SKRIPSI

ANALISIS PERFORMA ARSITEKTUR MVVM DAN MVP PADA APLIKASI ANDROID MENGGUNAKAN FITUR *NETWORKING* (STUDI KASUS: APLIKASI *MONITORING* MAHASISWA)

Disusun dan diajukan oleh:

YUSRIL
D121 17 1007

PROGRAM STUDI SARJANA
TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2023

LEMBAR PENGESAHAN SKRIPSI

ANALISIS PERFORMA ARSITEKTUR MVVM DAN MVP PADA APLIKASI ANDROID MENGGUNAKAN FITUR *NETWORKING* (STUDI KASUS: APLIKASI *MONITORING* MAHASISWA)

Disusun dan diajukan oleh

Yusril
D121 17 1007

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika
Fakultas Teknik Universitas Hasanuddin
Pada tanggal 10 Maret 2023
dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr. Eng. Muhammad Niswar, S.T., M.IT	A. Ais Prayogi Alimuddin, S.T., M.Eng
NIP 19730922 1999031 001	NIP 19830510 2014041 001

Ketua Program Studi,

Prof. Dr. Ir. Indrabayu, S.T., M.T.,
M.Bus.Sys., IPM, ASEAN.Eng.
NIP 19750716 2002121 004

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini ;

Nama	: Yusril
NIM	: D121 17 1007
Program Studi	: Teknik Informatika
Jenjang	: S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

Analisis Performa Arsitektur MVVM Dan MVP pada Aplikasi Android Menggunakan Fitur *Networking* (Studi Kasus: Aplikasi *Monitoring* Mahasiswa)

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitannya. Oleh karena itu semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak mana pun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala risiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 10 Maret 2023

Yang Menyatakan

Yusril

ABSTRAK

YUSRIL. *Analisis Performa Arsitektur MVVM dan MVP pada Aplikasi Android Menggunakan Fitur Networking (Studi Kasus: Aplikasi Monitoring Mahasiswa)* (dibimbing oleh Dr. Eng. Muhammad Niswar, S.T., M.IT dan A. Ais Prayogi Alimuddin, S.T., M.Eng)

Aplikasi Android merupakan aplikasi GUI paling populer yang berjalan di perangkat seluler, lebih menekankan pada pengoptimalan kinerja daripada perangkat lunak tradisional (misalnya, aplikasi GUI desktop) karena sumber dayanya yang terbatas. Oleh karena itu performa adalah aspek yang harus diperhatikan dalam pengembangan aplikasi android agar aplikasi yang dibangun dapat berjalan dengan maksimal pada perangkat android. Arsitektur yang digunakan dalam pengembangan aplikasi dapat berpengaruh terhadap performa aplikasi. Hal ini bisa menjadi tantangan bagi para pengembang aplikasi android untuk membuat aplikasi dengan performa yang maksimal. Penelitian ini dilakukan untuk mengetahui arsitektur terbaik pada aplikasi android dengan fitur *networking* berdasarkan performanya. Arsitektur yang diteliti adalah Model View ViewModel (MVVM) dan Model View Presenter (MVP) dengan aspek performa yang diukur adalah penggunaan CPU, penggunaan memori dan waktu eksekusi aplikasi pada perangkat android, serta jumlah dan ukuran baris kode (*line of code*) pada pengembangan android. Hasilnya menunjukkan bahwa aplikasi dengan arsitektur MVVM unggul pada empat aspek performa yang diujikan yaitu penggunaan memori, waktu eksekusi, jumlah dan ukuran baris kode. Rata-rata penggunaan memori pada arsitektur MVVM yaitu 137,39 MB yang lebih kecil dengan rata-rata penggunaan memori pada aplikasi dengan arsitektur MVP sebanyak 145,85 MB. Pada waktu eksekusi, aplikasi dengan arsitektur MVVM lebih unggul dengan rata-rata 621,16 ms yang lebih cepat dengan rata-rata waktu eksekusi pada aplikasi dengan arsitektur MVP selama 647,42 ms. Sedangkan arsitektur MVP unggul pada aspek penggunaan CPU dengan rata-rata 12,36% yang lebih sedikit dengan rata-rata penggunaan CPU pada aplikasi dengan arsitektur MVVM sebanyak 12,6%. Hal ini terjadi karena pada arsitektur MVVM terdapat tambahan library viewmodel yang membuat penggunaan memori dan waktu eksekusi lebih efisien tapi menambah beban pada CPU. Serta dalam pengembangan arsitektur MVVM memiliki jumlah dan ukuran kode yang lebih sedikit dibanding dengan MVP, hal ini terjadi karena penggunaan *interface contract* yang dibutuhkan pada MVP.

Kata Kunci: arsitektur android, Model View ViewModel, Model View Presenter, penggunaan CPU, penggunaan memori, waktu eksekusi, line of code, size of code, networking

ABSTRACT

YUSRIL. *Performance Analysis of MVVM and MVP Architectures on Android Applications Using Networking Features (Case Study: Student Monitoring Application)* (supervised by Dr. Eng. Muhammad Niswar, S.T., M.IT and A. Ais Prayogi Alimuddin, S.T., M.Eng)

