di aplikasi ini. Dimana adapter bertugas dalam membuat tampilan setiap item dengan mengikat data menuju tampilan tersebut dan menangani interaksi yang dilakukan oleh user seperti klik tombol atau item yang ada di aplikasi. Lalu, ada konstruktor dan parameter yang mana konstruktor dari MyAdapter menerima empat parameter penting seperti, books sebagai daftar data buku nantinya ditampilkan, onDetailButtonClicked, onInfoButtonClicked, dan onItemRootClicked menjadi function (callback) yang mengatur aksi saat user mengklik tombol "Detail", "Info", ataupun seluruh item. Hal ini memberikan fleksibilitas dan menjaga adapter tetap terpisah dari logika tampilan aplikasi.

Kemudian, class ViewHolder ada BookViewHolder merupakan class dalam (*inner class*) yang bertanggung jawab mengelola satu item tampilan di daftar dengan menyimpan objek ItemListBinding berisi referensi langsung menuju elemen-elemen UI seperti textViewName, imageView, dll. Dalam init, setiap elemen diberi setOnClickListener yang mengeksekusi callback sesuai posisi item di dalam daftar. Setelah itu, ada binding data dari fungsi bind() dipanggil saat RecyclerView nantinya menampilkan data di item tertentu terutama di dalam fungsi ini, nilai properti BookUi seperti title, release_date, dan author

diatur ke TextView. Jika tersedia, maka gambar sampul buku (cover) dimuat ke ImageView menggunakan library Glide. Berikutnya, onCreateViewHolder() dengan fungsi ini dipanggil oleh RecyclerView membatu dalam membuat ViewHolder baru karena di sini layout item_list.xml di-inflate melalui ViewBinding (ItemListBinding), dan sebuah instance BookViewHolder dikembalikan menjadikan tahap ini sebagai pembuatan tampilan tiap item list. Juga, ada getItemCount() yang mana fungsi ini mengembalikan jumlah item dalam daftar books. Nah, RecyclerView menggunakan nilai ini agar bisa mengetahui seberapa banyak data perlu ditampilkan di layar kepada user.

Selanjutnya, ada onBindViewHolder() menjadi fungsi yang dipanggil agar bisa mengisi data menuju ViewHolder sudah dibuat. Maksudnya fungsi bind() dipanggil dengan data buku di posisi tertentu untuk tampilan dapat menampilkan isi yang sesuai. Serta, ada updateBooks() merupakan fungsi sering digunakan untuk memperbarui seluruh daftar buku yang ingin ditampilkan. Dimana daftar yang lama dihapus (clear()), nantinya diganti dengan data baru (addAll(newBooks)), dan notifyDataSetChanged() dipanggil agar RecyclerView tahu bahwa seluruh datanya harus diperbarui.

Kesimpulan dari file MyAdapter ini tentang bagaimana cara implementasi adapter yang efisien dan modular dalam menampilkan daftar buku dengan memisahkan logika tampilan dan interaksi ke dalam ViewHolder serta menggunakan ViewBinding dan Glide agar kode menjadi lebih bersih, mudah dibaca, dan fleksibel untuk dikembangkan kedepaannya.

17. BookViewModel

Pada file BookViewModel merupakan bagian dari arsitektur MVVM yang bertugas menjembatani UI (fragment/activity) dengan domain layer (use case dan repository) mewarisi AndroidViewModel untuk mengakses Application context, yang digunakan saat membuat repository. Dimana ViewModel ini untuk

memuat data buku dari domain dan menyediakannya ke UI melalui LiveData. Nah, disini juga melakukan pembuatan dependency secara manual karena alih-alih kita menggunakan Dependency Injection (seperti Hilt), maka kode ini secara manual membuat instance BookRepositoryImpl menggunakan applicationContext, dan kemudian membuat instance GetBooksUseCase.

Kemudian, perlu melakukan deklarasi dan penggunaan LiveData karena ViewModel ini memiliki tiga LiveData utama seperti, _books untuk menyimpan daftar data buku nantinya ditampilkan di UI bertipe List<BookUi>, _isLoading menunjukkan apakah data sedang dimuat dengan biasanya untuk menampilkan atau menghilangkan ProgressBar, dan _errorMessage sebagai penyimpanan pesan kesalahan apabila terjadi error saat mengambil data. Dimana semua variabel LiveData disediakan ke UI dalam bentuk val agar hanya bisa dibaca dari luar dan tidak bisa dimodifikasi langsung dalam menjaga prinsip enkapsulasi.

