SKRIPSI

Analisis Performa *Reactive Programming Library* dalam Pengembangan Aplikasi Android

Disusun dan diajukan oleh:

ICHSANUL ALIFWAN
D121181020

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
MAKASSAR
2024

LEMBAR PENGESAHAN SKRIPSI

ANALISIS PERFORMA *REACTIVE PROGRAMMING LIBRARY* DALAM PENGEMBANGAN APLIKASI ANDROID

Disusun dan diajukan oleh

ICHSANUL ALIFWAN
D121181020

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian
Studi Program Sarjana Program Studi Teknik Informatika
Fakultas Teknik Universitas Hasanuddin
Pada tanggal 08 Juli 2024
dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
A. Ais Prayogi Alimuddin, S.T., M.Eng.	Iqra Aswad, S.T., M.T.
NIP 198305102014041001	NIP 199011282019043001

Ketua Program Studi,

Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM,ASEAN. Eng.
NIP 197507162002121004

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini;
Nama : Ichsanul Alifwan
NIM : D121181020
Program Studi : Teknik Informatika
Jenjang : S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

"Analisis Performa *Reactive Programming Library* dalam Pengembangan Aplikasi Android"

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 manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala resiko.

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, 30 Juli 2024

Yang Menyatakan

Ichsanul Alifwan

ABSTRAK

ICHSANUL ALIFWAN. *Analisis Performa Reactive Programming Library Dalam Pengembangan Aplikasi Android* (dibimbing oleh A. Ais Prayogi Alimuddin, S.T., M.Eng. dan Iqra Aswad, S.T., M.T.)

Reactive Programming telah menjadi pendekatan yang populer dalam pengembangan aplikasi Android, yang memungkinkan penanganan data asinkron dengan lebih efisien. Dalam penelitian ini, dianalisis performa beberapa *Reactive Programming library* yang umum digunakan dalam pengembangan aplikasi Android, seperti RxJava, Kotlin Flow, dan Reactor. Penelitian ini menggunakan metode *benchmark* untuk membandingkan kinerja ketiga *library* tersebut dalam skenario penggunaan yang umum dalam pengembangan aplikasi Android. Metrik performa yang diukur meliputi *execution time* dan *allocation count*. Selain itu, penelitian ini juga melakukan analisis statis menggunakan SonarQube, sebuah platform yang memungkinkan evaluasi *source code* untuk mengidentifikasi masalah potensial dan kesalahan dalam pengembangan aplikasi Android. Dalam analisis statis, diukur tiga metrik penting, yaitu *Lines of Code* (LOC), *Cognitive Complexity*, dan *Cyclomatic Complexity*. Hasil dari penelitian ini menunjukkan Reactor memiliki nilai *Cognitive Complexity* dan *Lines of Code* yang lebih rendah daripada Kotlin Flow dan RxJava, menunjukkan kode yang lebih sederhana. Namun, Kotlin Flow memiliki *Cyclomatic Complexity* yang lebih rendah. Kemudian, dari performa Kotlin Flow unggul dalam pengujian *network* dan integrasi, sementara Reactor lebih baik dalam pengujian database. Meskipun perbedaan pada *allocation count*, *memory usage*, dan *maximum cpu usage* hanya sedikit, namun perbedaan dalam *execution time* cukup signifikan. Hal ini penting untuk pencegahan ANR dan perilaku tak terduga pada perangkat Android atau API.

Kata Kunci: Android, *Reactive Programming*, *Benchmark*, SonarQube

ABSTRACT

ICHSANUL ALIFWAN. *Performance Analysis of Reactive Programming Library In Android Application Development* (*supervised* by A. Ais Prayogi Alimuddin, S.T., M.Eng. and Iqra Aswad, S.T., M.T.)

Reactive Programming has become a popular approach in Android app development, which enables more efficient handling of asynchronous data. In this study, the performance of several Reactive Programming libraries commonly used in Android app development, such as RxJava, Kotlin Flow, and Reactor, is analyzed. This research use benchmark methods to compare the performance of these three libraries in common usage scenarios in Android application development. The performance metrics measured include execution time and allocation count. In addition, this research also performed static analysis using SonarQube, a platform that enables source code evaluation to identify potential problems and errors in Android app development. In the static analysis, three important metrics were measured, namely Lines of Code (LOC), Cognitive Complexity, and Cyclomatic Complexity. The results of this study show Reactor has lower Cognitive Complexity and Lines of Code values than Kotlin Flow and RxJava, indicating simpler code. However, Kotlin Flow has lower Cyclomatic Complexity. In terms of performance, Kotlin Flow excels in network and integration testing, while Reactor does better in database testing. Although the difference in allocation count, memory usage, and maximum cpu usage is small, the difference in execution time is significant. This is important for ANR prevention and unexpected behavior on Android devices or APIs.

Keywords: Android, *Reactive Programming*, *Benchmark*, SonarQube

DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI i
PERNYATAAN KEASLIAN ii
ABSTRAK iii
ABSTRACT iv
DAFTAR ISI v
DAFTAR GAMBAR vii
DAFTAR TABEL ix
DAFTAR LAMPIRAN x
KATA PENGANTAR xi
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Tujuan Penelitian 3
1.4 Manfaat Penelitian 3
1.5 Batasan Masalah 4
BAB II TINJAUAN PUSTAKA 5
2.1 Android 5
2.2 Kotlin 6
2.3 Android Studio 8
2.4 Reactive Programming 9
2.5 Kotlin Flow 10
2.6 RxJava 11
2.7 Reactor 12
2.8 Benchmark 13
2.9 APK 14
2.10 Android Profiling 15
2.11 SonarQube 16
2.12 Analisis Statis 17
2.13 Cognitive Complexity 18
2.14 Cyclomatic Complexity 19
2.15 Lines Of Code (LOC) 19
BAB III METODE PENELITIAN 21
3.1 Tahapan Penelitian 21
3.2 Analisis Kebutuhan 23
3.3 Perancangan Sistem 24
3.4 Implementasi Sistem 29
3.5 Perancangan Antarmuka Sistem 37
3.6 Skenario Penggunaan Sistem 43

3.7 Mock Network Layer 45
3.8 Benchmarking 49
3.9 Pengujian Sistem 51
3.10 Skenario Pengujian 53
BAB IV. HASIL DAN PEMBAHASAN 57
4.1 Implementasi Antarmuka Aplikasi Barani 57
4.2 Hasil dan Pembahasan Pengujian Sistem 62
BAB V. KESIMPULAN DAN SARAN 78
5.1 Kesimpulan 78
5.2 Saran 79
DAFTAR PUSTAKA 80

