IMPLEMENTASI *CLEAN ARCHITECTURE* MVVM DAN *REPOSITORY PATTERN* UNTUK PENGEMBANGAN APLIKASI ANDROID JUAL BELI BARANG BEKAS "SECONDHAND"

Firmansyah Firdaus Anhar, Fetty Tri Anggraeny
Program Studi Informatika, Universitas Pembangunan Nasional "Veteran" Jawa Timur
Email: 19081010031@student.upnjatim.ac.id

Abstrak. *Pengembangan aplikasi android sudah dilakukan cukup lama, dalam pengembangannya, tercipta berbagai macam model arsitektur yang diterapkan. Aplikasi android semakin canggih dengan berkembanganya jaman, banyak fitur dan hal baru yang harus memaksa berbagai jenis arsitektur android untuk beradaptasi dengan perubahan sistem, bagaimana membuat arsitektur yang ramah terhadap sistem android, hingga arsitektur yang mempermudah penggunanya dalam menulis sebuah kode yang bersih. Model View ViewModel atau yang biasa disingkat MVVM merupakan salah satu desain arsitektur yang mulai sering digunakan dalam pengembangan aplikasi mobile khususnya android. MVVM memiliki beberapa tujuan, seperti mempermudah dalam penulisan kode dan melakukan testing, hingga bisa bertahan dalam perubahan konfigurasi pada aplikasi. Penerapan MVVM juga bisa diikuti dengan penerapan repository pattern, di mana arsitektur ini bertujuan sebagai jembatan penghubung untuk sumber data yang berbeda pada satu aplikasi. Penerapan dua arsitektur ini akan menghasilkan sebuah aplikasi yang kokoh dari segi bahasa, mudah dalam melakukan uji coba, dan sudah ramah terhadap sistem operasi android terbaru.*

Kata Kunci: MVVM, Repository Pattern, Android

Pengembangan aplikasi android sudah dilakukan cukup lama, dalam pengembangannya, tercipta berbagai macam model arsitektur yang diterapkan. Sistem operasi android sudah semakin berkembang sampai pada titik di mana diperlukan sebuah arsitektur baru yang dapat mendukung kinerja sistem operasi android dengan baik. Arsitektur bisa dikatakan sebagai arsitektur bersih jika dapat mengikuti satu aturan: *level* internalnya, sama tidak perlu mengetahui isi dari level eksternalnya. Pada kasus pengembangan android, *level* internal yang dimaksud adalah logika bisnisnya, sementara *level* eksternal adalah antarmukanya, sehingga keterhubungan antara dua *level* ini hanya sebatas pada perpindahan data [1].

Salah satu arsitektur yang pernah populer sebelum ini adalah Model View Presenter (MVP). Arsitektur ini merupakan sebuah varian dari arsitektur populer dalam pengembangan web yaitu MVC (Model View Controller). MVP membawa kelebihan dibandingkan dengan MVC, yaitu pemisahan *layer domain* dengan antarmuka yang lebih baik [2]. MVP memiliki tiga level abstraksi, hal ini dapat memudahkan dalam melakukan *debugging* pada aplikasi [4]. MVP sejatinya telah mengikuti salah satu aturan yang ada pada arsitektur bersih, yaitu setiap komponen pada aplikasi harus harus memiliki hanya satu tanggung jawab. Dengan berkembangnya skala suatu komponen, maka harus dilakukan pemisahan. Dengan mengikuti prinsip satu tanggung jawab ini, secara alami arsitektur akan mendukung pengembang dan mempermudah dalam uji coba sebagai hasil dari pemisahan tersebut [3].

Model View ViewModel atau yang biasa disingkat MVVM merupakan salah satu desain arsitektur yang mulai sering digunakan dalam pengembangan aplikasi mobile khususnya android. MVVM memiliki beberapa tujuan, seperti mempermudah dalam penulisan kode dan melakukan testing, hingga bisa bertahan dalam perubahan konfigurasi pada aplikasi. MVVM merupakan hasil evolusi dari arsitektur sebelumnya yaitu MVP [2]. Dalam evolusinya, MVVM bertujuan untuk membuat arsitektur menjadi lebih terpisah antara antarmuka dengan *domain layer*-nya [2]. Salah satu komponen dari MVVM, yaitu ViewModel,

berperan sebagai pengantar data kepada antarmuka untuk ditampilkan, kendati demikian, sebuah ViewModel tidak seharusnya mengetahui isi dari antarmukanya.