Android apps are the most popular GUI apps that run on mobile devices, putting more emphasis on performance optimization than traditional software (e.g., desktop GUI apps) due to their limited resources. Therefore, performance is an aspect that must be considered in the development of android applications so that the built applications can run optimally on android devices. The architecture used in application development can affect application performance. This can be a challenge for android application developers to create applications with maximum performance. This research was conducted to find out the best architecture in Android applications with networking features based on their performance. The architecture studied is Model View ViewModel (MVVM) and Model View Presenter (MVP) with performance aspects measured are CPU usage, memory usage and application execution time on android devices, and the number and size of lines of code in android development. The results show that applications with MVVM architecture excel at the four performance aspects tested, namely memory usage, execution time, number and size of lines of code. The average memory usage in the MVVM architecture is 137.39 MB which is smaller with the average memory usage in applications with an MVP architecture of 145.85 MB. At execution time, applications with MVVM architecture are superior with an average of 621.16 ms which is faster with an average execution time on applications with an MVP architecture of 647.42 ms. Meanwhile, the MVP architecture excels in terms of CPU usage with an average of 12.36% which is less than the average CPU usage in applications with an MVVM architecture of 12.6%. This happens because in the MVVM architecture there is an additional viewmodel library that makes memory usage and execution time more efficient but increases the load on the CPU. As well as in the development of the MVVM architecture has a smaller number and size of code compared to MVP, this happens because of the use of the contract interface needed in MVP.

Keywords: android architecture, Model View ViewModel, Model View Presenter, CPU usage, memory usage, execution time, line of code, size of code, networking

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SINGKATAN DAN ARTI SIMBOL

Lambang/Singkatan	Arti dan Keterangan
MVVM	*Model View ViewModel*
MVP	*Model View Presenter*
LOC	*Line of Code*
SOC	*Size of Code*

DAFTAR LAMPIRAN

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan kehadirat Allah *subhanahu wa ta'ala*, yang telah memberikan rahmat dan karunia-Nya, sehingga dapat menyelesaikan tugas akhir dengan judul "Analisis Performa Arsitektur MVVM dan MVP pada Aplikasi Android Menggunakan Fitur *Networking* (Studi Kasus: Aplikasi *Monitoring* Mahasiswa)" sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 di Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.

Penulis menyadari bahwa penyusunan dan penulisan laporan tugas akhir ini tidak lepas dari bantuan, bimbingan serta dukungan dari berbagai pihak, dari masa perkuliahan sampai dengan masa penyusunan tugas akhir. Oleh karena itu, penulis dengan senang hati menyampaikan terima kasih kepada:

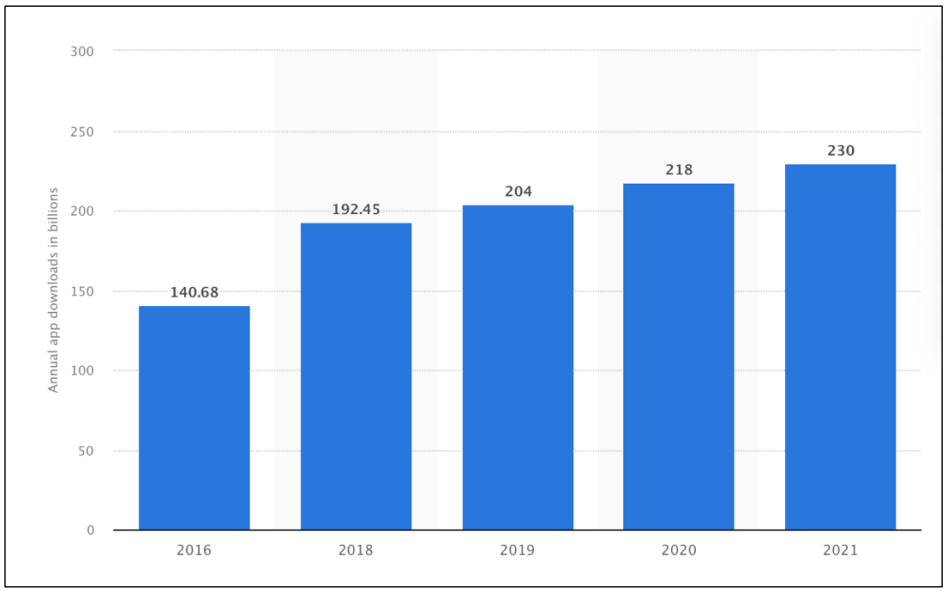
1. Allah *subhanahu wa ta'ala* atas semua karunia serta pertolongan-Nya yang tiada batas, yang telah diberikan kepada penulis di setiap langkah dalam pembuatan program hingga penulisan laporan ini.
2. Kedua orang tua penulis, Bapak Irwan dan Ibu Nurhaeni yang telah mendukung penuh dan selalu memberikan doa, dukungan, dan semangat yang tiada hentinya.
3. Bapak Dr. Eng. Muhammad Niswar, S.T., M.IT., selaku pembimbing I dan Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng., selaku pembimbing II, yang senantiasa menyediakan waktu, pikiran dan keyakinan yang luar biasa dalam mengarahkan penulis untuk menyelesaikan tugas akhir.
4. Bapak Dr. Indrabayu, S.T., M.T., M.Bus.sys., selaku ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.
5. Segenap staf dan dosen Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin yang telah membantu kelancaran penyelesaian tugas akhir.
6. Segenap keluarga Laboratorium UBICON Universitas Hasanuddin yang telah menjadi tempat yang nyaman untuk saling bertukar pikiran dan saling memberikan semangat di masa-masa sulit.
7. Seluruh anggota RECOGNIZER (Informatika 2017) yang telah memberi dukungan dan bantuan kepada penulis.
8. Seluruh teman masa kecil yang selalu mendukung dan menghibur penulis sampai sekarang.
9. Serta pihak-pihak lain yang tidak tersebutkan dan tanpa sadar telah menjadi inspirasi dan membantu penulis dalam menyelesaikan tugas akhir.