DAFTAR GAMBAR

Gambar 1. Tampilan Tab *measures* SonarQube 18
Gambar 2. Tahapan Penelitian 21
Gambar 3. MVVM *pattern* 25
Gambar 4. MVVM dan *Reactive Programming* 25
Gambar 5. Arsitektur Aplikasi Secara Keseluruhan 27
Gambar 6. Class diagram komponen utama 28
Gambar 7. Contoh *data streams* pada *reactive programming* 29
Gambar 8. Representasi *reactive programming* dalam Aplikasi Android 30
Gambar 9. Implementasi menggunakan RxJava pada ViewModel 32
Gambar 10. Implementasi menggunakan Kotlin Flow pada ViewModel 33
Gambar 11. Implementasi menggunakan Reactor pada ViewModel 34
Gambar 12. Hubungan Antar Modul 36
Gambar 13. Rancangan Halaman Registrasi 37
Gambar 14. Rancangan Halaman *Login* 38
Gambar 15. Rancangan Halaman *Home* 39
Gambar 16. Rancangan Halaman Profil 40
Gambar 17. Rancangan Halaman Laporan 41
Gambar 18. Rancangan Halaman List Berita 42
Gambar 19. Rancangan Halaman Detail Berita 43
Gambar 20. *Activity diagram* akses fitur berita dan proses registrasi 44
Gambar 21. *Activity diagram* akses daftar berita tanpa autentikasi 45
Gambar 22. Penggunaan Retrofit *service builder* 46
Gambar 23. Implementasi *Mockinterceptor* 47
Gambar 24. *Mocked Network Flow* 48
Gambar 25. Contoh pengujian *benchmark* 49
Gambar 26. Contoh *benchmark* JSON *report* 51
Gambar 27. Pengujian Sistem 52
Gambar 28. Penggunaan SonarQube 53
Gambar 29. Halaman Registrasi 57
Gambar 30. Halaman *Login* 58
Gambar 31. Halaman *Home* 59
Gambar 32. Halaman Profil 59
Gambar 33. Halaman Laporan 60
Gambar 34. Halaman Daftar Berita 61
Gambar 35. Halaman Detail Berita 61
Gambar 36. Grafik hasil pengujian penggunaan *Lines of Code* 63
Gambar 37. Grafik hasil pengujian penggunaan *Cognitive Complexity* 64
Gambar 38. Grafik hasil pengujian penggunaan *Cyclomatic Complexity* 66
Gambar 39. Grafik hasil pengujian *network benchmark* untuk metrik *execution time* 68
Gambar 40. Grafik hasil pengujian *network benchmark* untuk metrik *allocation count* 69
Gambar 41. Grafik Hasil pengujian database *benchmark* untuk metrik *execution time* 71

Gambar 42. Grafik Hasil pengujian database *benchmark* untuk metrik *allocation count*.....72
Gambar 43. Grafik hasil pengujian Integrasi *benchmark* untuk metrik *execution time*.....73
Gambar 44. Grafik hasil pengujian *integration benchmark* untuk metrik *allocation count*.....75
Gambar 45. Grafik hasil pengujian *profiling* untuk metrik Penggunaan Memori .76
Gambar 46. Grafik hasil pengujian *profiling* untuk metrik Penggunaan CPU Maksimum 76
Gambar 47. Grafik hasil pengujian *profiling* untuk metrik Waktu Eksekusi 77

DAFTAR TABEL

Tabel 1. Perbandingan antara RxJava, Kotlin Flow, dan Reactor. 34
Tabel 2. *Test Case* Pengujian integrasi 56
Tabel 3. Hasil pengujian penggunaan *Lines of Code* 62
Tabel 4. Hasil pengujian penggunaan *Cognitive Complexity* 64
Tabel 5. Hasil pengujian penggunaan *Cyclomatic Complexity* 65
Tabel 6. Hasil pengujian *network benchmark* untuk metrik *execution time* 67
Tabel 7. Hasil pengujian *database benchmark* untuk metrik *execution time* 70
Tabel 8. Hasil pengujian integrasi *benchmark* untuk metrik *execution time* 73

DAFTAR LAMPIRAN

Lampiran 1 *Source Code*.....84

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Allah SWT atas rahmat dan berkat-Nya, sehingga dapat menyelesaikan tugas akhir skripsi yang berjudul "**Analisis Performa *Reactive Programming Library* dalam Pengembangan Aplikasi Android"** sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 di Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.

Penulis menyadari banyak kesulitan dan kendala yang dihadapi saat penyusunan tugas akhir ini. Dalam prosesnya, penulis memperoleh banyak bantuan, dukungan, dan bimbingan dari berbagai pihak. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan terima kasih kepada:

1. Allah SWT melalui berkat dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir ini.
2. Kedua orang tua penulis, Ayah Nasrul Abadi dan Ibu Suriyani yang selalu menyertai penulis dalam doanya serta mendukung, membantu, memberi semangat serta kasih sayang dalam perjalanan penulis menyelesaikan tugas akhir ini.
3. Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng. selaku pembimbing I dan Bapak Iqra Aswad, S.T., M.T. selaku pembimbing II, yang senantiasa menyediakan waktu, tenaga, pikiran, dan perhatian yang luar biasa dalam mengarahkan penulis untuk menyelesaikan tugas akhir.
4. Segenap Dosen dan Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah banyak membantu penulis selama masa perkuliahan.

5. Teman-teman Teknik Informatika Angkatan 2018 selaku rekan yang telah memberi bantuan, dukungan dan semangat selama masa perkuliahan dan penyusunan tugas akhir ini.
6. Serta berbagai pihak atas segala dukungan dan bantuannya yang tidak dapat penulis tuliskan satu persatu.

Penulis berharap semoga Allah SWT berkenan membalas segala kebaikan yang telah diterima oleh penulis dari berbagai pihak yang telah membantu mempermudah penulis dalam mengerjakan tugas akhir ini. Penulis menyadari bahwa tugas akhir ini masih jauh dari kata sempurna, oleh karena itu penulis mengharapkan segala bentuk saran serta masukan yang membangun dari berbagai pihak. Semoga tugas akhir ini dapat memberikan pengetahuan dan manfaat bagi penulis dan pembaca.

Makassar, Juni 2024

Penulis

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pengembangan aplikasi Android saat ini terus berkembang pesat, di mana platform *smartphone* Android saat ini sangat populer dan sejauh ini memiliki pangsa pasar terbesar di antara semua jenis ponsel cerdas di seluruh dunia (Wei dkk., 2018). Oleh karena itu, pengembang aplikasi Android dituntut untuk selalu *up to date* dengan perkembangan teknologi Android saat ini. Maka dari itu diperkenalkan sebuah pedoman *best practices* dalam pengembangan Android yaitu *Modern Android Development* (MAD), MAD adalah alat pengembangan, API, bahasa, dan teknologi distribusi yang direkomendasikan oleh tim Android untuk membantu *developer* menjadi lebih produktif serta membuat aplikasi yang lebih baik dan berfungsi di miliaran perangkat (Android Developers, 2022).

