

Nama : John Isaac Witness

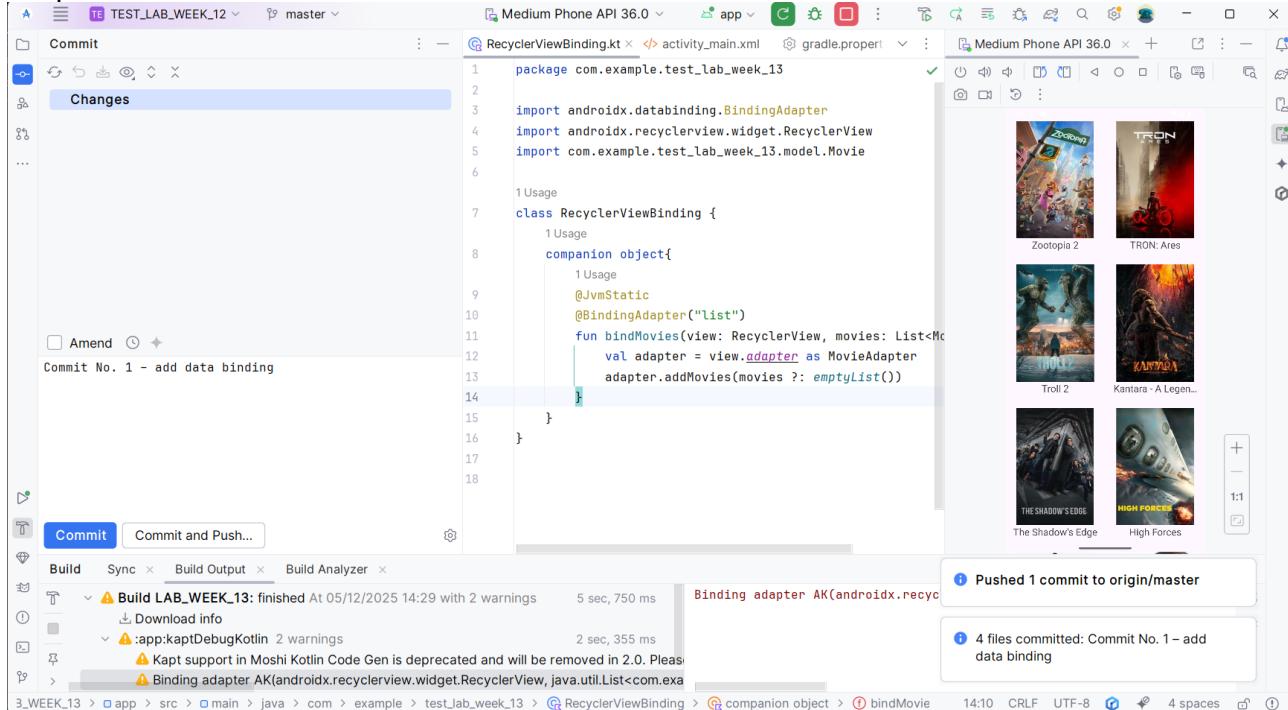
NIM : 00000088626

Mobile Application Programming Week 13

Link GitHub: https://github.com/akunjone/LAB_WEEK_13

Part 1 - Using Data Binding in Android

Output:



The screenshot shows the Android Studio interface. On the left, the 'Changes' tab in the Commit tool window displays the code for `RecyclerViewBinding.kt`. The code defines a `companion object` with a `bindMovies` function that takes a `RecyclerView` and a list of `Movie` objects, setting its adapter. On the right, the preview pane shows a grid of movie posters for "Zootopia 2", "TRON: Ares", "Troll 2", "Kantara - A Legen...", "The Shadow's Edge", and "High Forces". The bottom status bar indicates a successful build.

