BAB II
TINJAUAN PUSTAKA

Pada bagian ini, akan dikemukakan teori-teori yang akan mendukung proses penelitian serta karya-karya ilmiah yang terkait dengan penelitian ini.

II.1 Dasar Teori

II.1.1 Android

Android adalah sebuah sistem operasi untuk perangkat mobile berbasis linux yang mencakup sistem operasi, middleware, dan aplikasi. Di dunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari Google atau Google Mail Services (GSM) dan kedua adalah yang benar-benar bebas distribusinya tanpa dukungan langsung Google atau dikenal sebagai Open Handset Distribution (OHD). Pada saat ini kebanyakan vendor-vendor smartphone sudah memproduksi smartphone berbasis android, antara lain HTC, Motorola, Samsung, LG, Sony Ericsson, Acer, Nexus, Nexian, IMO, dan masih banyak lagi vendor smartphone di dunia yang memproduksi android. Hal ini karena android itu adalah sistem operasi yang open source sehingga bebas didistribusikan dan dipakai oleh vendor manapun [7].

II.1.2 Aplikasi Native

Aplikasi Native adalah aplikasi smartphone yang dibangun dengan menggunakan bahasa pemrograman yang spesifik untuk platform tertentu. Contoh populernya yakni penggunaan bahasa pemrograman Object-C atau Swift untuk platform IOS (Apple). Adapun platform Android yang menggunakan bahasa pemrograman Java atau Kotlin. Salah satu keunggulan dari aplikasi Android native adalah memberikan performa yang cepat dan tingkat ke andalan yang tinggi, dan juga memiliki akses ke berbagai perangkat ponsel, seperti kamera,bluetooth, dll. Namun, jenis aplikasi native terbilang mahal untuk dikembangkan karena terkait dengan satu jenis sistem operasi saja, memaksa perusahaan yang membuat aplikasi untuk membuat versi duplikat yang bekerja pada platform lain [6].

II.1.3 Konsep Android Native

Pada pengembangan aplikasi berbasis android native ada tiga konsep paling penting dalam tugas akhir ini diataranya adalah *activity, fragment, resource* [6].

1. *Activity* adalah satu hal yang terfokus yang dapat dilakukan oleh pengguna. Interaksi pengguna dengan aplikasi sebagian besar dilakukan melalui *activity*. Singkatnya, *activity* adalah layar penuh yang ditampilkan di perangkat.
2. *Fragment* adalah bagian dari antarmuka pengguna aplikasi atau perilaku yang dapat ditempatkan dalam *activity*. Ini pertama kali diperkenalkan di Android 3.0 (Android 11). Dari definisi, fragment mirip dengan *activity*. Perbedaan utamanya adalah bahwa *activity* adalah layar penuh sementara *fragment* adalah bagian dari layar. *Fragment* membuat UI aplikasi lebih fleksibel. Misalnya, di tablet, layar biasanya dibagi menjadi dua bagian: kiri dan kanan. Kemungkinan besar, bagian kiri dan kanan adalah dua *fragment* yang melekat pada suatu *activity*. Fragment melekat pada *activity*, namun, *life cycle* jauh lebih kompleks dari pada *activity*.
3. *Resource* adalah file tambahan dan konten statis yang akan di integrasikan dengan kode java. Ada berbagai jenis resource diantaranya adalah *layout, bitmaps, dimension, translation, animation* dan lain-lain. *Resource* yang paling penting dalam pembuatan aplikasi android adalah *layout* yang menentukan *User Interface* yang akan berinteraksi dengan penguna. File *layout* berada dalam format XML yang mengatur beberapa widget (*button, text, checkbox*), Posisi dan tampilan.

Konsep tersebut pada penelitian ini digunakan untuk tahap pengembangan aplikasi yang harus paham konsep android native sebelum masuk ke arsitektur android.

II.1.4 Performa

Performa dapat memiliki banyak arti yang berbeda untuk setiap orang. Ketika berkaitan dengan aplikasi mobile, performa dapat menggambarkan cara kerja aplikasi, seberapa efisien aplikasi bekerja, atau seberapa nyaman aplikasi digunakan [8].

Performa adalah faktor penting dalam keberhasilan aplikasi. Untuk mengevaluasi performa aplikasi android, ada beberapa faktor yang dipertimbangkan. Corral *et al* [11]. Meyakini bahwa penggunaan CPU (*CPU Usage*), waktu eksekusi (*execution time*), konsumsi baterai *(battery consumption*) dan penggunaan memori (*memory* usage) adalah faktor kunci untuk performa aplikasi Android. Untuk mengevaluasi waktu eksekusi pada aplikasi native dan aplikasi web, Corral mengembangkan aplikasi untuk melakukan tugas yang berbeda: menulis/membaca file, meminta data, dll. Hasilnya ternyata performa aplikasi native lebih baik daripada aplikasi web. Situs web resmi android menyarankan bahwa faktor-faktor performa dalam aplikasi android berisi waktu merespon, konsumsi baterai, penggunaan memori, penggunaan CPU, dan penggunaan GPU [11].