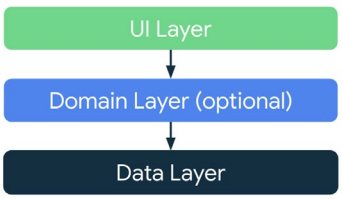
Arsitektur MVVM telah mendapat dukungan secara langsung dari Google sebagai salah satu arsitektur yang direkomendasikan. Dukungan ini dibuktikan dengan adanya sebuah fitur yang secara otomatis terbangun dengan penggunaan arsitektur ini, yaitu kebertahanan aplikasi pada perubahan konfigurasi saat aplikasi berjalan. Penerapan MVVM terkadang juga diikuti dengan

I. Metodologi

Terdapat dua metode yang digunakan, yaitu Model View ViewModel sebagai arsitektur utama bertujuan selain untuk penulisan kode yang lebih baik, sekaligus juga untuk mempertahankan kondisi aplikasi ketika terjadi sebuah perubahan konfigurasi. Arsitektur yang kedua adalah *repository pattern*, SecondHand memiliki dua sumber data yang berbeda, yaitu data lokal dan data *remote*, sehingga aplikasi ini bisa mengimplementasikan *repository pattern* sebagai penghubung antara dua sumber data yang berbeda.

Analisa

Pada fase ini dilakukan analisa untuk mendapatkan kebutuhan yang sesuai dengan pengembangan aplikasi. Analisa dilakukan dengan melihat arsitektur apa yang umum digunakan dalam pengembangan aplikasi android. Arsitektur yang umum digunakan adalah arsitektur yang setidaknya memiliki dua lapis, yaitu lapisan tampilan dan lapisan data [5].



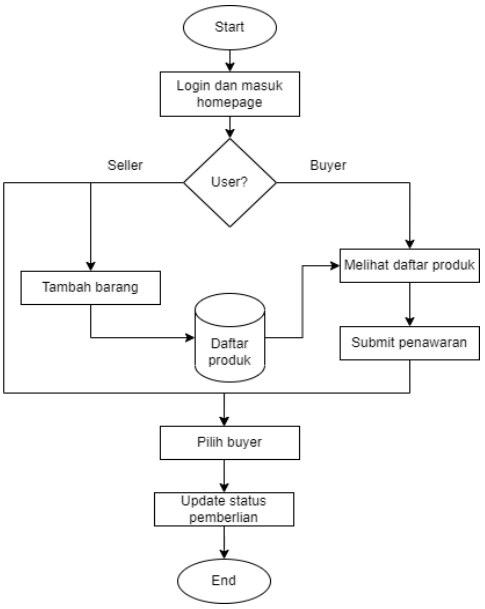
Gambar 1. Arsitektur umum pengembangan aplikasi

penerapan repository pattern, repository pattern adalah sebuah desain arsitektur dalam pengembangan software, di mana repository pattern bertujuan untuk memisahkan data layer dari keseluruhan aplikasi, sehingga bagian lain akan 'buta' terhadap sumber data yang diberikan. Aplikasi SecondHand mengimplementasikan dua arsitektur ini dengan ekspektasi bahwa pengembangan aplikasi menjadi lebih terstruktur dan tambahan fitur yang membuat aplikasi semakin kokoh dan sesuai dengan sistem operasi android terbaru

Analisa kedua dilakukan dengan melihat fitur yang akan diadopsi dalam aplikasi SecondHand. Aplikasi SecondHand mengambil data dari sebuah database yang berasal dari internet, sehingga bisa disebut sebagai data *remote*, dan SecondHand juga memiliki fitur riwayat pencarian yang disimpan pada masing masing perangkat, sehingga bisa disebut data lokal. Dengan mempertimbangkan dua sumber data yang berbeda ini, akhirnya ditentukan arsitektur baru sebagai pendukung MVVM yaitu *repository pattern.*

Perancangan Aplikasi

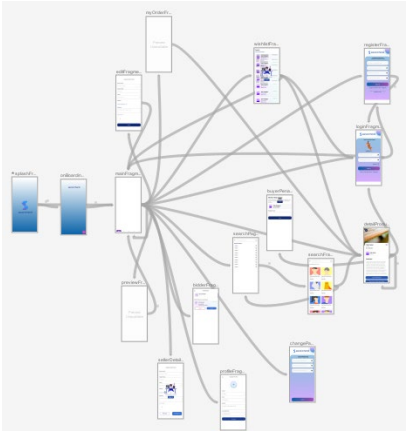
Dalam perancangan aplikasi SecondHand, dilakukan pembuatan *flowchart* terkait alur aplikasi SecondHand



Gambar 2. Alur bisnis aplikasi SecondHand

SecondHand memiliki dua *role* yaitu pembeli dan penjual, pembeli dapat mengunggah produk yang diinginkan, sementara penjual dapat melihatnya dengan bebas. Setiap penawaran pembeli, penjual akan mendapat sebuah notifikasi atas penawaran tersebut, penjual dapat memilih untuk menerima atau menolak tawaran dari pembeli tersebut.