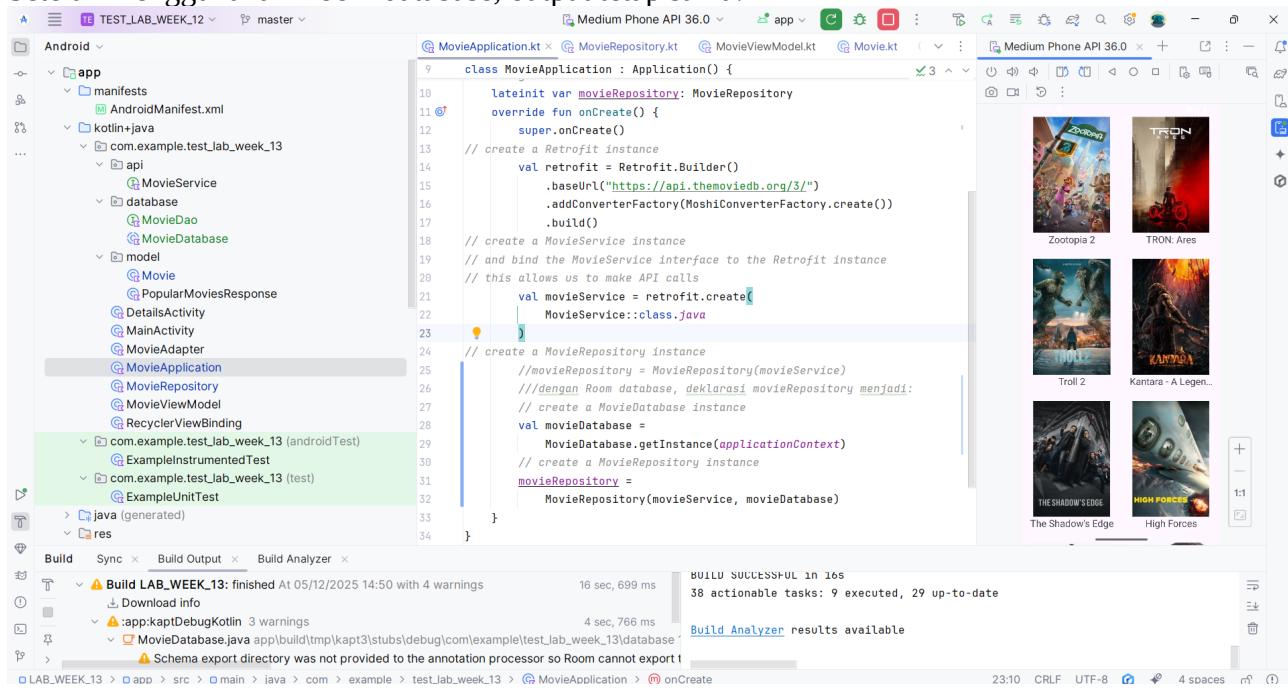
```
package com.example.test_lab_week_13

import androidx.databinding.BindingAdapter
import android.recyclerview.widget.RecyclerView
import com.example.test_lab_week_13.model.Movie

class RecyclerViewBinding {
    companion object {
        fun bindMovies(view: RecyclerView, movies: List) {
            val adapter = view.adapter as MovieAdapter
            adapter.addMovies(movies ?: emptyList())
        }
    }
}
```

Part 2 - Using Repository with Room

Setelah menggunakan Room database, output tetap sama:



The screenshot shows the Android Studio interface. On the left, the 'Android' tool window highlights the `MovieApplication` class. The code initializes a `MovieRepository`, creates a `Retrofit` instance, and binds the `MovieService` interface to it. It then creates a `MovieDatabase` and a `MovieRepository` instance using the `MovieDatabase`. On the right, the preview pane shows the same grid of movie posters as in the previous screenshot. The bottom status bar indicates a successful build.

```
class MovieApplication : Application() {
    lateinit var movieRepository: MovieRepository
    override fun onCreate() {
        super.onCreate()
        // create a Retrofit instance
        val retrofit = Retrofit.Builder()
            .baseUrl("https://api.themoviedb.org/3/")
            .addConverterFactory(MoshiConverterFactory.create())
            .build()
        // create a MovieService instance
        // and bind the MovieService interface to the Retrofit instance
        // this allows us to make API calls
        val movieService = retrofit.create(
            MovieService::class.java
        )
        // create a MovieRepository instance
        //movieRepository = MovieRepository(movieService)
        //dengan Room database, deklarasi movieRepository menjadi:
        // create a MovieDatabase instance
        val movieDatabase =
            MovieDatabase.getInstance(applicationContext)
        // create a MovieRepository instance
        movieRepository =
            MovieRepository(movieService, movieDatabase)
    }
}
```

Namun bedanya jika menggunakan Room, semua movie yang didapat disimpan dan dimasukkan kedalam Room. Sehingga jika koneksi terputus, movies tetap ada (namun tidak ter up-to-date dengan waktu saat ini).