II.1.4.1 *Performance Testing*

Performance testing adalah teknik pengujian non-fungsional yang dilakukan untuk menentukan parameter sistem seperti kecepatan atau stabilitis sistem. Banyak faktor yang mempengaruhi performa sebuah perangkat lunak. Dalam pengujian perangkat lunak, *metrics* adalah ukuran kuantitatif dari tingkatan suatu sistem, komponen sistem, atau attribut proses. *Metrics* diperlukan untuk memahami kualitas dan efektifitas pengujian performa. Beberapa *metrics* yang sering digunakan dalam pengujian performa adalah sebagai berikut :

- *Execution time*
- *Wait time*
- *Average load time*
- *Request per second*
- *CPU utilization*
- *Memory utilization*

Performa mengukur seberapa cepat aplikasi dijalankan, seberapa cepat memuat data, dan konektifitas keseluruhan aplikasi pada operator yang berbeda [5]. Pada penelitian ini, *metrics* aplikasi yang dibandingkan adalah *CPU usage*, *memory usage* dan *execution time*.

II.1.4.2 Tools Profiling Performa Aplikasi

Tools yang digunakan untuk melakukan pengukuran ada dua *tools* diantaranya adalah *Snapdragon Profiller* dan Android Profiler. *Snapdragon Profiler* adalah perangkat lunak yang berjalan pada platform Windows, Mac, dan Linux. Ini terhubung dengan perangkat android yang didukung oleh *processor Snapdragon* melalui *USB*. *Snapdragon Profiler* memungkinkan pengembang untuk menganalisis data *CPU*, *GPU*, *DSP*, memori, daya, termal, dan jaringan, sehingga dapat menemukan dan memperbaiki *bug* pada aplikasi dan data hasil profiling dapat di eksport ke dalam CSV sehingga mudah untuk di analisis. Sedangkan Android Profiler adalah perangkat lunak yang berjalan hanya pada IDE Android studio dengan fitur untuk memonitoring CPU, memory, network, dan baterai. Akan tetapi kekurangan dari *tools* ini tidak bisa mengeksport ke CSV. Sehingga pada penelitian ini *tools* yang dipakai adalah *Snapdragon Profiler* karena sudah dapat mengeksport ke CSV dan juga tersedia fitur untuk melihat waktu eksekusi aplikasi.

II.1.4.3 *CPU Usage*

CPU usage berkaitan dengan penggunaan sumber daya pemrosesan komputer, atau jumlah pekerjaan yang ditangani oleh *CPU*. Menurut [3], *CPU usage* dipengaruhi oleh jaringan, layar, dan semua perhitungan yang terjadi pada perangkat android. Meminimalkan penggunaan *CPU* atau penggunaan *CPU* yang kecil akan memberikan banyak keuntungan bagi pengguna seperti *user experience* yang lebih cepat dan menghemat penggunaan baterai perangkat. Pada penelitian ini *CPU usage* atau penggunaan *CPU* akan dijadikan sebagai variabel untuk melihat seberapa banyak penggunaan *CPU* yang digunakan.

II.1.4.4 *Memory Usage*

Memory berfungsi untuk membantu *processor* dalam menyimpan data dan informasi yang bersifat sementara. Dari perspektif *performance*, membaca dan menulis ke memori jauh lebih efisien dan cepat dibandingkan membaca dan menulis ke *disk*. *Memory usage* menunjukkan seberapa besar aplikasi mengonsumsi *memory* pada perangkat android. Android memiliki mekanisme perlindungan sendiri pada memori yang disebut *LMK* (*Low Memory Killer*) [4].

Ketika terlalu banyak memory yang digunakan, LMK akan menutup proses yang tidak aktif yang mengonsumsi banyak memori. Sehingga hal yang terjadi apabila *memory* penuh adalah aplikasi mengalami perlambatan atau aplikasi akan ditutup paksa (*force close*). Beberapa tindakan yang dapat mempengaruhi konsumsi *memory* adalah ukuran APK, *library* pihak ketiga atau *resource* yang di masukkan ke aplikasi [17].

II.1.4.5 *Execution Time*

Execution time adalah waktu yang dibutuhkan aplikasi dalam menjalankan/mengeksekusi suatu instruksi program [3]. Penulisan kode yang berbeda dapat mempengaruhi *execution time*. Dalam penelitian ini *execution time* yang diukur adalah seberapa cepat aplikasi dalam mengolah data yang diperoleh dari *local database* menjadi objek-objek pada aplikasi hingga menampilkan hasil tersebut menjadi informasi [4].

II.1.4.6 Snapdragon Profiler

Snapdragon profiler adalah *tools* yang digunakan untuk me-monitoring *CPU* dan *memory usage* pada perangkat android. *Tools* dipilih karena dapat menampilkan *resource* yang dikonsumsi oleh aplikasi yang di-*monitoring* saja tanpa dipengaruhi aplikasi lain. Kemudian aplikasi ini juga dapat memperlihatkan penggunaan konsumsi *resource* persatuan waktu, sehingga sangat mudah untuk memeriksa dan mendapatkan angka yang dibutuhkan. *Tools* ini dapat diperoleh dari *website* resminya Qualqomm dengan *url* https://developer.qualcomm.com/ software/snapdragon-profiler.