Dalam pengembangan aplikasi Android modern, *developer* menghadapi tantangan besar dalam menangani *task* yang membutuhkan sumber daya tinggi, seperti mengakses layanan jarak jauh atau database lokal. *Task* ini sering dijalankan secara asinkron untuk menjaga responsivitas *UI*. Pemrograman asinkron memungkinkan proses pada *UI thread* tetap berjalan sambil memulai proses tambahan pada *thread* berbeda, sehingga *UI* tidak terhambat oleh proses latar belakang. Namun, penggunaan banyak *thread* aktif dalam proses *asynchronous* dapat menimbulkan masalah pada perangkat Android karena keterbatasan memori dan manajemen *thread* yang kompleks (Chauhan dkk., 2021).

Pendekatan *reactive programming* menawarkan solusi untuk masalah ini. *Reactive programming* memungkinkan pengelolaan aktivitas *multi-threading* yang lebih efisien dengan menggunakan *reactive streams*. *Reactive streams* menyediakan aplikasi yang responsif dan mampu menangani banyak permintaan per detik serta mengelola *backpressure* keadaan di mana *Publisher* (pengirim) memberikan data yang berlebih sehingga *Subscriber* (penerima) tidak dapat menampungnya (Davis, 2019). Dengan paradigma ini, setiap proses dapat berjalan pada *thread* yang berbeda, sehingga tidak memberatkan satu *thread* saja atau

memblok *UI thread*, yang dapat menyebabkan aplikasi *freeze* dan terasa tidak responsif.

Reactive programming memberikan beberapa manfaat penting. Pertama, menjaga responsivitas aplikasi agar tetap interaktif meskipun menangani tugas berat. Kedua, meningkatkan skalabilitas aplikasi untuk menangani banyak permintaan simultan. Ketiga, penggunaan sumber daya lebih efisien dengan pengelolaan *thread* yang lebih baik. Keempat, memudahkan pemeliharaan aplikasi melalui *data streams* yang lebih jelas dan *error handling* yang lebih baik. Dengan kata lain, kita membutuhkan *reactive programming* untuk meningkatkan *user experience*. Aplikasi akan menjadi lebih fleksibel terhadap perubahan data. Bereaksi terhadap perubahan data dengan *code* yang lebih sedikit, tanpa perlu mengkhawatirkan manajemen *low-level threading*, sehingga developer dapat fokus pada logika bisnis.

Reactive programming banyak diterapkan dalam berbagai aplikasi dunia nyata. Contohnya, aplikasi *real-time* seperti sosial media memerlukan pembaruan konten secara *real-time*. Selain itu, aplikasi keuangan, *trading* saham atau *cryptocurrency* yang memantau dan memproses ribuan transaksi secara *real-time*. Namun untuk memilih *library* dengan implementasi paradigma tersebut tentunya mempertimbangkan beberapa hal. Biasanya *developer* memilih paradigma tertentu atau *library* berdasarkan faktor-faktor berikut: kemudahan implementasi, popularitas dan dukungan komunitas, dan pengalaman pemrograman pengembang sebelumnya (Komolov dkk., 2020).

Terdapat banyak *library* yang mengimplementasikan *reactive streams* atau *reactive programming*, salah satu *library open source* yang populer ialah RxJava. RxJava menyediakan pengurutan tugas yang lebih baik dari alternatif *closed-source* lainnya, abstraksi *threading*, dan memiliki sifat *reactive* yang dengan mudah menjadikannya alat terbaik untuk membangun *reactive systems* (Almeida, 2020). *Library* lain yang mengimplementasikan *reactive streams* ialah Reactor. Reactor adalah library reaktif generasi keempat, berdasarkan spesifikasi *reactive streams*, untuk membangun aplikasi *non-blocking* pada JVM (Project Reactor, 2022). Terdapat juga Kotlin Flow yang dibuat di atas coroutines. Coroutines adalah *concurrency design pattern* yang dapat digunakan pada Android untuk

menyederhanakan kode yang dijalankan secara asinkron (Android Developers, 2022b). Selain itu, coroutines lebih ringan untuk dipakai dan berguna untuk menulis kode secara asinkron, generator, dan proses *concurrent*. Tidak seperti JVM *Threads* yang membutuhkan memori hingga satu *megabyte*, coroutines hanya membutuhkan beberapa ratus *byte* (Chauhan dkk., 2021). Walaupun demikian ketiga *library* tersebut memiliki implementasi dan operator yang berbeda dalam menjalankan sebuah *task*.

Berdasarkan hal tersebut, maka pada penelitian ini penulis mengajukan penelitian untuk melakukan Analisis Performa *Reactive Programming Library* dalam Pengembangan Aplikasi Android. Adapun dalam penelitian ini metrik yang akan diukur adalah *lines of code*, *code complexity*, *memory usage*, *CPU usage*, *allocation count,* dan *execution time*.

1.2 Rumusan Masalah

Sesuai dengan permasalahan sebelumnya, rumusan masalah yang dapat diambil sebagai berikut:

a. Bagaimana mengimplementasikan *reactive programming library* pada pengembangan aplikasi berbasis Android?
b. Bagaimana menguji dan menganalisis performa *reactive programming library* pada aplikasi Android?

1.3 Tujuan Penelitian

Penelitian ini memiliki tujuan sebagai berikut:

a. Mengimplementasikan *reactive programming library* pada pengembangan aplikasi berbasis Android.
b. Untuk menguji dan menganalisis performa *reactive programming library* pada aplikasi Android.

1.4 Manfaat Penelitian

Manfaat penelitian ini ialah:

a. Bagi pembaca umum dapat dijadikan informasi mengenai implementasi dan analisis *reactive programming library* pada aplikasi Android serta sebagai acuan untuk penelitian lebih lanjut.
b. Bagi *Android Developer* dapat memberikan informasi hasil analisis *reactive programming library* sebagai pertimbangan dalam pemilihan *library*.
c. Bagi peneliti dapat menambah pengetahuan mengenai implementasi *reactive programming library* serta pengukuran performa pada pengembangan aplikasi Android.

1.5 Batasan Masalah

Adapun batasan masalah pada penelitian ini adalah sebagai berikut:

a. *Reactive programming library* yang akan dibandingkan adalah Kotlin Flow, RxJava, dan Reactor.
b. Bahasa pemrograman yang digunakan adalah Kotlin.
c. IDE yang digunakan ialah Android Studio.
d. Pengembangan aplikasi Android menggunakan arsitektur MVVM dan *repository pattern*.
e. Performa dan analisis statis diukur menggunakan Jetpack *benchmark library* dan SonarQube.
f. Metrik yang akan diukur adalah *execution time*, *allocation count*, *memory usage*, *maximum CPU usage*, *lines of code*, *cognitive complexity*, dan *cyclomatic complexity*.

BAB II
TINJAUAN PUSTAKA

2.1 Android

Android adalah sistem operasi sumber terbuka yang berjalan yang berjalan di atas kernel linux. Aplikasi Android dikembangkan menggunakan bahasa Java. Google memiliki SDK sendiri yang memungkinkan kode Java ini untuk mengontrol perangkat seperti ponsel tablet, dll. (Sarkar dkk., 2019).

Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi yang digunakan oleh bermacam piranti bergerak. Android saat ini telah menjadi sistem operasi mobile terpopuler di dunia. Perkembangan Android tidak lepas dari peran sang raksasa Google. Android pada mulanya didirikan oleh Andy Rubin, Rich Minner, Nick Sears, dan Chris White pada tahun 2003 (Kadek dkk., 2018).

Sistem operasi Android mendapatkan popularitas di kalangan pengembang karena sifatnya yang dapat disesuaikan. Ini sangat efisien untuk membangun aplikasi dalam satu platform dan menyebarkannya di beberapa platform secara bersamaan tanpa harus khawatir tentang perubahan yang harus dilakukan. Lintas platform mengusulkan beberapa pendekatan untuk mencapai tujuan ini dengan cara yang efisien dan cara. Keamanan adalah perhatian utama dari operasi android. perangkat. Itu tidak memungkinkan aplikasi eksternal untuk mengubah atau memodifikasi *file* yang terinstal (Sarkar dkk., 2019).

Android memiliki beberapa kelebihan yang membuat android banyak diminati oleh khalayak umum, berikut adalah kelebihan dari android (Verawati, 2019) :

- *User Friendly* yaitu sistem android mudah dijalankan sehingga bagi pengguna yang belum terbiasa menggunakannya hanya membutuhkan waktu yang singkat untuk mempelajari sistem android.
- Adanya notifikasi, pengguna akan sangat mudah dalam mengetahui info terkini karena mendapat beragam notifikasi dari smartphone dengan mengatur beberapa akun yang dimiliki seperti SMS, e-mail, *voice dial*, dan fitur atau aplikasi lainnya.

- Android mengusung konsep dan teknologi iOS hanya saja android merupakan versi murah dari iOS sehingga tampilan sistem android tidak kalah baik dan menarik dibanding dengan iOS (*Apple*).
- Konsep *open-source*, pengguna dapat bebas mengembangkan sistem android versi miliknya sendiri. Sehingga akan banyak sekali custom ROM yang bisa pengguna gunakan.
- Tersedia berbagai pilihan aplikasi menarik mulai dari aplikasi gratis hingga aplikasi berbayar. Pengguna dapat mendownload aplikasi tersebut di Google Playstore yang sudah tersedia pada smartphone pengguna.

Selain memiliki kelebihan, android juga memiliki kekurangan. Berikut ini adalah beberapa kekurangan yang dimiliki android :

- Android memiliki banyak tipe dan merek, karena hal tersebut penggunaan android menjadi tidak konsisten. Tidak seperti iPhone yang hanya memiliki 1 tipe dan dikembangkan oleh 1 pabrikan yaitu Apple.
- Karena memiliki banyak tipe dan merek, kekurangan yang sering dirasakan pengguna android adalah tidak semua *smartphone* mendapatkan *update*. Karena walaupun google rajin memperbarui android, semua *update smartphone* tetap kembali lagi pada pabrikan yang memproduksi tipe dan merek yang pengguna miliki.

Karena banyak merek dan tipe smartphone android serta *update* yang tidak sama, maka spesifikasinya juga berbeda-beda. *Smartphone* android dengan spesifikasi rendah cenderung akan lebih mudah lag dan lemot.

2.2 Kotlin

Kotlin adalah bahasa berbasis JVM yang dikembangkan oleh JetBrains. Kotlin dibuat dengan mempertimbangkan pengembang Java, dan dengan IntelliJ sebagai IDE pengembangan utamanya. Tujuan utama Kotlin adalah untuk menyediakan alternatif yang lebih aman, lebih produktif, dan lebih ringkas untuk semua platform yang saat ini menggunakan Java (Y. Shah dkk., 2018).

Selain itu Kotlin memiliki beberapa kelebihan sebagai bahasa pemrograman, menurut Volodko (2021), dengan menggunakan Kotlin untuk pengembangan Android, kita bisa mendapatkan keuntungan:

- *Less code combined with greater readability*. Menghabiskan lebih sedikit waktu untuk menulis kode Anda dan bekerja untuk memahami kode orang lain.
- *Fewer common errors*. Aplikasi yang dibuat dengan Kotlin 20% lebih kecil kemungkinannya untuk crash berdasarkan data internal Google.
- *Kotlin support in Jetpack libraries*. Jetpack Compose adalah *toolkit* modern yang direkomendasikan Android untuk membangun UI asli di Kotlin. Ekstensi KTX menambahkan fitur-fitur bahasa Kotlin, seperti coroutines, fungsi ekstensi, lambdas, dan parameter bernama ke *library* Android yang sudah ada.
- *Support for multiplatform development*. Kotlin Multiplatform memungkinkan pengembangan tidak hanya untuk Android tetapi juga iOS, backend, dan aplikasi web. Beberapa *library* Jetpack sudah *multiplatform*. Compose Multiplatform, kerangka kerja UI deklaratif JetBrains berdasarkan Kotlin dan Jetpack Compose, memungkinkan untuk berbagi UI di seluruh platform - iOS, Android, desktop, dan web.
- *Mature language and environment*. Sejak dibuat pada tahun 2011, Kotlin telah berkembang secara terus menerus, tidak hanya sebagai bahasa tetapi juga sebagai sebuah ekosistem yang utuh dengan peralatan yang kuat. Sekarang Kotlin terintegrasi dengan mulus ke dalam Android Studio dan secara aktif digunakan oleh banyak perusahaan untuk mengembangkan aplikasi Android.
- *Interoperability with Java*. Anda dapat menggunakan Kotlin bersama dengan bahasa pemrograman Java dalam aplikasi Anda tanpa perlu memigrasi semua kode Anda ke Kotlin.
- *Easy learning*. Kotlin sangat mudah dipelajari, terutama bagi para pengembang Java.
- *Big community*. Kotlin memiliki dukungan yang besar dan banyak kontribusi dari komunitas yang berkembang di seluruh dunia. Lebih dari 95% dari seribu aplikasi Android menggunakan Kotlin.

2.3 Android Studio

Android Studio adalah alat atau lingkungan pengembangan terintegrasi resmi untuk sistem operasi Android Google. Ini dibangun di atas Perangkat lunak IntelliJ IDEA dari JetBrains (Verma dkk., 2018). Android Studio adalah IDE pemrograman Android resmi dari Google yang dikembangkan oleh IntelliJ. Sebelumnya, IDE resmi pemrograman Android adalah Eclipse. Tetapi sejak kemunculan Android Studio, Google telah menjadikan Android Studio sebagai IDE resminya. Dikarenakan sudah meresmikan Android Studio, Google menghentikan *support* ADT ke Eclipse dan ADT resmi hanya didapatkan oleh Android Studio. Android Studio dipilih karena memiliki banyak fitur yang memudahkan para pembuat program terutama *programmer* level dasar. Selain memiliki banyak fitur, Android Studio juga memiliki banyak *library* yang sudah siap untuk digunakan (Mulyati & Wardono, 2019).