Selanjutnya, ada fungsi fetchBooks() untuk memulai proses pengambilan data buku dengan cara isLoading di-set true sebagai pemberi sinyal bahwa proses sedang berjalan, lanjutannya viewModelScope.launch digunakan untuk menjalankan coroutine agar proses jaringan berlangsung secara asinkron tanpa memblok UI thread. Nah, di dalam coroutine getBooksUseCase() dipanggil untuk mengambil data dari domain layer yang hasilnya ditangani oleh fungsi onSuccess dan onFailure dalam kondisi jika berhasil, maka data dari domain layer diubah (mapped) menuju BookUi menggunakan toUiModel() yang dikirim ke UI melalui _books serta jika gagal, maka error message ditampilkan melalui _errorMessage. Setelah selesai, isLoading dikembalikan ke false agar progress indicator bisa disembunyikan.

Berikutnya, logging dan debugging karena di kode saya lebih tepatnya di file ini ada melakukan pemanggilan Log.d() dan Log.e() agar mencatat keberhasilan atau kegagalan pemanggilan data. Hal ini berguna untuk proses

debugging selama pengembangan aplikasi dan memberikan visibilitas tentang apa saja yang terjadi ketika fungsi `fetchBooks()` dijalankan.

Kesimpulan dari file `BookViewModel` menjadi pusat logika pada tampilan yang berperan dalam mengatur pengambilan dan penyediaan data buku menuju `UI`. Dengan adanya bantuan `LiveData` dan coroutine (`viewModelScope`) membuat `ViewModel` ini bisa menjaga `UI` tetap responsif, menangani loading atau error state, serta memisahkan logika tampilan dari logika data. Nah, pendekatan inilah yang membuat aplikasi lebih mudah diuji dan dirawat.

18. BookViewModelFactory

Pada file `BookViewModelFactory` berguna untuk membuat instance dari `BookViewModel`. Dimana `ViewModel` ini memiliki konstruktor dengan parameter (`Application`), sehingga tidak bisa langsung dibuat oleh sistem melalui `ViewModelProvider()` secara `default`. Oleh karena itu, kita perlu membuat `Factory` khusus disini untuk menangani pembuatan `BookViewModel` yang memerlukan argumen tambahan.

Kemudian, di file ini ada class yang mengimplementasikan interface `ViewModelProvider.Factory` yang mengharuskan kita untuk menulis `override` fungsi `create()` karena fungsi ini nantinya dipanggil saat sistem ingin membuat instance `ViewModel` dan kita bisa mengatur cara pembuatannya sendiri.

Lalu, ada implementasi fungsi `create()` untuk menerima parameter `modelClass`, yaitu class `ViewModel` yang ingin dibuat. Di dalam fungsi akan dilakukan pengecekan apakah `modelClass` cocok dengan `BookViewModel`. Jika cocok, maka dibuat dan dikembalikan instance `BookViewModel` dengan menyertakan application sebagai parameter konstruktor. Perlu diperhatikan karena casting dilakukan secara manual, maka saya disini menggunakan anotasi `@Suppress("UNCHECKED_CAST")` agar bisa menghindari peringatan kompilator. Tetapi ada kondisi jika tipe `ViewModel` yang diminta bukan

BookViewModel, maka nanti dilemparkan IllegalArgumentException sebagai bentuk validasi.

Selanjutnya, Perlunya Factory karena sangat penting dalam arsitektur MVVM Android saat ViewModel mempunyai dependensi konstruktor seperti Application atau objek lain (misalnya repository). Tanpa adanya factory ini, kita akan mendapatkan error saat mencoba membuat instance BookViewModel menggunakan ViewModelProvider() secara langsung.

Kesimpulan dari file BookViewModelFactory menjadi solusi di Android dalam hal menangani pembuatan ViewModel dengan konstruktor berparameter. Dengan menerapkan prinsip ini dapat menggunakan ViewModel yang lebih fleksibel dan sesuai dengan kebutuhan aplikasi saya buat terutama saat memerlukan konteks aplikasi atau dependensi lainnya ketika proses inisialisasi. Adanya Factory ini menjaga arsitektur aplikasi tetap modular dan testable.

19. Detail_fragment.xml

Pada file detail_fragment.xml ada struktur utama yaitu ConstraintLayout digunakan sebagai root layout dalam file ini karena memberikan fleksibilitas ketika saya ingin mengatur tata letak tampilan UI secara responsif. Dimana layout ini mengatur posisi elemen-elemen utama seperti Toolbar dan NestedScrollView sesuai constraint terhadap parent atau elemen lain, sehingga memungkinkan desainnya konsisten di berbagai ukuran layar. Lalu, ada elemen Toolbar berguna menjadi elemen paling atas sebagai action bar khusus untuk fragment detail ini, termasuk back button dan judul halaman. Dimana toolbar ini disesuaikan tampilannya dengan warna latar belakang dan tema AppCompat gelap, serta teks judul berwarna putih. Hal ini diterapkan dalam project ini untuk meningkatkan konsistensi navigasi dan pengalaman user saat berpindah antar halaman.