II.1.5 Bahasa Pemrograman Android Native

Pada pengembangan aplikasi berbasis android native, ada dua bahasa pemrograman yang dapat digunakan dan menjadi bahasa yang cukup populer saat ini yaitu java dan kotlin

II.1.5.1 Java

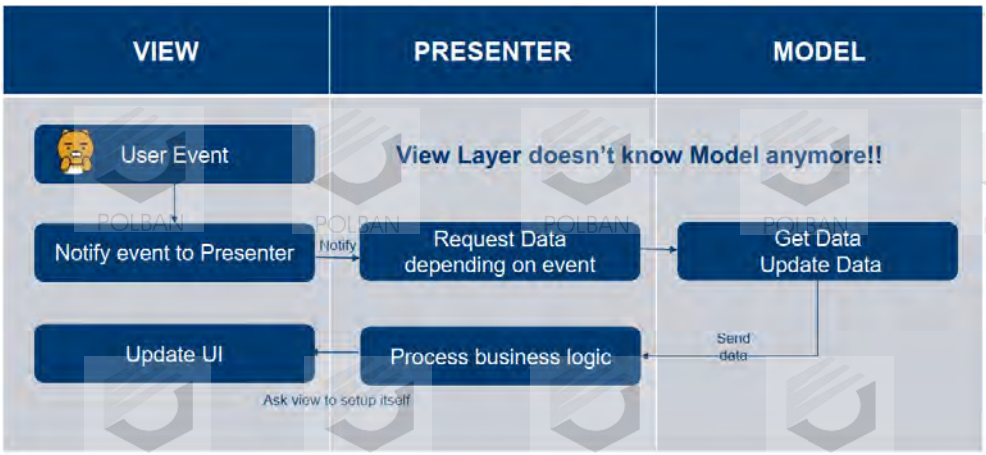
Java adalah bahasa pemrograman yang sangat populer sejak tahun 1995 [7]. Bahasa Java dapat dijalankan di berbagai komputer dan perangkat *mobile*. Bahasa

pemrograman ini sudah digunakan pada banyak pengembangan aplikasi berbasis android. Bahasa pemrograman ini menganut *Object Oriented Paradigm* yang akan menjadi salah satu pilihan dalam pengembangan aplikasi yang akan dijadikan sebagai objek penelitian.

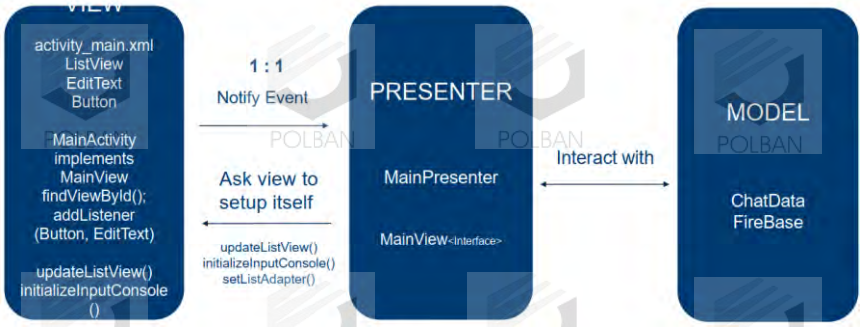
II.1.6 Arsitektur MVP

MVP atau singkatan dari Model View Presenter adalah sebuah konsep arsitektur pengembangan aplikasi yang dikembangkan oleh Google bertujuan untuk memperbaiki arsitektur MVC yang memiliki banyak masalah pada saat pengembangan dan pada saat aplikasi rilis produksi. Arsitektur ini memisahkan antara tampilan aplikasi dengan proses bisnis yang bekerja pada aplikasi. Arsitektur inidi rancang berdasarkan dua pertanyaan : manajemen data dan UI yaitu, cara mengolah data dan bagaimana pengguna berinteraksi dengan data. Sebenarnya pada arsitektur MVP terdapat enam layer yang masing-masing memiliki tugas yaitu *model, selections, commands, presenter, interactor* dan *view.*

1. *Model* pada arsitektur MVP memiliki peran sama seperti pada arsitektur MVC yang merupakan *layer* yang menunjuk kepada objek dan data yang ada pada aplikasi. *Selection* berupa subnet data yang akan di operasikan contoh seperti teks yang akan di blok (*highligted text*). *Commands* berperan untuk menyajikan perintah / *action* yang dapat dilakukan seperti *copy, paste,cut,* dan lain-lain. *Iterator* berperan untuk medeteksi aksi/perilaku yang di *trigger* oleh user seperti pergerakan *mouse, click, keyboard* dan lain-lain.
2. *Presenter* layer yang berperan hampir sama seperti *controller* dalam MVC yang bertujuan untuk mengatur dan mengkoordinasikan semua komponen *interactor, selections and commands.* dan untuk lebih spesifiknya *presenter* merupakan *layer* yang menghubungkan komunikasi antara *model* dan *view*. Setiap interaksi yang dilakukan oleh pengguna akan memanggil *presenter* untuk memrosesnya dan mengakses *model* lalu mengembalikan responnya kembali kepada *view*.
3. *View* adalah sebuah *layer* pada sama seperti pada MVC yang mengandung keseluruhan detail dari implementasi *user interface* yang akan langsung berinteraksi dengan pengguna.