Android Studio memiliki banyak fitur yang dapat digunakan untuk mengembangkan aplikasi berbasis Android. Beberapa fitur yang dapat digunakan yaitu :

- Memiliki *open-source build system* yang fleksibel.
- Emulator yang mudah, cepat, dan kaya akan fitur.
- Dapat digunakan untuk mengembangkan semua jenis perangkat Android.
- Memiliki fitur Instan Run yaitu adalah fitur menjalankan aplikasi tanpa membuat APK baru.
- Mengimpor kode dengan mudah karena memiliki *template* kode dan terintegrasi dengan GitHub.
- Alat pengujian dan *frameworks* yang ekstensif
- *Lint tools* untuk menganalisis kinerja, kegunaan, kompatibilitas versi, dan masalah-masalah lain saat pembuatan aplikasi.
- Dukungan bahasa pemrograman C++ dan *Native Development Kit* (NDK).
- Dukungan bawaan untuk Google Cloud Platform, mempermudah pengintegrasian Google Cloud Messaging dan App Engine.

Di dalam Android Studio, setiap projeknya memiliki satu atau beberapa jenis modul seperti modul aplikasi, modul library, dan modul Google Cloud yang berisi *source code files* dan *resource files* (Android Developers, 2019).

2.4 Reactive Programming

Reactive programming (RP) merupakan sebuah trend dalam pemrograman pada pengembangan aplikasi. RP diusulkan sebagai solusi untuk menyederhanakan pengembangan sistem reaktif (Joyo dkk., 2020).

Reactive programming menyediakan abstraksi untuk menyederhanakan tugas bersamaan dan asinkron yang kompleks melalui ekstensi bahasa reaktif seperti *library* RxJava dan Project Reactor, paradigma reaktif hadir melalui ekstensi untuk bahasa berorientasi objek seperti java (Dobslaw dkk., 2020).

Pemrograman secara reaktif dimungkinkan dalam berbagai bahasa, dan pedomannya dinyatakan dalam Reactive Manifesto, Ini adalah paradigma yang digerakkan oleh peristiwa, dan menurut Reactive Manifesto, pemrograman reaktif harus *non-blocking* di penyampaian pesan asinkronnya (Bonér dkk., 2023).

Pemrograman reaktif membawa beberapa manfaat bagi arsitektur perangkat lunak modern. Keunggulan ini membantu pengembang membuat aplikasi yang lebih efisien, responsif, dan dapat diskalakan, sekaligus menyederhanakan penanganan kesalahan dan pemeliharaan aplikasi yang kompleks. Manfaat utama meliputi:

- Peningkatan kinerja, pemrograman reaktif dapat secara signifikan meningkatkan kinerja aplikasi dengan meminimalkan operasi pemblokiran, memungkinkan komponen bekerja secara bersamaan dan merespons peristiwa saat terjadi. Ini mengarah pada waktu pemrosesan yang lebih cepat dan pemanfaatan sumber daya yang lebih baik.
- Pengurangan penggunaan sumber daya, karena sifat pemrograman reaktif yang asinkron dan digerakkan oleh peristiwa, aplikasi sering menggunakan lebih sedikit sumber daya, seperti CPU dan memori. Hal ini sangat bermanfaat untuk aplikasi dengan sejumlah besar pengguna secara bersamaan, karena mengalokasikan sumber daya secara efisien menjadi semakin penting.
- Responsif yang ditingkatkan di *UI/UX*, pemrograman reaktif dapat meningkatkan responsivitas antarmuka pengguna dengan mengaktifkan komponen untuk diperbarui secara otomatis saat data berubah. Ini berarti pengguna mengalami interaksi yang lebih lancar dengan aplikasi, dan

komponen *UI* diperbarui secara *real-time*, meningkatkan pengalaman pengguna.

- Penanganan kesalahan yang disederhanakan, penanganan kesalahan dalam aplikasi reaktif dapat dipusatkan dan dikelola melalui aliran data, membuatnya lebih mudah untuk menangani dan memulihkan dari kesalahan operasional. Ini membantu memastikan bahwa aplikasi tetap stabil meskipun terjadi kesalahan.
- Skalabilitas, aplikasi reaktif dapat diskalakan dengan lebih efisien, karena dirancang untuk menangani berbagai beban kerja dan secara otomatis beradaptasi dengan perubahan permintaan. Ini membantu memastikan bahwa aplikasi dapat terus memenuhi persyaratan kinerja meskipun jumlah pengguna atau kompleksitas sistem meningkat.
- Pemeliharaan aplikasi kompleks yang lebih mudah, pemrograman reaktif menyederhanakan proses penulisan, pemeliharaan, dan *debugging* aplikasi kompleks dengan mempromosikan sambungan longgar antar komponen dan mengikuti pendekatan deklaratif, berbasis peristiwa.
- Pengelolaan *backpressure* yang lebih baik agar menghindari *overload* pada sistem dengan mengelola aliran data secara efisien.

2.5 Kotlin Flow

Kotlin Flow adalah sebuah *library* dalam bahasa pemrograman Kotlin yang memungkinkan pemrogram untuk bekerja dengan aliran data *asinkronik* yang dapat diobservasi secara berurutan. Flow memungkinkan pemrogram untuk mengirimkan nilai-nilai secara bertahap dan dapat mengubah aliran data dengan menggunakan operator yang tersedia (Elizarov, 2021). Kotlin adalah sebuah bahasa pemrograman yang berjalan di atas *Java Virtual Machine* dikembangkan oleh JetBrains, Kotlin merupakan bahasa pemrograman yang cocok untuk dikembangkan dalam android dan mendapatkan dukungan/rekomendasi resmi dari Google selaku pengembang resmi dari sistem operasi android. Kotlin juga dikembangkan dari bahasa pemrograman Java secara *behavior* gaya penulisan di Kotlin tidaklah jauh berbeda dengan Java yaitu menggunakan konsep *Object Oriented Programming* (OOP).

Sama seperti Java kotlin juga mampu membuat aplikasi berbasis dekstop, web, dan penanganan *backend* lainnya. (S. Doug, 2010)

Kotlin Flow merupakan fitur yang sangat berguna dalam pengembangan aplikasi Android, karena memungkinkan pengelolaan aliran data secara asinkron dan reaktif. Dalam penelitian yang dilakukan oleh U. Sarker, M. R. Islam, dan S. K. Dey (2020), mereka menjelaskan tentang penggunaan Kotlin Flow dalam pengembangan aplikasi Android yang responsif dan efisien. Mereka menunjukkan bagaimana Kotlin Flow dapat digunakan untuk mengelola aliran data yang kompleks, mengurangi *blocking thread*, dan meningkatkan kinerja aplikasi. Penelitian ini memberikan pemahaman yang mendalam tentang penggunaan Kotlin Flow dalam konteks pengembangan aplikasi Android. Kotlin memilik kelebihan sebagai berikut, yaitu sebagai berikut :

- Memiliki kemampuan *Null Safety*, sehingga meminimalisir sebuah *error* yang disebut *NullPointerException.*
- Penulisan kode yang lebih ringkas dibanding Jawa.
- Dukungan IDE dari JetBrains sebagai pengembang resmi (R.K. Panchal dan A. K. Patel, 2017).