Kedua, dilakukan pembuatan diagram navigasi terkait alur halaman aplikasi, pembuatan diagram dilakukan dengan menggunakan Android Studio menggunakan *navigation component* sebagai salah satu *tools* dalam pengembangan aplikasi android.



Gambar 3. Navigasi aplikasi SecondHand

Saat aplikasi pertama kali dibuka, akan secara otomatis membuka halaman utama, sehingga user tidak diharuskan melakukan *login* pada aplikasi, untuk mengakomodasi jika ada user yang tidak ingin membuat akun namun masih ingin melihat barang barang yang ada, namun untuk mengakses fitur fitur yang membutuhkan sebuah akun, maka user akan diwajibkan *login* terlebih dahulu sebelum melanjutkan.

Testing

Uji coba aplikasi dilakukan dengan dua cara, yaitu menggunakan *unit testing* dan *instrumental testing*. *Unit testing* adalah sebuah metode uji coba dengan menguji masing masing fungsi yang ada pada aplikasi, sehingga disebut *unit*. *Instrumental testing* merupakan sebuah metode uji coba dengan menguji alur sebuah aplikasi, biasanya dibagi menjadi beberapa skenario.

Unit test dilakukan pada seluruh repository yang ada pada aplikasi SecondHand, yaitu

1. Home Repository
2. Login Repository
3. Register Repository
4. Order Repository
5. Product Repository
6. Notification Repository
7. Search History Repository

Masing masing *repository* berisi kode logika bisnis terkait alur bisnis SecondHand yang kemudian diuji coba.

Instrumental test dilakukan dengan membuat tujuh skenario aplikasi, yaitu

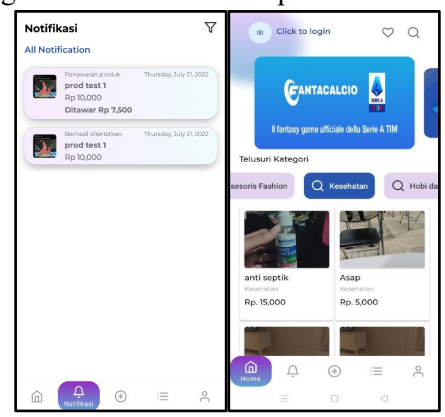
1. Login Register
2. Logout
3. Ubah password
4. Melihat order
5. Pencarian produk
6. Menambahkan produk
7. Submit tawaran

masing masing skenario akan diuji apakah alur yang diinginkan sesuai dengan keluaran yang terjadi pada proses uji coba, proses ini akan dilakukan berulang kali hingga hasil uji coba mendapat keluaran sesuai yang diinginkan.

II. Hasil dan Pembahasan

Implementasi Antarmuka

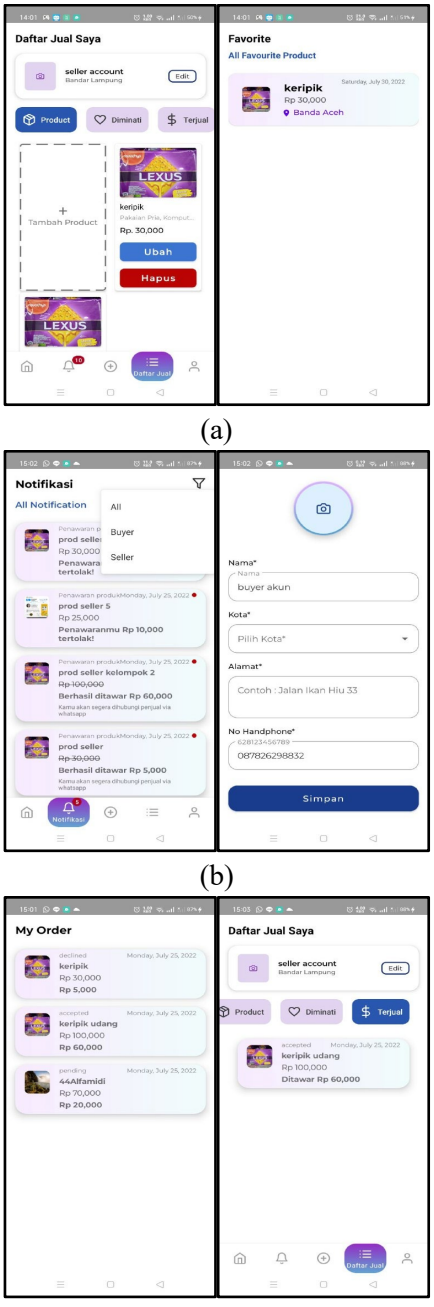
Aplikasi SecondHand membuat tampilannya menggunakan bahasa XML sebagai *markup language* dan Kotlin sebagai bahasa pemrogramannya, dengan menghasilkan 22 halaman aplikasi.