Gambar II.1 Ilustrasi Arsitektur MVP secara umum [6]

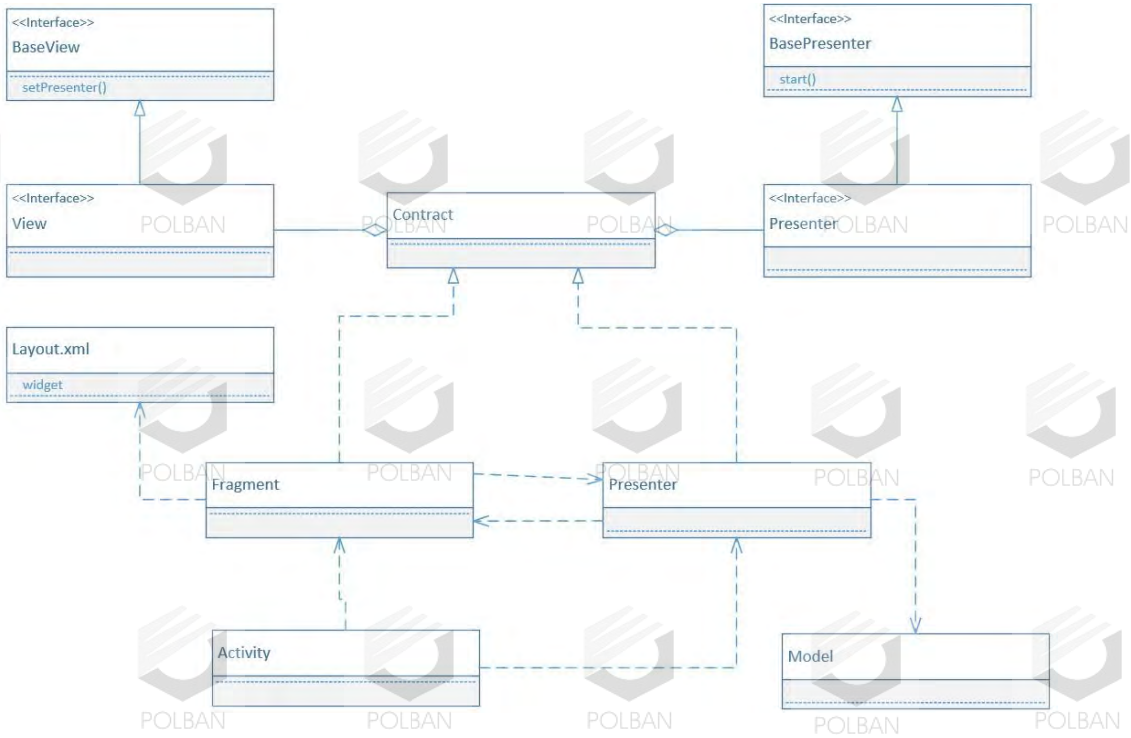


Gambar II.2 Ilustrasi Arsitektur MVP pada android [6]

Arsitektur MVP memiliki dua variasi berdasarkan tugas presenter: Supervising Controller dan Passive View [6]. Supervising Controller merupakan versi asli MVP. Dalam versi ini, view (tampilan) akan mengontrol bagian logic yang sederhana, sedangkan presenter mengontrol logika yang lebih rumit. Passive View lebih suka menganggap tampilan sebagai tiruan dan meneruskan semua logika ke presenter.

Android menggunakan *passive view*. Tetapi tidak ada peraturan khusus maupun peraturan formal untuk mengimplementasikan MVP. Untungnya, Google menerbitkan sampel open source MVP di GitHub [14]. Mereka menyebut sampel ini dengan sebutan *Android Architecture Blueprints* [beta]. Ini merupakan upaya untuk memformalkan peraturan implementasi MVP. Oleh karena itu, penelitian ini mengambil sampel tersebut sebagai standar.

Diagram UML *class* pada Gambar II.2 menunjukkan implementasi MVP. *Model* dan *view* yang ada didalamnya sama dengan yang ada pada MVC. *Presenter* merupakan objek java biasa yang mengatur dan menggabungkan antara *view* dan *model*.