Akhir kata, penulis berharap semoga Tuhan Yang Maha Esa berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga tugas akhir ini dapat memberikan manfaat bagi pengembangan ilmu selanjutnya. Aamiin.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan industri perangkat lunak sudah semakin meningkat. Terlebih pada perkembangan penggunaan teknologi aplikasi seluler yang meningkat secara signifikan. Berbagai macam aktivitas telah dilakukan secara seluler karena mudah, cepat dan dapat dilakukan di mana saja dan kapan saja terlebih pada kondisi WFH (*work from home*) yang popular saat ini. Menurut Ceci (2022) pada laman Statista menunjukkan bahwa perkembangan pengguna aplikasi *mobile* meningkat dari tahun ke tahun dan diperkirakan akan terus meningkat.



Gambar 1 Grafik jumlah unduh aplikasi mobile di dunia dari tahun 2016 sampai 2021 (Statista, 2022)

Aplikasi Android merupakan aplikasi GUI paling populer yang berjalan di perangkat seluler, lebih menekankan pada pengoptimalan kinerja daripada perangkat lunak tradisional (misalnya, aplikasi GUI desktop) karena sumber dayanya yang terbatas (Feng et al., 2019).

Banyak kriteria performa yang harus dipenuhi dalam pengembangan aplikasi android. Seperti penggunaan *resource* yang seminim mungkin, konsumsi daya yang kecil dan lain-lain (Sibarani et al., 2018). Bahkan dalam pemilihan arsitektur dan bahasa yang berbeda dapat memengaruhi performa aplikasi *mobile*.

Sebelumnya sebuah penelitian yang dilakukan oleh Wisnuadhi, dkk (2020) dengan judul "*Performance Comparison of Native Android Application on MVP and MVVM*". Pengukuran yang dilakukan dalam penelitian itu terbatas pada pengolahan data secara internal di dalam aplikasi (luring). Objek aplikasi yang digunakan sebagai eksperimen hanya untuk kebutuhan pembuktian konsep, sehingga hasil ini perlu ditinjau lebih lanjut pada aplikasi dengan kompleksitas yang lebih tinggi, termasuk pengaruh beberapa faktor seperti akses jaringan eksternal (daring), penggunaan API atau *Library* dari sistem lain (pihak ketiga). Selain itu, penelitian selanjutnya juga dapat membandingkan penggunaan arsitektur MVVM (*Model View ViewModel*) atau MVP (*Model View Presenter*) pada bahasa *native* Android lainnya seperti Kotlin, apakah hasilnya akan juga berlaku sama atau ada pengaruh lain dengan penggunaan bahasa, ini juga akan menjadi sesuatu yang menarik untuk dibahas.

Dari penjelasan tentang penelitian sebelumnya pada arsitektur MVP dan MVVM bahwa masih terbatas pada penggunaan data lokal saja dan pada penelitian kali ini akan menggunakan fitur *networking*. Maka dari itu, penulis akan membuat aplikasi android untuk *monitoring* mahasiswa untuk mendukung dalam pengujian pada penelitian ini. Aplikasi ini dipilih karena sistem *monitoring* mahasiswa bimbingan di Universitas Hasanuddin hanya tersedia di *platform* website, sehingga dengan adanya versi aplikasi android bisa memudahkan dalam akses dan dengan tampilan yang lebih ramah dengan pengguna. Selain itu, aplikasi ini akan menampilkan data mahasiswa bimbingan dengan fitur-fitur tambahan seperti, status evaluasi mahasiswa yang tidak ada di versi website.

Maka dari itu, pada penelitian kemudian mengangkat sebuah penelitian untuk mengetahui bagaimana Pengaruh Performa Arsitektur MVVM dan MVP pada Aplikasi Android Menggunakan Fitur *Networking* (Studi Kasus: Aplikasi *Monitoring* Mahasiswa). Adapun dalam penelitian ini pengukuran performa dilakukan dengan empat variabel yakni penggunaan CPU, penggunaan memori, dan waktu eksekusi serta efisiensi baris kode pada pengembangan kedua arsitektur ini.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah dari penelitian ini ialah sebagai berikut:

a. Bagaimana cara mengembangkan arsitektur MVVM dan MVP pada aplikasi *monitoring* mahasiswa menggunakan fitur *networking*?
b. Bagaimana cara mengukur performa arsitektur MVVM dan MVP pada aplikasi *monitoring* mahasiswa menggunakan fitur *networking*?

1.3 Tujuan Penelitian

Adapun tujuan dari penelitian ini ialah sebagai berikut:

a. Untuk membuat aplikasi android dengan arsitektur MVVM dan MVP.
b. Untuk mendapatkan perbandingan performa arsitektur MVVM dan MVP pada aplikasi android dengan fitur *networking*

1.4 Manfaat Penelitian

Adapun manfaat dari penelitian ini ialah sebagai berikut:

a. Memberikan informasi mengenai performa arsitektur MVVM dan MVP pada aplikasi android dengan fitur *networking*
b. Untuk mendapatkan perbandingan performa arsitektur MVVM dan MVP pada aplikasi android dengan fitur *networking*