Gambar 4. Tampilan home dan notifikasi aplikasi SecondHand

Dalam implementasinya, aplikasi SecondHand menerapkan beberapa fitur tambahan di luar alur bisnis sebagai penunjang pengalaman pengguna, seperti

1. Tambah produk ke favorit
2. Pencarian produk berdasarkan kategori
3. Mengubah informasi akun
4. Melihat *banner* sebagai media promosi
5. Filter notifikasi
6. Melihat histori penjualan



(a)

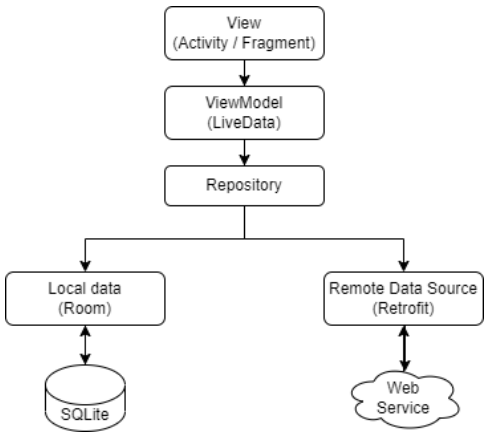
(b)

(c)

Gambar 5 (a-c). Fitur tambahan di luar alur bisnis

Implementasi MVVM dan *Repository Pattern*

Dalam implementasinya, aplikasi SecondHand menerapkan arsitektur MVVM dan *repository pattern* yang dapat direpresentasikan seperti diagram berikut



Gambar 6. Diagram alur MVVM dan *repository pattern*

View yang berperan sebagai tampilan luar yang berupa halaman aplikasi, meminta data untuk diolah atau ditampilkan dari sebuah *viewModel,* lalu *viewModel* sebagai penghubung antara data dengan tampilan, akan meminta sebuah data kepada *repository,* yang di dalamnya terdapat dua sumber data, yaitu dari *web service* dan dari data lokal. *viewModel* sebagai perantara data tidak perlu mengetahui sumber data yang diterimanya, sehingga *viewModel* dapat mengolahnya dengan setara. Pada pengembangan android umumnya, sumber data lokal yang digunakan adalah SQLite dengan bantuan pustaka Room, sementara untuk sumber data *remote* bisa berasal dari Rest API, dengan bantuan pustaka Retrofit sebagai penghubung antara aplikasi dengan *web service* yang ada.

Bagaimana sebuah *view* bisa memanggil data dari sebuah viewModel jika tidak ada keterkaitan antara keduanya? Pada tahun 2015, pada Google I/O Developer Conference diluncurkan sebuah pustaka bernama Data Binding sebagai penghubung antara sebuah data tampilannya [6]. Data Binding akan secara otomatis mengisi tampilan sesuai dengan viewModel yang diterapkannya, sehingga

dapat mengurangi baris kode dan mempermudah pengembangan aplikasi [6].

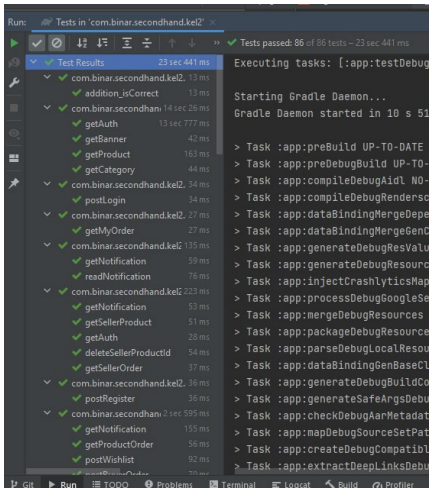
Implementasi Uji Coba

Implementasi dilakukan dengan menggunakan tiga pustaka yang umum disediakan dan digunakan dalam pengembangan aplikasi android. Pertama adalah JUnit dan Mockito sebagai pustaka yang digunakan dalam pembuatan *unit test* dengan kode sebagai berikut

```
@Test
fun getProducts()= runTest{
    val getProductResponseItem =
        mock<Flow<PagingData<GetProductResponseItem>>>()

    given(repository.getProductStream())
        .willReturn(getProductResponseItem)

    viewModel.getProducts()

    advanceUntilIdle()

    Mockito.verify(repository, Mockito.times(1))
        .getProductStream()
}
```