Kemudian, ada NestedScrollView untuk konten Scrollable yang mana bagian NestedScrollView akan membungkus seluruh konten di bawah

Toolbar dan memberi kemungkinan kepada user bisa menggulir atau scroll isi halaman jika terlalu panjang. Hal ini sangat penting agar deskripsi buku yang panjang tetap dapat dibaca seluruhnya tanpa mengganggu struktur visual antarmukanya. Nah, ScrollView ini diberi constraint agar menempel ke bagian bawah Toolbar dan mengikuti ukuran parent di sisi samping dan bawah.

Selanjutnya, ada LinearLayout sebagai kontainer vertikal karena di dalam NestedScrollView terdapat LinearLayout dengan orientasi vertikal. Dengan adanya layout ini untuk menempatkan ImageView (gambar cover buku) dan LinearLayout lain berisi teks-teks deskriptif (seperti judul dan deskripsi buku). Dimana struktur vertikal ini berguna untuk menjaga urutan elemen agar tampil dari atas ke bawah secara rapi. Berikutnya, ImageView untuk sampul buku digunakan untuk menampilkan gambar sampul buku dengan tinggi tetap 600dp melibatkan atribut scaleType="centerCrop" agar gambar menyesuaikan lebar tampilanlebih baik di layar. Juga, jangan lupakan peran tools:src yang hanya digunakan saat preview di Android Studio karema atribut ini memberi gambaran kepada saya sebagai developer tentang tampilan visual tanpa perlu memengaruhi runtime aplikasi ini.

Setelah itu, ada konten berupa teks berisi judul dan deskripsi terletak di bagian bawah terdapat LinearLayout berisi dua TextView. Dimana TextView pertama (detailTitle) digunakan untuk menampilkan judul buku dengan gaya Headline5 agar menonjol. Terus peran dari TextView kedua (detailDescription) digunakan untuk deskripsi buku karena diberi lineSpacingMultiplier agar teks lebih nyaman dibaca. Padding keseluruhan diberikan di container LinearLayout ini agar isi teks tidak menempel ke pinggir layar dan memberikan tampilan yang bagus membantu dalam membaca apa saja yang ada di aplikasi itu.

Kesimpulan file detail_fragment.xml dibuat untuk menampilkan tampilan detail dari sebuah buku dengan penekanan di bagian struktur yang fleksibel, scrollable, dan ramah pengguna. Dimana layout ini menyusun konten secara vertikal dengan gambar di atas dan teks di bawah, dipadukan dengan

Toolbar untuk navigasi, dan menjadikannya cocok bagi aplikasi berbasis Android dengan arsitektur fragment.

20. activity_main.xml

Pada file activity_main ada layout ini menggunakan ConstraintLayout sebagai root agar memberikan fleksibilitas dalam menyusun elemen-elemen UI dengan sistem constraint atau batasan antar elemen. Dimana layout ini mengatur ukuran lebar dan tinggi untuk mengisi seluruh layar perangkat (match_parent). Terdapat atribut tools:context menuju bagian MainActivity, sehingga layout ini nanti dipergunakan oleh activity tersebut dan android:background diatur ke tema default perangkat agar tampilan latar konsisten dengan sistem tersebut.

Kemudian, di dalam ConstraintLayout terdapat sebuah FrameLayout dengan ID fragmentContainer. Dimana elemen ini berguna sebagai tempat untuk menampilkan fragment, misalnya di sini HomeFragment atau DetailFragment nantinya ditambahkan dari kode MainActivity. Dengan ukuran FrameLayout dibuat 0dp untuk lebar dan tinggi, namun dikendalikan oleh constraint agar menempel ke keempat sisi parent (atas, bawah, kiri, kanan). Hal ini membuat fragment yang dimuat dapat mengisi seluruh area tampilan tersedia.

Selanjutnya, ada ProgressBar digunakan sebagai indikator loading ketika data sedang diambil dari API atau sumber lain. Letaknya diatur di tengah layar constraint ke semua sisi (top, bottom, start, dan end) dari parent layout. Default-nya bagian ProgressBar disembunyikan (android:visibility="gone") agar tidak selalu terlihat, tetapi bisa diatur tampil ketika aplikasi sedang dalam proses latar seperti pemanggilan data. Terdapat atribut tools:visibility="visible" hanya berguna untuk preview di Android Studio agar saya tetap bisa melihat posisi ProgressBar saat mendesain UI.