1.5 Ruang Lingkup

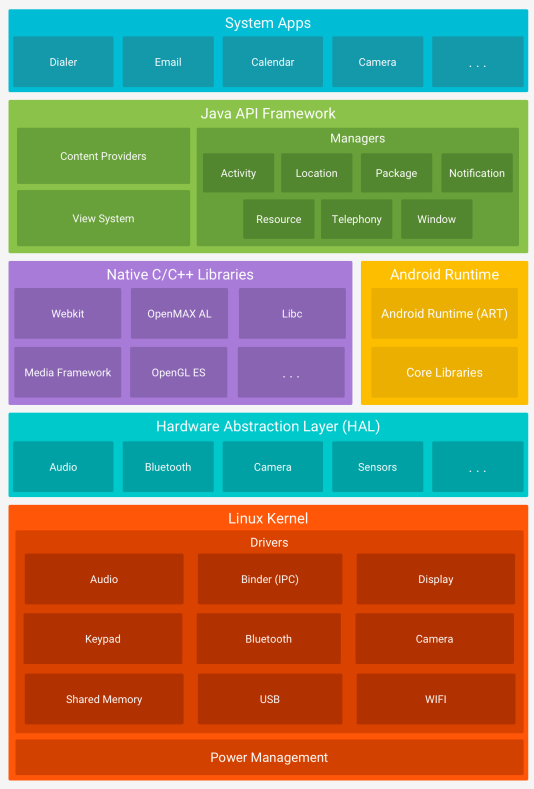
Ruang lingkup dalam penelitian ini ialah sebagai berikut:

a. IDE yang digunakan adalah Android Studio.
b. Menggunakan bahasa pemrograman Kotlin.
c. *Networking Library* yang digunakan adalah Retrofit
d. Variabel – variabel pengukuran performa terdiri dari:
    - Penggunaan CPU
    - Penggunaan memori
    - Waktu eksekusi
    - Jumlah baris dan ukuran file kode

BAB II
TINJAUAN PUSTAKA

2.1 Android

Android adalah sistem operasi berbasis Linux yang terutama dirancang untuk perangkat seluler layar sentuh seperti perangkat *smartphone*, dan tablet yang banyak tersedia di pasaran. Android merupakan sistem operasi yang menyediakan aplikasi yang nyaman bagi pengguna dan merupakan sistem operasi *open source* yang berarti pengembang bisa bebas menggunakan dan memodifikasinya (Degu, 2019)



Gambar 2 Struktur Android

Menurut laman Android Developer (2022), struktur Android dapat diuraikan sebagai berikut:

1. ***Linux Kernel***

   Fondasi *platform* Android adalah *kernel* Linux. Sebagai contoh, *Android Runtime* (ART) bergantung pada *kernel* Linux untuk fungsionalitas dasar seperti *threading* dan pengelolaan memori tingkat rendah.

Menggunakan *kernel* Linux memungkinkan Android untuk memanfaatkan fitur keamanan inti dan memungkinkan produsen perangkat untuk mengembangkan *driver* perangkat keras untuk *kernel* yang cukup dikenal.

2. ***Hardware Abstraction Layer* (HAL)**

   HAL memberikan antarmuka standar yang mengungkap kemampuan perangkat keras perangkat ke kerangka kerja API Java yang lebih tinggi. HAL terdiri atas beberapa *library*, masing-masing menerapkan *interface* untuk komponen perangkat keras tertentu, seperti modul kamera atau *Bluetooth*.

3. ***Android Runtime***

   Untuk perangkat yang menjalankan Android versi 5.0 (API level 21) atau lebih tinggi, setiap aplikasi menjalankan proses masing-masing dengan tahap Android *Runtime* (ART). ART ditulis guna menjalankan beberapa mesin virtual pada perangkat bermemori rendah dengan mengeksekusi *file* DEX, format *bytecode* yang dirancang khusus untuk Android yang dioptimalkan untuk *footprint* memori minimal.

4. ***Native C/C++ Libraries***

   Banyak komponen dan layanan sistem Android inti seperti ART dan HAL dibuat dari kode bawaan yang memerlukan *library* bawaan yang tertulis dalam C dan C++. Platform Android memungkinkan kerangka kerja API Java meningkatkan fungsi beberapa *library* bawaan pada aplikasi. Misalnya, Anda dapat mengakses OpenGL ES melalui kerangka kerja API OpenGL Java Android guna menambahkan dukungan untuk menggambar dan memanipulasi grafik 2D dan 3D pada aplikasi.

5. ***Java API Framework***

   Keseluruhan rangkaian fitur pada Android OS tersedia untuk Anda melalui API yang ditulis dalam bahasa Java. API ini membentuk elemen dasar yang harus dibuat pada aplikasi Android dengan menyederhanakan penggunaan *core*, komponen dan layanan sistem modular.

6. ***System Apps***

   Android dilengkapi dengan serangkaian aplikasi inti untuk email, SMS, kalender, menjelajahi internet, kontak, dll. Aplikasi yang disertai

dengan platform tidak memiliki status khusus pada aplikasi yang pengguna ingin *install*. Jadi, aplikasi pihak ketiga dapat menjadi *browser* web utama, pengolah pesan SMS atau bahkan *keyboard* utama (beberapa pengecualian berlaku, seperti aplikasi *Settings* sistem).

Sistem Android mengimplementasikan *principle of least privilege*. Ini berarti, secara *default* aplikasi hanya memiliki akses ke komponen yang diperlukan untuk melakukan pekerjaannya dan tidak lebih dari itu. Hal ini menghasilkan lingkungan yang sangat aman sehingga aplikasi tidak bisa mengakses bagian sistem bila tidak diberi izin. Komponen aplikasi adalah blok pembangun penting dari aplikasi Android. Setiap komponen adalah titik masuk tempat sistem atau pengguna dapat memasuki aplikasi Anda. Beberapa komponen bergantung pada komponen lainnya.