Gambar 7. Kode *unit test* untuk daftar produk

Hasil uji coba didapatkan total dari 89 fungsi yang diuji, semuanya tidak mengalami error



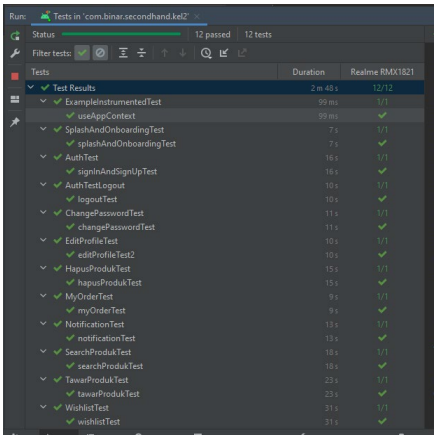
Gambar 8. Hasil uji coba *unit test repository*

Kedua, dalam uji coba *instrumental*, digunakan pustaka Espresso dengan kode seperti berikut

```
@Test
fun tawarProdukTest() {
    val appCompatImageView = onView(···
    )
    appCompatImageView.perform(click())

    val textInputEditText3 = onView(···
    )
    textInputEditText3.perform(···
    )
    val recyclerView = onView(···
    )
    recyclerView.perform
        (actionOnItemAtPosition<ViewHolder>(0, click()))
    val appCompatButton2 = onView(···
    )
    appCompatButton2.perform(click())
    val textInputEditText4 = onView(···
    )
    textInputEditText6.perform(closeSoftKeyboard())
    val materialButton2 = onView(···
    )
    materialButton2.perform(click())
}
```

Gambar 9. Kode *instrumental test* untuk submit penawaran

Hasil uji coba *instrumental* dengan total dua belas skenario, dihasilkan keluaran seperti berikut



Gambar 10. Hasil luaran *instrumental test* aplikasi SecondHand

Seluruh skenario uji coba telah mendapatkan hasil keluaran yang diinginkan sehingga semua skenario mendapat centang hijau tanda semua parameter terpenuhi dengan baik.

III. Kesimpulan

Berdasarkan hasil analisis, implementasi, hingga uji coba yang dilakukan terhadap aplikasi SecondHand, dapat disimpulkan bahwa penerapan arsitektur MVVM dan penerapan pola repositori, dapat membantu dalam penulisan kode yang lebih terstruktur, dan pembuatan skenario dan *test*

case yang menjadi lebih mudah dengan adanya pemisahan dalam layer aplikasi. Pemisahan layer aplikasi yang diusung dalam arsitektur ini memudahkan dalam melakukan uji coba, karena uji coba bisa dilakukan secara terpisah terhadap datanya dan terhadap tampilannya, yang dipisahkan dalam *unit test* dan *instrumental test.*

IV. Daftar Pustaka

[1] Krasko, R., & Oskin, A. (2020). Clean Architecture For Android Applications. MATERIALS OF XII JUNIOR RESEARCHERS' CONFERENCE, UDC 004.41.

[2] Daoudi, A., ElBoussaide, G., Moha, N., & Kpodjedo, S. (2019). An Exploratory Study of MVC-based Architectural Patterns in Android Apps. 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19).

[3] Verdecchia, R., Malavolta, I., & Lago, P. (2019). Guidelines for Architecting Android Apps: A Mixed-Method Empirical Study. IEEE International Conference on Software Architecture (ICSA)

[4] Humeniuk, V. (2019). Android Architecture Comparison: MVP vs. VIPER. Linne Universitetet

[5] Google. (2022). Guide To App Architecture. Retrieved from https://developer.android.com/topic/architecture

[6] Sun, W., Chen, H., & Yu, W. (2016). The Exploration and Practice of MVVM Pattern on Android Platform. Advances in Computer Science Research, volume 71. 4th International Conference on Machinery, Materials and Information Technology Applications (ICMMITA 2016)