Misal koneksi terputus (saya ubah sedikit API-key nya), maka akan tetap muncul movienya:

```

1 package com.example.test_lab_week_13
2
3 import android.app.Application
4 import com.example.test_lab_week_13.api.MovieService
5 import com.example.test_lab_week_13.database.MovieDatabase
6 import retrofit2.Retrofit
7 import retrofit2.converter.moshi.MoshiConverterFactory
8
9 class MovieApplication : Application() {
10    override fun onCreate() {
11        super.onCreate()
12        // create a Retrofit instance
13        val retrofit = Retrofit.Builder()
14            .baseUrl("https://api.themoviedb.org/3/")
15            .addConverterFactory(MoshiConverterFactory.create())
16            .build()
17
18        // create a MovieService instance
19        // and bind the MovieService interface to the Retrofit instance
20        // this allows us to make API calls
21        val movieService = retrofit.create(
22            MovieService::class.java
23        )
24
25        // create a MovieRepository instance

```

Build LAB_WEEK_13: finished At 05/12/2025 14:54 with 2 warnings
 Build SUCCESSFUL in 5s
 38 actionable tasks: 6 executed, 32 up-to-date
 Build Analyzer results available
 Install successfully finished in 775 ms

Part 3 - Using WorkManager

Project di Part 2 sudah hampir selesai, tapi database tidak bisa direfresh datanya ke versi terbaru dari API. Cara menyelesaikan masalah ini adalah dengan mengambil data dari API saat selang waktu tertentu atau saat user tidak membuka aplikasi. Dengan cara ini, data yang sudah dicache akan selalu up-to-date. Kita bisa melakukannya dengan mengatur WorkManager.

Output:

```

dependencies {
    implementation(libs.androidx.room.runtime)
    implementation(libs.androidx.room.ktx)
    kapt(libs.androidx.room.compiler)
    implementation(libs.androidx.recyclerview)
    implementation(libs.glide)
    implementation(libs.retrofit)
    implementation(libs.moshi.kotlin)
    implementation(libs.converter.moshi)
    implementation(libs.kotlinx.coroutines.core)
    implementation(libs.kotlinx.coroutines.android)
    kapt(libs.moshi.kotlin.codegen)
    implementation("com.squareup.moshi:moshi-kotlin:1.15.1")
    implementation(libs.androidx.lifecycle.livedata.ktx)
    implementation(libs.androidx.lifecycle.viewmodel.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.core.ktx)
}

```

Sama seperti sebelumnya, namun, database sudah ter up-to-date.

ASSIGNMENT

1. Why is MVVM important? Which files represent Model, which files represent View, and which files represent ViewModel?
2. In Part 1, you implemented Data Binding, why is this more efficient than using the normal method?

3. In Part 2, you implemented the Singleton Pattern, why is this important?
4. In Part 2 & 3, you implemented the Repository Pattern, why is this important?
5. In part 3, you implemented the Worker Manager, is there another way to refresh your database with the latest data other than using Worker?

ANSWERS FOR ASSIGNMENT

1. MVVM memisahkan UI dan logic bisnis. Jadi ketika kita ingin melakukan perbaikan untuk UI atau business logic, kita hanya perbaiki yang relevan saja. MVVM menjadi penting, karena ini membuat pekerjaan menjadi lebih mudah untuk menambah fitur baru, test kode baru, mengelola dan menjaga data lebih terarah juga. MVVM menjadi lebih baik juga karena strukturnya jelas dan tugas masing-masing juga jelas.
 - Yang menjadi Model adalah yang ada didalam folder “model”: Movie.kt dan PopularMoviesResponse.kt. Mereka menjadi struktur datanya (data layer).
 - View yang ada untuk layout aplikasi, untuk render data: MainActivity.kt, activity_main.xml, view_movie_item.xml, activity_details.xml.
 - ViewModel yang mengambil data dari Model dan menyediakannya ke View: MovieViewModel.kt
2. Data binding penting karena view model langsung berkomunikasi dengan views tanpa menggunakan yang seperti FindViewById, dll. Tujuannya untuk sinkronisasi secara otomatis antara View dan ViewModel, jadi ketika data pada satu sisi berubah, yang di sisi lainnya juga berubah.
3. Singleton pattern digunakan untuk deklarasi database. Dimana Singleton memastikan hanya ada satu instance database di beberapa thread. Tujuan singleton untuk memastikan data validity dan menghindari Race Condition. Singleton menjadi penting untuk mengurangi penggunaan resource yang mahal saat membuat instance database.
4. Repository pattern untuk menyinkronkan data antara API dengan Room (database). Repository ini memastikan data di database lokal selalu up-to-date dengan data yang ada di web. Ini penting di saat tidak ada internet, atau koneksi terputus, namun tetap membutuhkan info/sesuatu yang up-to-date.
5. Bisa menggunakan Coroutines saat aktif di foreground, Foreground Service untuk tugas yang diharuskan berjalan lama, JobScheduler, atau bisa menggunakan bantuan dari Firebase.

Terima kasih.