Berikut adalah uraian dari empat tipe komponen aplikasi:

1. ***Activities***

   Activity adalah *entry point* untuk berinteraksi dengan pengguna. Ini mewakili satu layar dengan antarmuka pengguna. Misalnya, aplikasi email mungkin memiliki satu aktivitas yang menampilkan daftar email baru, aktivitas lain untuk menulis email, dan aktivitas satunya lagi untuk membaca email. Walaupun semua aktivitas bekerja sama untuk membentuk pengalaman pengguna yang kohesif dalam aplikasi email, masing-masing tidak saling bergantung. Karenanya, aplikasi berbeda bisa memulai salah satu aktivitas ini (jika aplikasi email mengizinkannya). Misalnya, aplikasi kamera bisa memulai aktivitas dalam aplikasi email yang membuat email baru agar pengguna bisa berbagi gambar

2. ***Services***

   Service adalah *entry point* serbaguna untuk menjaga aplikasi tetap berjalan di latar belakang untuk semua jenis alasan. Ini adalah komponen yang berjalan di latar belakang untuk melakukan operasi yang berjalan lama atau untuk melakukan pekerjaan bagi proses jarak jauh. Layanan tidak menyediakan antarmuka pengguna. Misalnya, sebuah layanan bisa memutar musik di latar belakang sementara pengguna berada dalam aplikasi lain, atau layanan bisa menarik data lewat jaringan tanpa memblokir interaksi pengguna dengan *activity*.

3. Broadcast Receivers

Broadcast receiver adalah komponen yang memungkinkan sistem menyampaikan kejadian di luar alur pengguna *regular*, menjadikan aplikasi tersebut dapat merespons pengumuman siaran seluruh sistem. Oleh karena penerima siaran adalah entri yang didefinisikan dengan baik ke dalam aplikasi, sistem dapat mengirimkan siaran meskipun ke aplikasi yang saat ini tidak berjalan. Jadi, misalnya, suatu aplikasi dapat menjadwalkan alarm untuk mengirimkan notifikasi agar pengguna tahu tentang acara yang akan datang.

4. Content Providers

Content Provider mengelola set data aplikasi secara bersama-sama, yang dapat Anda simpan di sistem *file*, di database SQLite, di web, atau di lokasi penyimpanan lain yang dapat diakses aplikasi Anda. Melalui penyedia materi, aplikasi lain bisa melakukan kueri atau memodifikasi data jika *content provider* mengizinkannya

2.2 Android Studio

Menurut Android Developer (2022), Android Studio adalah IDE (*Integrated Development Environment)* resmi untuk pengembangan aplikasi Android, yang didasarkan pada IntelliJ IDEA . Selain sebagai *editor* kode dan fitur developer IntelliJ yang kuat, Android Studio menawarkan banyak fitur yang meningkatkan produktivitas dalam membuat aplikasi Android, seperti:

1. Sistem *build* berbasis *Gradle* yang fleksibel
2. Emulator yang cepat dan kaya akan fitur
3. Lingkungan terpadu tempat Anda bisa mengembangkan aplikasi untuk semua perangkat Android
4. Integrasi Github untuk membantu membuat fitur aplikasi dan mengimpor contoh kode
5. *Framework* dan alat pengujian yang lengkap
6. *Lint Tool* untuk merekam performa, kegunaan, kompatibilitas versi, dan masalah lainnya
7. Dukungan C++ dan NDK (*Native Development Kit*)

8. Dukungan bawaan untuk *Google Cloud Platform*, yang memudahkan integrasi *Google Cloud Messaging* dan *App Engine.*

2.3 Kotlin

Pada Google I/O 2017, Google mengumumkan Kotlin, bahasa pemrograman baru yang resmi digunakan untuk mengembangkan aplikasi Android. Kotlin adalah bahasa pemrograman yang dikembangkan oleh JetBrains, yang juga mengembangkan Android Studio IDE. Bahasa Kotlin adalah pengembangan dari bahasa Java yang sebelumnya populer. Bahasa Kotlin memiliki fitur bahasa yang lebih baru daripada bahasa Java (Sibarani dkk, 2018).

Sejak 2017, pengembang Android dapat mulai menulis aplikasi Android dari awal menggunakan Kotlin, mengembangkan aplikasi Android mereka yang sudah ada yang ditulis dalam Java dengan menambahkan kode Kotlin (mungkin berkat interoperabilitas antara dua bahasa), atau memigrasikan aplikasi Android mereka dari Java ke Kotlin.

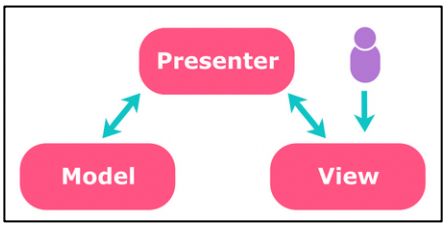
Dengan migrasi kode Java ke Kotlin, pengembang dapat menggunakan fitur bahasa pemrograman (misalnya *extension functions, lambdas, smart casts*) yang tidak tersedia dalam pengembangan Java untuk Android, dan untuk mendapatkan kode yang lebih aman (menghindari *Null Pointer Exceptions*) (Martinez dan Gois Mateus, 2021).

2.4 *Architectural Pattern*

Menurut Dicoding Indonesia (2022), *Architectural Pattern* memiliki skala yang lebih besar tentang bagaimana pengorganisasian keseluruhan komponen aplikasi. Pola ini lebih berfokus pada *separation of concern* (SOC), yaitu pembagian tugas yang lebih jelas antar komponennya.

Model View Presenter

Menurut Akhtar dan Ghafoor (2021), MVP merupakan alternatif untuk mengatasi permasalahan pada arsitektur MVC dengan memisahkan beberapa bagian satu sama lain. Pada arsitektur ini dibagi menjadi tiga layer utama, yaitu:



Gambar 3 Diagram arsitektur MVP

1. ***Model***

   Bertanggung jawab untuk menangani logika bisnis dan *data state*. Model mengambil dan memanipulasi data, berkomunikasi dengan *Presenter*, berinteraksi dengan *database* dan tidak memiliki hubungan dengan *View*.