Gambar II.3 Class Diagram Arsitektur MVP [6]

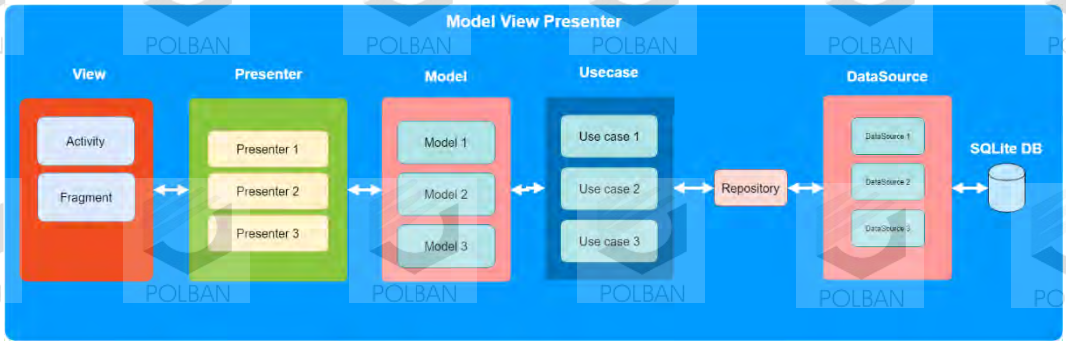
Base Interface. *BaseView* dan *BasePresenter* adalah induk dari semua *view* dan presenter. Kedua *base interface* ini memastikan bahwa komponen *Presenter* dan *View* terikat bersama dan data yang diperlukan dimuat. *BaseView Interface* adalah *base class* (kelas dasar) dari komponen *View* yang memiliki satu metode setPresenter() yang paling penting untuk mengatur tampilan presenter ini. *BasePresenter Interface* adalah dasar untuk komponen Presenter. Metode yang paling penting yaitu ada di dalam onStart() untuk menyiapkan data yang akan ditampilkan dalam *view*. Metode onStart() dalam *BasePresenter* ini biasanya disebut metode onResume() dalam Fragmen.

Contract Class adalah komponen untuk mengelola *interface* antara tampilan spesifik tertentu dan presenternya yang merupakan komposisi *View interface* dan

Presenter interface. *View* hanya dapat memanggil metode *Presenter* yang ditampilkan pada *Presenter interface* dan begitupula sebaliknya.

Fragmen/*Activity* dan tata letak xml merupakan *view* (tampilan). Seringkali, Fragmen bertanggungjawab terhadap *view*.

Untuk melihat arsitektur MVP dari arsitektur diagram dari mulai view sampai ke database, akan ditunjukan pada Gambar II.4.



Gambar II.4 Arsitektur Diagram MVP [9]

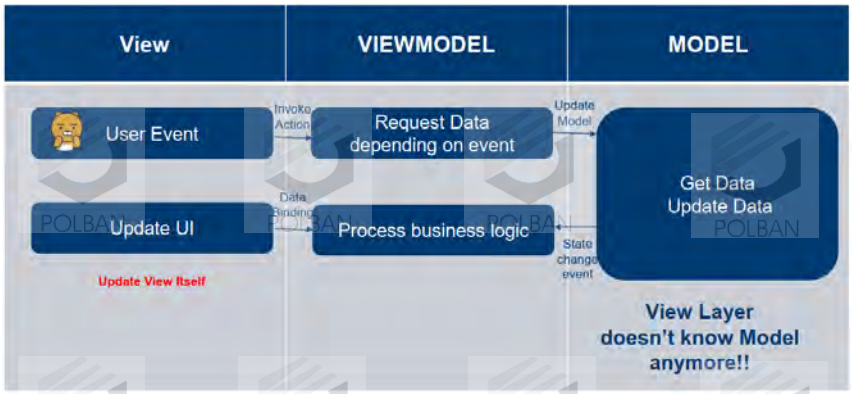
II.1.7 Arsitektur MVVM

MVVM merupakan singkatan dari Model View ViewModel sebuah konsep arsitektur pengembangan aplikasi yang dirilis oleh Google pada waktu yang bersamaan dengan diluncurkannya arsitektur MVP. Arsitektur ini bukan untuk memperbaiki arsitektur MVP, akan tetapi memiliki peran yang sama untuk memperbaiki arsitektur MVC. Akan tetapi memiliki prilaku yang berbeda antara arsitektur ini dengan MVP. Arsitektur ini pertama kali diperkenalkan oleh Martin Fowler dari Microsoft. Menurut Martin Fowler [6], MVVM biasa digunakan untuk membangun user interface dan digunakan oleh Microsoft untuk mengimplementasikan aplikasi Windows. Kehebatan dari arsitektur ini adalah penggunaan dari *binding enggine* atau yang sering *developer* sebut *data binding*. *Binding enggine* akan menghindari semua kode boilerplate yang harus kita tulis untuk menghubungkan view model kita dengan view agar tetap diperbarui. Arsitektur MVVM memiliki tiga komponen, sebagaimana disebutkan dalam