2.6 RxJava

RxJava adalah sebuah *library* pemrograman reaktif yang populer dalam pengembangan aplikasi Android. Dalam sebuah penelitian yang dilakukan oleh A. Olowonmi, P. Lai, dan S. Xu (2019), mereka mempelajari penggunaan RxJava dalam pengembangan aplikasi Android untuk meningkatkan kinerja dan responsivitas. Penelitian ini menyoroti bagaimana RxJava dapat digunakan untuk mengelola aliran data asinkron, mengurangi kompleksitas kode, dan meningkatkan produktivitas pengembang. Jurnal ini memberikan wawasan yang baik tentang penggunaan RxJava dalam konteks pengembangan aplikasi Android.

Dikarenakan RxJava adalah sebuah *library* yang populer dan *robust* untuk *reactive programming* di lingkungan Android. *library* ini memungkinkan pengembang untuk menghadapi kompleksitas aliran data asinkron dengan cara yang efisien dan mudah dibaca. Dalam dokumentasinya, RxJava memiliki fitur-fitur utama sebagai berikut:

a. Observables dan Observers: Observable dan Observer dalam RxJava.
b. Operators: Operator-operator yang tersedia dalam RxJava.
c. Schedulers: Mengatur eksekusi aliran data dalam RxJava.

RxJava merupakan sebuah *framework* untuk menerapkan paradigma *reactive* untuk bahasa pemrograman Java. RxJava pada bagian tertentu diimplementasikan dengan functional programming yang bersifat *declarative, thread-safe,* dan *testable* (C. Arriola and A. Huang, 2017). *Framework* ini digunakan untuk menangani beberapa informasi secara reaktif agar proses manipulasi yang rumit pada *user interface* dapat ditangani dengan mudah (T. Subonis, 2017).

Unsur-unsur pada RxJava adalah:

- ***Observables***, merupakan sumber dari semua data dan struktur inti atau kelas yang akan kita kerjakan. Pada dasarnya *observable* bertugas untuk mengirim serta mengamati data yang disebarluaskan. *Observable* dalam RxJava terdapat dalam sebuah kelas yang memiliki nama yang sama yaitu kelas *Observable*. *Observable* dapat terdiri dari *observable* lain yang berbeda. Pada dasarnya *observable* adalah *interface universal* untuk memanfaatkan aliran data dengan cara yang reaktif.
- ***Disposables***, atau sering disebut sebagai *Subscription* pada RxJava 1.0. Sedangkan di RxJava 2.*0 subscription* disebut dengan *disposable*. *Disposable* adalah alat yang digunakan untuk mengontrol siklus dari *observable*. Jika aliran data yang dihasilkan *observable* tidak terbatas, hal ini tidak menjadikan masalah untuk aplikasi di server tetapi akan menimbulkan masalah pada android yang menyebabkan *memory leaks* atau kebocoran memori.
- ***Schedulers***, adalah sesuatu yang dapat menjadwalkan unit kerja untuk dieksekusi sekarang atau nanti. Pada dasarnya *schedulers* ini dapat diartikan sesuai dengan bahasa Indonesia yaitu penjadwalan. Dalam hal ini *schedulers* mengontrol di mana kode akan dieksekusi dengan memilih beberapa *thread* tertentu.

2.7 Reactor

Project Reactor adalah sebuah *reactive programming library* yang digunakan dalam bahasa pemrograman Java. Reactor menawarkan dua tipe utama, Flux dan

Mono, yang merepresentasikan urutan peristiwa yang tidak sinkron dan mengaktifkan gaya pemrograman deklaratif yang menyederhanakan aplikasi berbasis peristiwa yang kompleks. *Library* ini merupakan implementasi dari spesifikasi Reactive Streams yang menyediakan alat dan komponen untuk membangun aplikasi reaktif (*Project Reactor Documentation*, 2021). Dengan menggunakan Project Reactor, *developer* dapat memanipulasi aliran data asinkronik secara deklaratif dan komposisional. Reactor dapat digunakan dalam pengembangan aplikasi Android.

Dalam sebuah penelitian yang dilakukan oleh K. Shah, S. Shetty, dan R. Kulkarni pada tahun 2021, mereka menggali penggunaan Reactor dalam pengembangan aplikasi Android untuk meningkatkan responsivitas, skalabilitas, dan efisiensi. Penelitian ini mencakup penggunaan Reactor dalam manajemen aliran data asinkron, pemrosesan tugas secara paralel, dan pemodelan aplikasi berbasis *event-driven*. Jurnal ini memberikan wawasan yang berharga tentang penggunaan Reactor dalam konteks pengembangan aplikasi Android.

2.8 Benchmark

Benchmarking adalah salah satu metode mendasar untuk menganalisis kinerja proses komputasi atau *thread*. Dalam domain komputasi berkinerja tinggi, *benchmarks* sangat penting untuk menilai sistem komputer (Hunold dkk., 2021). *Benchmark* pada aplikasi Android penting untuk mengevaluasi kinerja aplikasi dalam berbagai aspek seperti waktu respons, penggunaan sumber daya, dan kecepatan *render*. Referensi yang dapat digunakan dalam *benchmarking* aplikasi Android adalah menggunakan alat bawaan seperti Android Profiler yang disediakan oleh Android Studio, serta alat pihak ketiga seperti Geekbench dan AnTuTu Benchmark. Proses ini penting untuk menentukan kekuatan dan kelemahan dari berbagai metode komputasi dan memberikan rekomendasi tentang metode yang paling sesuai untuk digunakan dalam situasi tertentu. Elemen Utama dalam *Benchmarking* Komputasi (Weber, L.M., Saelens, W., Cannoodt, R. et al, 2019):

- **Pemilihan Dataset**: dataset referensi adalah komponen kritis dalam *benchmarking*. *Dataset* ini bisa berupa data yang dihasilkan secara eksperimental atau data simulasi yang memungkinkan pengenalan sinyal

kebenaran dasar yang diketahui. Simulasi data harus mencerminkan sifat relevan dari data nyata untuk memberikan hasil yang berguna.

- **Pengaturan Parameter**: parameter dapat sangat mempengaruhi kinerja algoritma. Tiga strategi utama untuk pengaturan parameter adalah menggunakan nilai *default*, memilih berdasarkan pengalaman atau nilai yang dipublikasikan, dan menggunakan prosedur *tuning* parameter sistematis seperti pencarian *grid* atau validasi silang.
- **Metode Evaluasi**: Evaluasi dapat dilakukan dengan membandingkan metode terhadap standar yang diterima atau metode lain yang ada. Evaluasi juga bisa dilakukan secara kualitatif dengan menilai apakah metode tersebut dapat menghasilkan penemuan sebelumnya.