2. ***View***

   Pada layer ini akan berisi *User Interface* (UI) dari aplikasi yang terdiri dari *Activity* dan *Fragment* serta layer ini akan berkomunikasi dengan *Presenter*.

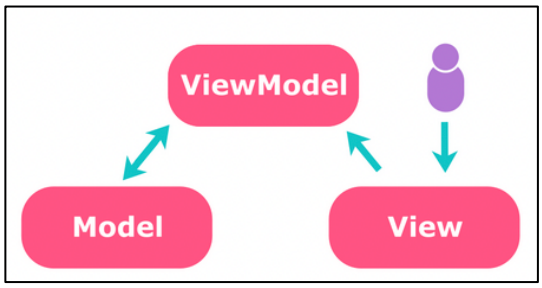
3. ***Presenter***

   Bertanggung jawab untuk menyajikan informasi dari *Model*. Mengontrol *View* dari aplikasi dan menyajikan *View*. Setiap interaksi antara *Model* dengan *View* dilakukan oleh *Presenter* dan menyimpan kembali data ke dalam *Model*.

MVP memiliki kelebihan yaitu komponen *view* dan *presenter* bisa digunakan kembali, kode lebih mudah dipahami dan dikelola, sedangkan untuk kekurangannya adalah ukuran kode bisa menjadi lebih besar dan antara *view* dan *presenter* memiliki hubungan yang terlalu erat (Yudhistira, 2020).

Model View ViewModel

Menurut Abdullah (2022), MVVM merupakan salah satu arsitektur dalam pembuatan aplikasi yang berfokus pada pemisahan antara logika bisnis dengan tampilan aplikasi. Dalam penerapannya, MVVM terdiri dari tiga layer, yaitu *Model, View*, dan *ViewModel*.



Gambar 4 Diagram arsitektur MVVM

1. ***Model***

   Layer ini adalah model atau entitas yang mereprentasikan data yang akan digunakan pada logika bisnis. Umumnya kelas-kelas yang ada di dalamnya berupa POJO (*Plain Old Java Object*) dan *Data Classes* jika menggunakan Kotlin. Pada layer ini juga yang akan digunakan untuk mengambil data baik dari API ataupun *local database*

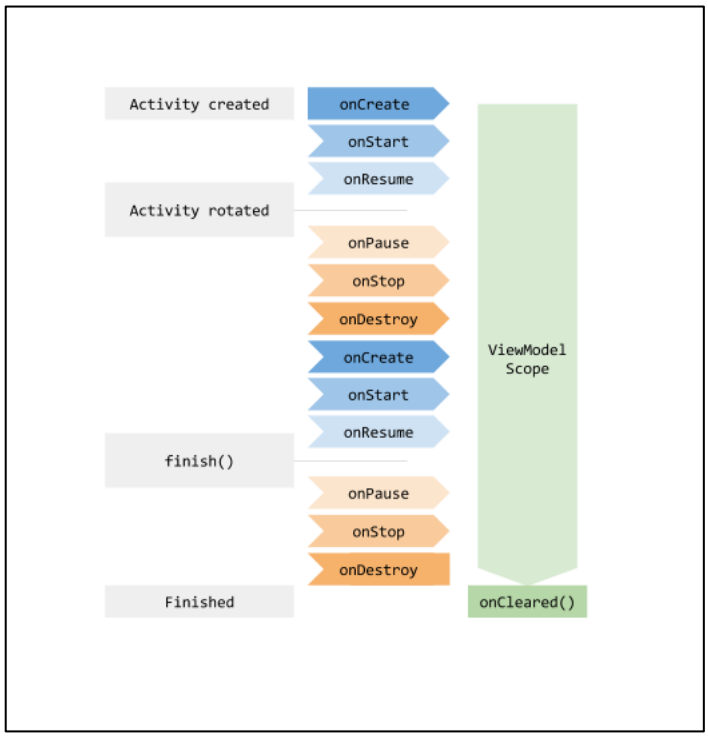
2. ***View***

   Tidak seperti sebelumnya, pada layer ini akan berisi *User Interface* (UI) dari aplikasi untuk mengatur bagaimana informasi akan ditampilkan. Pada umumnya, layer ini akan berisi kelas-kelas, seperti *Activity* dan *Fragment*.

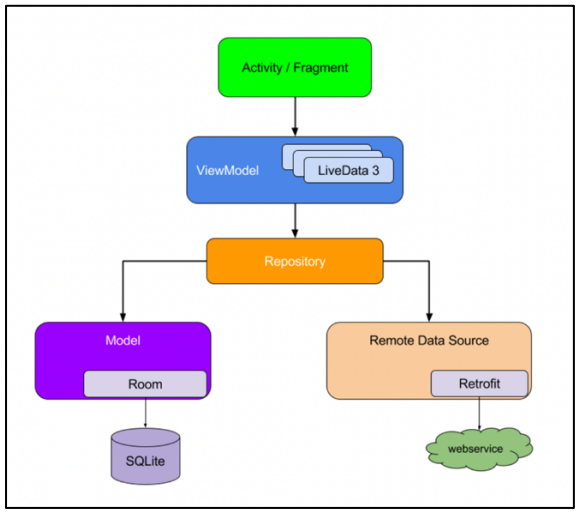
3. ***ViewModel***

   Layer terakhir adalah *ViewModel* yang bertugas sebagai penghubung antara dua layer sebelumnya. *ViewModel* akan berinteraksi dengan layer *model* dan data yang didapatkan akan diteruskan ke layer *view*.