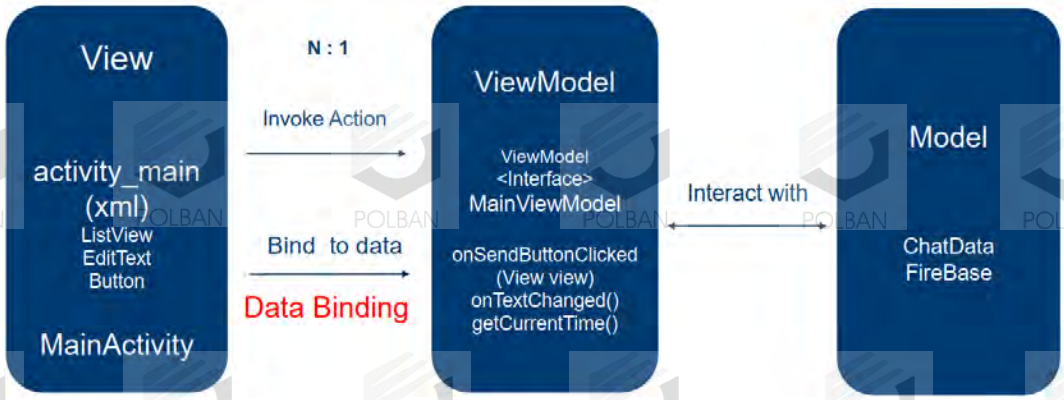
namanya MVVM, yaitu *Model*, *View*, dan *View-Model*. Komponen *View* menunjukkan aplikasi UI [6, 14].

1. *Model* pada arsitektur MVVM memiliki peran sama seperti pada arsitektur MVP yang merupakan *layer* yang menunjuk kepada objek dan data yang ada pada aplikasi.
2. *View* adalah sebuah *layer* sama seperti pada MVP yang mengandung keseluruhan detail dari implementasi *user interface* yang akan langsung berinteraksi dengan pengguna.
3. *ViewModel*, atau singkatan dari *model of view*, dimaksudkan untuk mengelola keadaan *view*. Komponen ini akan meneruskan data dan operasi untuk melihat dan juga mengelola logika serta perilaku tampilan.

Ilustrasi dari arsitektur MVVM ini ditunjukan pada Gambar II.4



Gambar II.5 Ilustrasi Arsitektur MVVM [6]



Gambar II.6 Ilustrasi Arsitektur MVVM pada android [6]

Pada arsitektur MVVM, *ViewModel* dan *View* memiliki koneksi yang lebih kompleks dibandingkan pada arsitektur MVP. Ada dua tipe koneksi yaitu *traditional connection* dan *databinding connection*. *traditional connection* memiliki kemiripan pada MVC dan MVP yaitu pada komponen *View-Model* yang akan mengubah *View* pada kode java [6].

Penyatuan data atau *databinding* adalah mekanisme baru yang diperkenalkan pada MVVM. Mekanisme ini memungkinkan *view* terikat secara langsung pada properties dan operasi *View-Model*. Melalui *databinding*, komponen *View-Model* tidak harus memberi tahu perubahan tampilan melalui *code*, *view* mengatahui bahwa data dimuat dan menunjukan data dengan tampilannya sendiri. Sebagai contoh, dalam arsitektur MVP dan MVC, setelah data dimuat, presenter dan *controller* akan mengatur tampilan dalam kode. Dengan mekanisme *databinding*, tampilan disatukan pada data ini dan ketika data dimuat, tampilan akan berubah secara otomatis.

Databinding antara *View* dan *View-Model* dapat dibagi menjadi kategori searah (*directional*) ataupun dua arah (*bidirectional*). Ketika data yang terikat dalam tampilan diubah, maka data dalam komponen *View-Model* juga mengetahui perubahannya. Ikatan ini, yang disebut sebagai *property binding*, termasuk ke dalam kategori *bi-directional*) Ada pula *databinding* yang disebut *operation binding* yang termasuk kategori *directional*. Sebuah operasi yang dibuat dalam

View-Model terikat dalam *widget* pada *view*. Ikatan ini seperti kode *JavaScript* dalam tampilan formulir HTML. Ketika pengguna menekan tombol *save* dalam formulir nya, maka perintah ini akan memanggil metode *pre-difined* dalam kode *JavaScript*. Melalui cara ini, pada komponen View-Model, kita tidak perlu menulis kode untuk menangkap *click event*.

Untuk menggunakan *databinding library* pada Android, kita perlu membuat beberapa perubahan berdasarkan pada implementasi Android saat ini. Pertama, import *library*. Kedua, atur *binding object* dalam *Activity class* saat menggembungkan tata letak file di metode OnCreate(). Ketiga, dalam file xml terkait, bagian data baru dengan variable-variabel terikat dimunculkan.

Sebagai contoh, aplikasi akan menunjukkan nama pengguna. Pada *Activity* tidak lagi menggunakan setContent(), tetapi akan menggunakan metode databinding. Dengan menambahkan sebauh objek *user* yang memiliki nama depan *Test* dan nama belakang *User*. Berikut adalah contoh kode dari penerapan data binding pada MVVM pada Gambar II.4.