Dengan menggunakan alat-alat ini, pengembang dapat mengukur dan membandingkan kinerja aplikasi mereka dengan standar yang diharapkan atau dengan aplikasi sejenis lainnya. Contohnya, dalam sebuah penelitian yang dilakukan oleh Saini dkk., pada tahun 2018, mereka menggunakan Geekbench untuk melakukan *benchmarking* dan membandingkan kinerja aplikasi Android dalam hal kecepatan eksekusi pada berbagai perangkat.

2.9 APK

APK adalah singkatan dari *android application package*. Ini adalah format paket aplikasi *default* untuk perangkat yang berjalan di android. *File* APK pada dasarnya adalah *file* arsip *zip*. Ini termasuk semua *file* yang diperlukan yang membentuk aplikasi Anda. *Files* yang terdiri dari file APK adalah: *File* kelas Java, *File* sumber daya, dan *file* yang terdiri dari sumber daya yang dikompilasi. APK Android yang sudah diinstal sebelumnya disimpan di folder sistem/aplikasi dan yang diinstal pengguna adalah yang diinstal pengguna disimpan dalam folder data/aplikasi. Jika Anda ingin melihat isi *file* APK, Anda harus mengubah ekstensi *file* menjadi *.zip* dan membukanya (Y. Shah dkk., 2018).

APK adalah arsip yang mengandung semua komponen yang dibutuhkan untuk menjalankan aplikasi Android, seperti kode terkompilasi, aset, sumber daya, dan *file* manifes. APK pada dasarnya adalah arsip ZIP dengan struktur *file* yang spesifik, termasuk (Android Developers, 2019):

- **META-INF**: berisi *file* manifes dan sertifikat aplikasi.
- **lib**: direktori yang berisi kode terkompilasi untuk platform tertentu.
- **res**: direktori yang berisi sumber daya yang tidak dikompilasi.
- ***assets***: direktori yang berisi aset aplikasi yang dapat diakses oleh *AssetManager*.
- ***AndroidManifest.xml***: *file* manifes yang mendeskripsikan nama, versi, hak akses, dan *library* yang dirujuk oleh aplikasi.
- ***classes.dex***: kelas yang dikompilasi dalam format *file .dex* yang dijalankan oleh *Android Runtime*.
- ***resources.arsc***: *file* yang berisi sumber daya yang telah dikompilasi, seperti XML biner.

2.10 Android Profiling

Android Profiling adalah proses mengumpulkan dan menganalisis kinerja aplikasi Android dalam berbagai aspek, seperti penggunaan CPU, penggunaan memori, penggunaan jaringan, waktu respons, dan aspek lainnya. Hasil dari *android profiling* ini bertujuan untuk melengkapi hasil dari *benchmark* dalam tes integrasi.

Alat ini bernama Android Profiler, merupakan *tool* bawaan dari aplikasi Android Studio yang melakukan pemantauan aplikasi Android secara *real-time* yang menampilkan berbagai parameter yang dibutuhkan (Almeida, 2020). Parameter yang diperoleh dari *Android Profiling* adalah *memory usage* dan *maximum of CPU percentage usage*. Seiring dengan kedua parameter ini, terdapat parameter *execution time* yang dihitung secara otomatis oleh *application code* menggunakan *time software*. Meskipun pengukuran waktu eksekusi menggunakan pendekatan ini mungkin tidak sebaik dengan pendekatan *benchmark*, namun nilai yang dihasilkan dapat menunjukkan perbedaan yang perbedaan signifikan antara RxJava dan Kotlin Coroutines (Almeida, 2020).

Profiling membantu pengembang mengidentifikasi dan memperbaiki masalah kinerja dalam aplikasi mereka. Beberapa alat utama yang disediakan oleh Android Studio untuk *profiling* adalah (Android Developers, 2024) :

- ***Memory Profiler***: alat ini membantu mengidentifikasi penggunaan memori aplikasi. *Memory Profiler*, bisa melihat berapa banyak memori yang digunakan oleh aplikasi dalam kategori yang berbeda seperti *Java, Native, Graphics,* dan lainnya.
- **CPU *Profiler***: alat ini merekam aktivitas CPU dari aplikasi Android, membantu memahami apakah aplikasi mengelola bebannya dengan baik. Profil ini menampilkan thread aktif dari aplikasi dan memplot aktivitasnya dari waktu ke waktu, sehingga dapat mengidentifikasi *thread* mana yang menggunakan waktu CPU lebih banyak dari yang seharusnya.
- ***Network Profiler***: alat ini berguna untuk aplikasi yang berinteraksi dengan banyak jaringan. *Network Profiler* mencatat urutan permintaan jaringan yang dikirim dan diterima, data yang dipertukarkan, serta kecepatan jaringan saat interaksi terjadi. Ini membantu mengidentifikasi permintaan jaringan yang gagal atau *endpoint* yang membutuhkan waktu lebih lama dari biasanya.
- ***Battery Profiler***: dikenal juga sebagai *Energy Profiler*, alat ini membantu memvisualisasikan dampak penggunaan baterai oleh aplikasi dari waktu ke waktu. Alat ini berfungsi mengidentifikasi penggunaan energi yang berlebihan oleh aplikasi Android, seperti saat aplikasi memegang *wake lock* tanpa melakukan pemrosesan berat.

2.11 SonarQube

SonarQube adalah alat jaminan kualitas *open-source* yang dirilis oleh SonarSource pada tahun 2009 dan secara aktif dikembangkan sejak saat itu. Platform ini membantu pengguna melacak sistem mereka dan memastikan kualitas. Platform ini mendukung lebih dari 20 bahasa dan mendefinisikan berbagai metrik *source code*, aturan pengkodean pelanggaran, duplikasi kode, dan cakupan pengujian untuk memenuhi syarat sistem, dan juga memberikan perkiraan *technical debt* (Wang dkk., 2019).

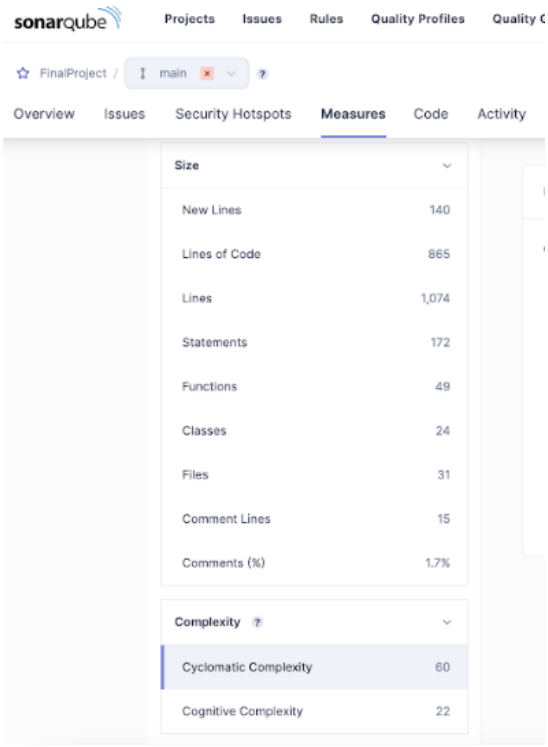
SonarQube menghitung beberapa metrik seperti jumlah baris kode dan kompleksitas kode, dan memverifikasi kepatuhan kode terhadap seperangkat "aturan pengkodean" tertentu yang ditetapkan untuk bahasa pengembangan yang paling umum. Jika kode sumber yang dianalisis melanggar aturan pengkodean atau

jika metrik berada di luar ambang batas yang telah ditentukan, SonarQube menghasilkan "masalah". SonarQube mencakup aturan Keandalan, Pemeliharaan, dan Keamanan (Lenarduzzi dkk., 2020). Fitur utama SonarQube meliputi:

- **Integrasi dengan Platform DevOps**: mendukung integrasi dengan GitHub, GitLab, Azure DevOps, dan Bitbucket.
- **Analisis Cepat**: menyediakan metrik kode bersih yang dapat ditindaklanjuti dalam hitungan menit.
- **Gerbang Kualitas Sonar**: menyediakan sinyal *go/no-go* untuk kualitas kode yang dapat dikonfigurasi, mencegah masalah kode untuk dirilis ke produksi.
- **Integrasi IDE dengan SonarLint**: menyediakan umpan balik instan di IDE favorit Anda untuk menemukan dan memperbaiki masalah kode saat menulisnya.

2.12 Analisis Statis

Analisis statis merupakan metode untuk mengevaluasi kode program dengan cara memeriksa struktur, properti, dan perilaku kode tanpa harus menjalankannya. Pendekatan ini memungkinkan identifikasi masalah potensial dalam kode sebelum aplikasi dijalankan. Metrik dari analisis statis ini diukur menggunakan SonarQube. Aplikasi SonarQube secara lokal berjalan secara *default* pada port 9000, dapat diakses melalui *browser*. (Almeida, 2020). Pada SonarQube setelah memilih projek yang ingin dianalisis, detail nilai metrik yang akan diukur dapat langsung dilihat pada tab *measures*, seperti berikut :



Gambar 1. Tampilan Tab *measures* SonarQube

Server SonarQube mengidentifikasi projek sebagai aplikasi Kotlin dan menyediakan nilai analisis yang disesuaikan dengan bahasa tersebut. Hasil yang diperoleh dari analisis statis bergantung sepenuhnya pada algoritma dan aturan SonarQube karena proses tersebut tidak dimodifikasi dalam bentuk apa pun dari konfigurasi *default*-nya (Almeida, 2020).

2.13 Cognitive Complexity

Cognitive Complexity adalah metrik yang digunakan untuk mengukur kompleksitas kode dan berfokus pada *indentation levels*. *Cognitive Complexity* memperhitungkan beban kognitif yang diperlukan untuk memahami kode tersebut. Ini mempertimbangkan faktor-faktor seperti kondisional bersarang, kompleksitas perulangan, dan struktur keseluruhan dari kode tersebut. Tujuannya adalah untuk menjaga *Cognitive Complexity* dari metode dan fungsi sekecil mungkin sambil tetap menjaga kejelasan dan keterbacaan kode. Semakin tinggi nilai *Cognitive Complexity*, semakin sulit bagi seseorang untuk memahami kode tersebut (Ganglani, 2023).

Semakin rendah nilai *Cognitive Complexity* pada suatu *code*, maka artinya kode yang ditulis lebih mudah dipahami. Hal ini berdampak pada proses pengembangan perangkat lunak secara keseluruhan. Selain itu, kode yang mudah dipahami juga membuat proses pengujian menjadi lebih efektif karena pengujian dapat dilakukan dengan lebih tepat dan menyeluruh (Ganglani, 2023).

2.14 Cyclomatic Complexity

Cyclomatic Complexity adalah salah satu metrik yang dapat kita gunakan untuk mengukur kompleksitas kode. Metrik ini diperkenalkan oleh Thomas McCabe pada tahun 1976, dan mengindikasikan kompleksitas suatu metode dengan satu angka. *Cyclomatic Complexity* mengukur seberapa banyak jalur yang berbeda yang dapat diambil dalam suatu program, tergantung pada struktur kontrol seperti percabangan (*if-else*), perulangan (*loop*), dan panggilan fungsi (Baron, 2019).

Cyclomatic Complexity yang tinggi menunjukkan bahwa kode mungkin sulit dipahami, diuji, dan dipelihara. Hal ini sering menandakan perlunya refaktorisasi untuk menyederhanakan kode. SonarQube menandai kompleksitas tinggi sehingga pengembang dapat mengidentifikasi area yang perlu ditingkatkan. Dengan mengidentifikasi area dengan kompleksitas tinggi, SonarQube dapat memberi peringatan kepada *developer* tentang kemungkinan titik-titik *bug*, mendorong mereka untuk meninjau dan mungkin menulis ulang kode. Dengan menggunakan *Cyclomatic Complexity* sebagai salah satu metrik, SonarQube membantu memastikan bahwa kode yang dihasilkan memiliki struktur yang lebih sederhana, lebih mudah dipelihara, dan lebih dapat dipahami, sehingga meningkatkan kualitas dan keandalannya (Campbell, 2023).

2.15 Lines of Code (LOC)

Lines of Code (LOC) adalah metrik sederhana yang menghitung jumlah baris kode dalam suatu program, mengacu pada jumlah baris fisik yang berisi setidaknya satu karakter yang bukan merupakan spasi kosong, tabulasi, atau bagian dari komentar (*SonarQube source*, 2024).

Lines of Code memberikan gambaran kasar tentang seberapa besar program tersebut. Ukuran kode adalah faktor yang penting dalam menilai kompleksitas dan keberlanjutan proyek perangkat lunak. *File* dengan jumlah *Lines of Code* yang tinggi cenderung lebih kompleks dan sulit dipahami. Dengan melacak *Lines of Code*, SonarQube dapat mengidentifikasi *file* yang mungkin perlu ditinjau lebih lanjut untuk memastikan kualitas dan konsistensinya. *File* dengan jumlah *Lines of Code* yang tinggi mungkin memerlukan perhatian lebih dalam hal refaktorisasi dan pemeliharaan. Ada sejumlah pengecualian yang perlu diperhatikan saat menghitung *Lines of Code* (*SonarQube source*, 2024):

1. Baris kode uji tidak dihitung sebagai *Lines of Code*.
2. Kode dalam bahasa yang tidak didukung tidak dihitung sebagai *Lines of Code*. Namun, jika Anda menjalankan plugin pihak ketiga yang memperkenalkan dukungan (untuk bahasa tersebut), kode yang baru dianalisis akan dihitung sebagai *Lines of Code*.
3. Komentar atau baris kosong tidak dihitung sebagai *Lines of Code*.