Pada pengembangan arsitektur MVVM memiliki kelebihan yaitu tidak ada *interface* antara *View* dan *Model*, tidak ada hubungan erat antara *View* dan *ViewModel*, sedangkan untuk kekurangannya adalah kode ditulis bisa banyak (Yudhistira, 2020). Adapun *lifecycle* dari *viewmodel* itu sendiri akan mengikut dari activity parent yang *viewmodel* tempati.



Gambar 5 *Lifecycle viewmodel*



Gambar 6 Cara kerja MVVM

Untuk penjelasan cara kerja arsitektur MVVM sebagai berikut,

1. *Activity* atau *Fragment* akan bertanggung jawab sebagai *View*.
2. *View* akan melakukan observasi terhadap data yang disimpan pada *ViewModel*. Jika terjadi perubahan data pada *ViewModel*, maka *View* akan bertanggung jawab untuk melakukan perubahan pada tampilan sesuai dengan data yang terbaru.

3. *ViewModel* menyimpan data berupa *LiveData* supaya *View* dapat melakukan observasi.
4. *ViewModel* berkomunikasi dengan *Repository* (*Model*) untuk mendapatkan data dan melakukan perubahan pada data yang dimiliki.
5. *Repository* bertanggung jawab untuk mengatur sumber data yang dibutuhkan. Data bisa didapatkan baik dari *server* maupun dari *database* lokal (Putra, 2020).

2.5 Retrofit

Menurut (Lachgar et al., 2018), Retrofit merupakan *library* Android yang sederhana dan ringan yang dikembangkan oleh Square Inc. Dengan kata lain, Retrofit adalah REST *client* untuk Android yang dapat membuat *interface* yang mudah digunakan dan dapat mengubah aplikasi Android yang dapat berkomunikasi dengan *database* jarak jauh.

Kelebihan:

1. Sangat cepat
2. Memungkinkan untuk berkomunikasi langsung dengan *web service*
3. Sangat mudah untuk digunakan dan dipahami
4. Mendukung pembatalan *request*
5. Mendukung *post requests* dan *multipart upload*
6. Mendukung *synchronous* dan *asynchronous request*
7. Mendukung URL yang dinamis

Kekurangan:

1. Tidak mendukung pemuatan gambar, untuk hal ini akan membutuhkan *library* lain seperti Glide dan Picasso
2. Tidak mendukung pengaturan prioritas (Section.io, 2022)

2.6 *Agile Development*

Menurut Alsaqqa et al., (2020), *agile development* adalah proses yang mendukung filosofi *Agile*, yaitu nilai dan prinsip *Agile*. Setiap metode *Agile* terdiri dari kombinasi praktik yang berbeda, yang menjelaskan bagaimana pekerjaan sehari-hari dilakukan oleh pengembang perangkat lunak. Setiap metode berbeda

satu sama lain dengan memilih set terminologi dan praktik yang sesuai. Ada berbagai jenis metode *Agile* seperti Metode *Test-Driven Development* (TDD), Metode *Feature Driven Development* (FDD), Metode *Extreme Programming* (XP), Metode *Scrum*, Metode *Dynamic System Development Model* (DSDM) dan Metode *Crystal* dll. Setiap metode memiliki prinsip, siklus hidup, peran, keuntungan dan kerugian sendiri. Semua metode pengembangan perangkat lunak *Agile* ini membangun perangkat lunak dalam iterasi dan proses inkremental. Pada penelitian ini akan menggunakan metode FDD.

Feature Driven Development

Metode *Feature Driven Development* (FDD) adalah salah satu metode pengembangan *Agile*, yang mengelola iterasi inkremental pendek yang mengarah pada perangkat lunak fungsional. Fitur adalah fungsi yang dihargai untuk pengguna dalam perangkat lunak yang diperlukan.

FDD adalah metode pengembangan perangkat lunak yang sangat adaptif yang dapat menerima perubahan terakhir dalam persyaratan perangkat lunak. Fokus utama FDD adalah untuk menyediakan hasil berkualitas tinggi selama semua tahap proses pengembangan. Siklus hidup metode FDD terdiri dari lima proses berurutan seperti yang ditunjukkan pada gambar 2.5, proses-proses ini dilakukan dalam cara inkremental iteratif di mana akan menyediakan perangkat lunak akhir. Langkah-langkah ini adalah:

1. Mengembangkan model keseluruhan (*develop the overall model*): Dalam langkah ini, pengembang mendefinisikan konteks dan lingkup proyek secara keseluruhan yang dibutuhkan.
2. Membuat daftar fitur (*build the feature list*): Dalam langkah ini, model keseluruhan dan dokumentasi persyaratan digunakan untuk membuat daftar fitur keseluruhan untuk sistem yang dibutuhkan oleh pengguna dalam sistem.
3. Merencanakan berdasarkan fitur (*plan by feature*): Rencana ini diperoleh dari daftar fitur yang sebelumnya disetujui. Rencana ini akan berisi jadwal untuk milestone utama proyek dan jadwal rinci untuk setiap fitur.

4. Mendesain berdasarkan fitur (*design by feature*): Langkah ini adalah langkah iteratif; setiap iterasi dapat memakan waktu beberapa hari namun tidak lebih dari dua minggu dan menghasilkan paket desain untuk setiap kelas.
5. Membangun berdasarkan fitur (*build by feature*): Ini adalah langkah terakhir dalam proses FDD, dalam fase ini desain diimplementasikan (dikode), kemudian kode akan diinspeksi, melakukan proses pengujian. Langkah ini juga merupakan langkah iteratif yang sama dengan langkah desain berdasarkan fitur, setelah semua iterasi selesai maka fitur yang dikembangkan akan dipublikasikan dalam *build* utama, kemudian set fitur baru dimulai dan seterusnya.