Kesimpulan file activity_main.xml merupakan layout utama dari MainActivity yang menyiapkan dua elemen inti seperti, FrameLayout sebagai kontainer dinamis untuk fragment, dan ProgressBar sebagai penanda proses loading. Dengan adanya struktur simpel namun fleksibel ini, maka MainActivity dapat berfungsi sebagai kerangka utama aplikasi secara dinamis menampilkan berbagai tampilan (fragment) dan memberi feedback visual saat data sedang melakukan proses.

21. home_fragment.xml

Pada file home_fragment.xml menggunakan ConstraintLayout sebagai root layout. Hal ini memungkinkan penempatan elemen UI lebih fleksibel menggunakan aturan constraint antar elemen. Dimana layout diatur agar mengisi seluruh layar (match_parent untuk lebar dan tinggi), dan atribut tools:context menunjukkan bahwa layout ini digunakan oleh HomeFragment. Dengan adanya pendekatan ini, UI dari fragment bisa tampil dengan layout konsisten dan responsif.

Kemudian, ada AppBarLayout dan Toolbar terdiri atas header fragment karena bagian atas layout berisi AppBarLayout dari material design berfungsi sebagai tempat untuk elemen toolbar yang di dalamnya terdapat Toolbar dengan ID homeToolbar menjadi header untuk fragment ini. Dimana Toolbar menampilkan nama aplikasi (melalui app:title="@string/app_name") dengan latar belakang mengikuti tema utama aplikasi (?attr/colorPrimary) dan teks berwarna putih agar kontras. Toolbar ini juga menggunakan tema Dark.ActionBar untuk memastikan tampilan sesuai aplikasi.

Selanjutnya, RecyclerView berisi daftar buku dengan ini bagian inti dari layout adalah RecyclerView dengan ID rv_character berguna untuk menampilkan daftar item buku atau karakter. Dimana RecyclerView ditempatkan tepat di bawah AppBarLayout dan memenuhi sisa ruang layar sampai ke bawah

(layout_constraintTop_toBottomOf, Bottom_toBottomOf, dll). Adanya RecyclerView ini menggunakan LinearLayoutManager (vertikal secara default) agar bisa menyusun item satu per satu ke bawah dan padding 4dp untuk jarak dari tepi layar. Melibatkan atribut tools:listitem="@layout/item_list" hanya digunakan saat melakukan pratinjau tampilan list di Android Studio menggunakan file layout item_list.xml.

Kesimpulan file home_fragment.xml untuk menyusun interface HomeFragment yang berfungsi sebagai halaman utama aplikasi. Dengan komponen Toolbar berguna bagi navigasi ataupun header, sedangkan RecyclerView di sini menampilkan daftar data seperti buku atau karakter yang diambil dari API. Struktur layout ini mendukung pemakaian fragment yang ringan, responsif, dan modular di aplikasi saya buat.

22. item_list.xml

Pada file item_list.xml menggunakan CardView untuk memberikan efek visual seperti kartu dengan sudut melengkung dan bayangan (elevation), sehingga tampilan daftar terlihat rapi. Dimana properti seperti cardCornerRadius="16dp" agar bisa memberikan efek melengkung, sementara cardElevation="4dp" menciptakan bayangan agar terlihat sedikit "terangkat" dari latar belakang. Melibatkan margin luar 8dp juga memberikan ruang antar kartu dalam daftar RecyclerView. Lalu, di dalam CardView terdapat ConstraintLayout sebagai kontainer utama agar bisa menyusun semua komponen UI secara efisien. Dengan padding sebesar 16dp agar tampilan dalam kartu mempunyai ruang yang baik dan terlihat proporsional. ConstraintLayout berguna dalam mengatur posisi relatif antar komponen (gambar, teks, dan tombol) presisi tinggi.