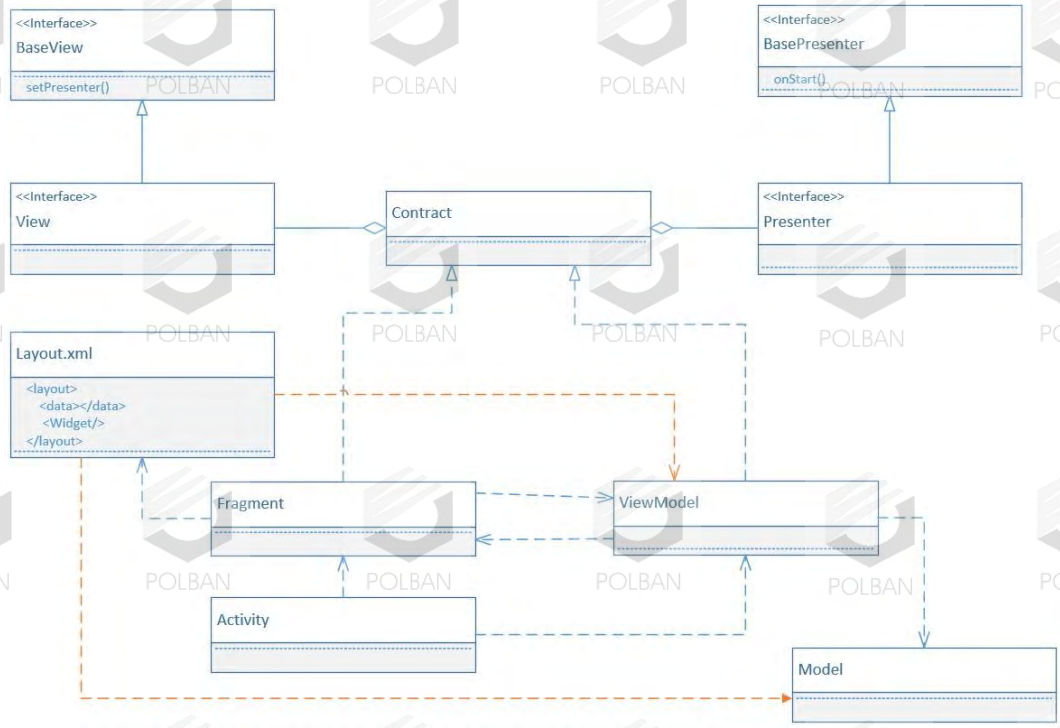
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MainActivityBinding binding = DataBindingUtil.setContentView(this, R.layout.main_activity);
    User user = new User("Test", "User");
    binding.setUser(user);
}
```

Gambar II.7 contoh penerapan data binding [9]

Selanjutnya dengan membuat file xml untuk tampilan dari activity. Ada sedikit perbedaannya dimana untuk data binding tidak lagi menggunakan Id, tetapi menggunakan @{} untuk merujuk atribut dari suatu objek yang akan dibuat dinamis. Contohnya ditunjukan pada Gambar II.5.

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="handlers" type="com.example.Handlers"/>
        <variable name="user" type="com.example.User"/>
    </data>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.firstName}"
            android:onClick="@{user.isFriend ? handlers.onClickFriend : handlers.onClickEnemy}"/>
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.lastName}"
            android:onClick="@{user.isFriend ? handlers.onClickFriend : handlers.onClickEnemy}"/>
    </LinearLayout>
</layout>
```

Gambar II.8 Contoh XML yang menerapkan databinding [9]

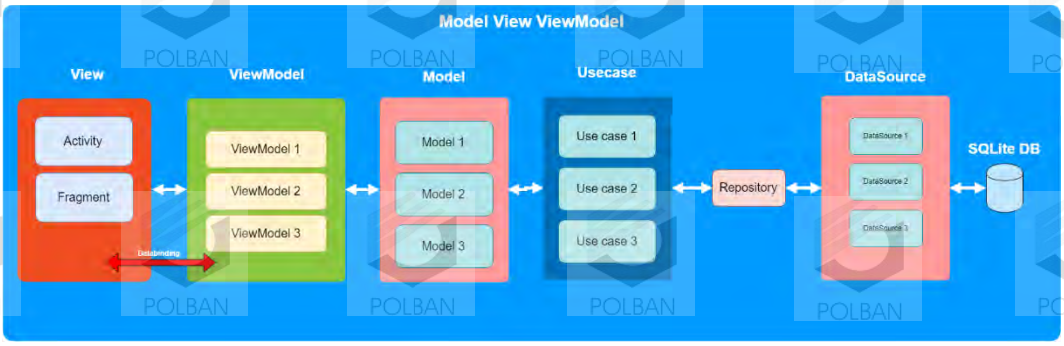


Gambar II.9 Class Diagram Arsitetkur MVVM [6]

Gambar 10 menunjukkan diagram Class Diagram dari arsitektur MVVM pada android. Dari *blueprint* yang sudah diterbitkan, arsitektur MVVM memiliki kemiripan dengan arsitektur MVP. Hal ini merupakan peningkatan dengan melibatkan *databinding library* dalam arsitektur MVP. Perbedaan utama yaitu

terletak pada adanya ketergantungan dari tampilan file xml terhadap ViewModel/dan Model Class (garis orange). Tamiplan file xml perlu untuk mengetahui struktur model. Pada contoh diatas, dibutuhkan pemahaman bahwa User object memiliki Nama depan (first name), nama akhir (LastName) dan atribut isFriend (isFriend attribute). Dengan cara yang sama untuk memanggil metode-metode (*methods*), file XML perlu mengetahui nama kelas MyHandler() dan metode-metodenya. Pada penelitian ini, konsep terkait Arsitektur MVVM menjadi yang paling penting, karena akan menjadi pembanding dan yang akan di implementasikan di android. Sehingga konsep ini perlu di pelajari.