Gambar 7 Proses FDD

2.7 *Performance Testing*

Menurut Wisnuadhi et al., (2020), *Performance testing* adalah teknik pengujian non-fungsional yang dilakukan untuk menentukan parameter sistem seperti kecepatan atau stabilitas sistem. Banyak faktor yang mempengaruhi performa sebuah perangkat lunak. Dalam pengujian perangkat lunak, *metrics* adalah ukuran kuantitatif dari tingkatan suatu sistem, komponen sistem, atau atribut proses. *Metrics* diperlukan untuk memahami kualitas dan efektivitas pengujian performa. Beberapa *metrics* yang sering digunakan dalam pengujian performa adalah sebagai berikut:

1. *Execution time*
2. *Wait time*
3. *Average load time*
4. *Request per second*
5. *CPU utilization*
6. *Memory utilization*

Performa mengukur seberapa cepat aplikasi dijalankan, seberapa cepat memuat data, dan konektivitas keseluruhan aplikasi pada operator yang berbeda. Pada penelitian ini, metrics aplikasi yang dibandingkan adalah penggunaan CPU, penggunaan memori dan waktu eksekusi.

2.8 Line dan Size of Code

Menurut Kaur (2016), LOC (*Line of Code*) merupakan salah satu langkah termudah untuk mengukur ukuran program perangkat lunak dalam proses pengembangan perangkat lunak. Sedangkan pada laman Geeksforgeeks ("*Lines of Code (LOC) in Software Engineering*," 2021), LOC adalah semua baris teks dalam kode yang bukan merupakan komentar, baris kosong. Karena LOC hanya menghitung volume kode, jadi hanya dapat digunakan untuk membandingkan atau memperkirakan proyek yang menggunakan bahasa yang sama dan menggunakan standar kode yang sama.

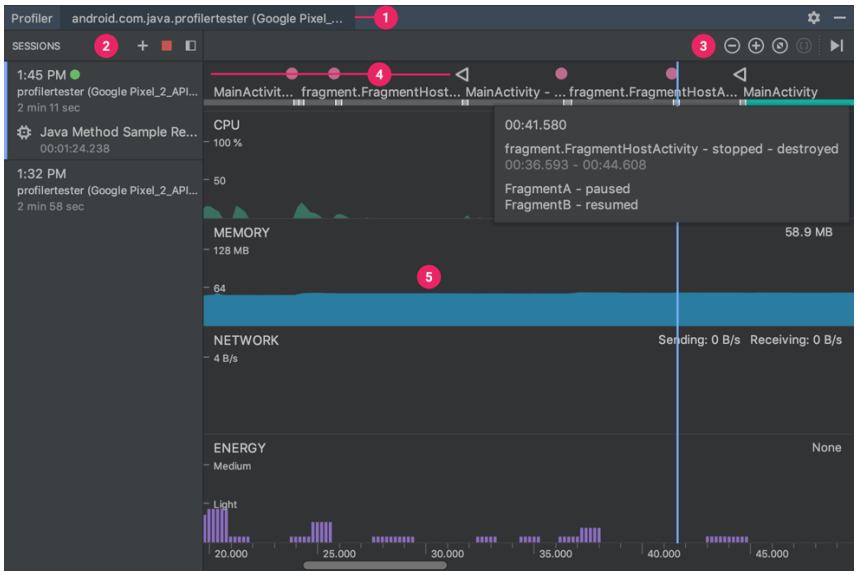
LOC merupakan metrik yang paling banyak digunakan dalam estimasi biaya karena sangat mudah untuk dilakukan. Selain daripada itu, berhubungan dengan LOC ini akan didapatkan juga hasil pengukuran berupa ukuran kode atau SOC (*Size of Code*) dari suatu *file* program. Salah satu *plugin* yang bisa digunakan untuk menghitung LOC dan ukuran kode adalah Statistic dari JetBrains.

2.9 Android Studio Profiler

Menurut laman Android Developer (2022), Android Profiler di Android Studio 3.0 dan yang lebih baru menggantikan *Android Monitor Tools*. Fitur Android Profiler menyediakan data *real-time* untuk membantu dalam memahami cara aplikasi menggunakan *resource* CPU, memori, jaringan, dan baterai.

Untuk membuka jendela Profiler, pilih **View** -> **Tool Windows** -> **Profiler** atau klik **Profile** di **toolbar**. Jika diminta oleh dialog ***Select Deployment Target***, pilih perangkat tempat yang ingin dibuat profil aplikasi. Jika telah menghubungkan perangkat melalui USB tetapi tidak melihatnya dalam daftar, pastikan telah mengaktifkan proses *debug* USB. Jika Anda menggunakan Android Emulator atau perangkat yang telah di-*root*, Android Profiler akan mencantumkan semua proses yang berjalan, meskipun proses tersebut mungkin tidak dapat di-*debug*. Ketika

meluncurkan aplikasi yang dapat di-*debug*, proses tersebut akan dipilih secara *default*.



Gambar 8 Tampilan Android Studio Profiler

1. Android Profiler menampilkan proses dan perangkat yang sedang dibuat profilnya.
2. Di panel *Sessions*, pilih sesi yang ingin dilihat, atau mulai sesi pembuatan profil baru.
3. Gunakan tombol *zoom* untuk mengontrol seberapa banyak lini masa yang ditampilkan, atau gunakan tombol *Attach to live* untuk menuju langsung ke *update real-time*.
4. Lini masa peristiwa akan menampilkan peristiwa yang berhubungan dengan input pengguna, termasuk aktivitas *keyboard*, perubahan kontrol volume, dan rotasi layar.
5. Tampilan lini masa bersama, yang meliputi grafik untuk penggunaan CPU, memori, jaringan, dan energi.