Kemudian, ada ImageView berfungsi saat ingin menampilkan gambar cover buku dengan ID imageView yang memiliki lebarnya diatur 100dp dan tingginya 150dp dengan scaleType="centerCrop" agar gambar memenuhi area tanpa

distorsi. Posisi gambar ditautkan menuju sisi kiri dan atas kontainer menggunakan constraint. Terdapat atribut tools:src hanya digunakan sebagai placeholder saat melihat preview di Android Studio. Berikutnya, ada TextView dengan ID textViewName berguna untuk menampilkan judul utama buku. Dimana letaknya berada di sebelah kanan ImageView dengan margin kiri 16dp sebagai pemisah visual. Adanya teks ini dibatasi hingga dua baris (maxLines="2") dan nanti dipotong jika terlalu panjang (ellipsize="end"), serta menggunakan teks tebal (textStyle="bold") dan ukuran Subtitle1 dari Material Design. Selanjutnya, ada dua TextView yaitu, textViewAuthor dan textViewYear berguna untuk menampilkan nama penulis dan tahun terbit buku secara berurutan di bawah judul. Dimana keduanya terletak tepat di bawah elemen sebelumnya menggunakan constraint Top_toBottomOf, sehingga konten teks tertata secara vertikal di sebelah gambar. Juga, textViewAuthor dibatasi hanya satu baris dan textViewYear menggunakan Caption agar lebih kecil dan bersifat informatif.

Setelah itu, ada bagian paling bawah dari layout yaitu, LinearLayout horizontal berisi dua tombol seperti buttonInfo dan buttonDetail. Dimana LinearLayout berada di sisi kanan bawah card dan menyesuaikan tombol ke kanan (gravity="end"). Tombol buttonInfo menggunakan TextButton agar tampil ringan, sementara buttonDetail memakai tombol standar dari *Material Components*. Nah, kedua tombol ini memungkinkan user nantinya bisa melihat info atau detail dari buku yang dipilih.

Kesimpulan file item_list.xml untuk mendefinisikan tampilan satu item (satu buku) dalam daftar RecyclerView. Dimana layout ini memadukan gambar cover , informasi dasar (judul, penulis, tahun), serta tombol menggunakan CardView. Struktur ini memastikan setiap item terlihat jelas, informatif, dan mudah untuk digunakan user.

23. nav_graph.xml

Pada file `nav_graph.xml` menggunakan elemen `<navigation>` sebagai `root` yang mendefinisikan struktur navigasi antar fragmen di dalam aplikasi. Dimana atribut `app:startDestination="@id/HomeFragment"` menentukan bahwa `fragment` pertama yang ditampilkan saat aplikasi dimulai itu `HomeFragment`. Dengan elemen ini bagian dari fitur *Jetpack Navigation Component* memudahkan manajemen navigasi antar tampilan dalam arsitektur berbasis `fragment`.

Kemudian, di sini `Fragment` pertama yang dideklarasikan itu `HomeFragment` dengan `ID` `@+id/HomeFragment` dan nama class `com.example.myapi_test.presentation.ui.HomeFragment` untuk tampilan utama aplikasi. Di dalamnya terdapat `<action>` dengan `ID` `action_HomeFragment_to_detailFragment` yang mendefinisikan transisi dari `HomeFragment` ke `DetailFragment` saat pengguna memilih salah satu item, misalnya dari daftar buku. Adanya elemen `<action>` ini sangat penting agar bisa memungkinkan ketika melakukan perpindahan antar tampilan sistem navigasi.

Selanjutnya, ada deklarasi `DetailFragment` dengan `ID` `@+id/detailFragment` dan nama class lengkap `com.example.myapi_test.presentation.ui.DetailFragment` menjadi tujuan dari aksi navigasi yang sebelumnya didefinisikan. Di dalam `fragment` ini terdapat tiga elemen `<argument>` yang masing-masing mendeskripsikan data yang harus diteruskan saat melakukan navigasi, yaitu, `imageResId` tipe datanya integer untuk menyampaikan resource `ID` gambar (misalnya sampul buku), `nama` bertipe string sebagai nama buku atau judul, dan `deskripsi` bertipe string untuk deskripsi lengkap buku. Nah, dari semua `argument` ini memungkinkan `DetailFragment` menampilkan konten sesuai data diteruskan dari `HomeFragment`.

Kesimpulan file nav_graph.xml ini menjadi peta navigasi yang berguna dalam mengatur perpindahan antar tampilan `fragment` di dalam aplikasi. Juga, dari file

ini menunjukkan bahwa pengguna memulai dari `HomeFragment` dan bisa berpindah menuju `DetailFragment` sambil membawa data penting seperti `gambar`, `nama`, dan `deskripsi`. Menggunakan *Jetpack Navigation Component* sangat membantu navigasi ini menjadi terstruktur dan mudah dikelola di satu tempat.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

https://github.com/alysaarmelia/AlysaArmelia_2310817120009_Pemrograman_Mobile/tree/815163e170f25c999ee932a4a23201e823f32a4e/PRAK_MODUL5