Dan untuk arsitektur MVVM apabila dintunjukan kedalam arsitektur diagram untuk menggambarkan alur pada saat di dimplementasikan akan seperti pada Gambar II.10.



Gambar II.10 Arsitektur Diagram MVVM [9]

II.1.8 Point Of Sales

Point Of Sales atau yang biasa yang disingkat POS, merupakan kegiatan yang berorientasi pada penjualan serta sistem yang membantu proses transaksi. Penggunaan POS telah meningkatkan efisiensi kerja karena dalam mempercepat proses order pesanan oleh customer, dapat membuat laporan secara cepat, mempercepat proses perhitungan, memungkinkan pencarian data, serta melacak transaksi harian [16].

Aplikasi POS dibuat dengan berbagai kebutuhan seperti untuk penjualan pada toko kelontongan, rumah makan, supermarket, dll. Dan pada penelitian ini, Point Of Sale akan diterapkan pada aplikasi untuk menjadi objek penelitian. dan untuk

pengaplikasian POS ini akan diterapkan untuk toko-toko kelontongan dalam bertransaksi. Sehingga akan dibuat aplikasi untuk Point Of Sale untuk membantu toko kelntongan yang ada di indonesia. Salah astu contoh aplikasi POS dapat dlihat pada Gambar II.7.

Gambar II.11 Contoh aplikasi POS [16]

II.2 Karya Ilmiah Terkait

Lou, T. pada tahun 2016 [6], berusaha membandingkan Arsitektur antara MVC, MVP, dan MVVM. Tesis ini bertujuan membuktikan bahwa arsitektur MVP dan MVVM lebih baik dari MVC dilihat dari perspektif *quality*. Untuk membandingkannya digunakan 3 atribut pengukuran yaitu *testability, modifiability* dan *performance* dengan mengukur konsumsi *memory* dalam iterasi waktu. Pengukuran digunakan menggunakan metode ATAM dengan beberapa proses yang dilalui. Hasil perbandingan menunjukan bahwa arsitektur MVP dan MVVM lebih baik dibandingkan dengan MVC dilihat dari *modifiability, testability*, dan *performance* dengan mengukur konsumsi *memory*. Dari perbandingan konsumsi memori, MVVM = MVP < MVC sehingga MVC yang lebih banyak memakan memori. namun dari sisi *testability* MVVM lebih baik, dan pada *modofiability* MVP lebih baik dari MVVM. Penelitian ini juga menjelaskan bahwa aplikasi yang layak untuk pengujian peforma adalah aplikasi yang proses performanya banyak dilakukan di aplikasi itu sendiri dan yang yang menjadi salah satu isu yang diangkat di *performance* adalah *load* data, semakin banyak data yang digunakan dan strukturnya semakin kompleks (tidak hanya data dalam bentuk teks) semakin mempengaruhi *performance* aplikasi. Dan pada penelitian ini, aplikasi aplikasi yang dijadikan sebagai objek penelitian adalah aplikasi Todo-list yang dapat menampung banyak data gambar, text.

Cokun Aygun, pada tahun 2015 [15], membuat penelitian dengan judul "*The Performance Analysis of Applications Written Using MVP and MVC"* melakukan analisis perbandingan arsitektur antara arsitektur MVC dan MVP dengan melihat *runing time* dari aplikasi. Aplikasi yang menjadi objek eksperimennya adalah CRUD sederhana, dan yang menjadi variabel adalah jumlah pengguna yang menggunakan aplikasi. Dan hasil menunjukan bahwa MVP lebih baik dibandingkan MVC. Dan juga pada penelitian ini di bandingkan dari segi pengujian, dengan hasil bahwa MVP lebih mudah dilakukan pengujian dibandingkan dengan MVC.

Teerath Das, Penta dan Malavolta pada tahun 2016 [5], melakukan penelitian yang berjudul "*A Quantitative and Qualitative Investigation of Performance-Related Commits in Android Apps"*. Pada penelitian ini banyak membahas tentang masalah-masalah yang sering terjadi pada aplikasi android. Pada penelitian ini mencatat bahwa pada 2.443 commit yang dilakukan aplikasi android, 180 diantaranya mengandung 547 masalah performa yang didokumentasikan. Dengan berbagai masalah tentang performa seperti masalah GUI, *networking, managemen memory, load image*. Penelitian ini mengungkapkan bahwa pentingnya dilakukan analasis performa, karena performa menjadi salah satu masalah utama pada setiap pengembangan aplikasi.