Analisis Performa BloC dan Riverpod State Management Library Pada Flutter Untuk Perangkat Lunak Berbasis Android

Proposal Tugas Akhir

Kelas MK Penulisan Proposal (CCH4A3)

1301213238
ANGGIAT MARASI B.S HASUGIAN

Program Studi Sarjana Informatika
Fakultas Informatika
Universitas Telkom
Bandung
2024

Lembar Persetujuan

Analisis Performa BloC dan Riverpod State Management Library Pada Flutter Untuk Perangkat Lunak Berbasis Android

Performance Analysis of BloC and Riverpod State Management Libraries in Flutter for Android-based Software

NIM :1301213238

ANGGIAT MARASI B.S HASUGIAN

Proposal ini diajukan sebagai usulan pembuatan tugas akhir pada
Program Studi Sarjana S1 Informatika
Fakultas Informatika Universitas Telkom

Bandung, 17/Mei/2024
Menyetujui

Calon Pembimbing 1	Calon Pembimbing 2
MONTERICO ADRIAN, S.T., M.T	SHINTA YULIA PUSPITASARI, M.T
20870024	18880124

ABSTRAK

Dalam era teknologi yang berkembang pesat, aplikasi *mobile* yang efisien dan responsif menjadi kebutuhan mendesak. *Flutter,* kerangka kerja dari *Google,* populer berkat kemampuannya menciptakan antarmuka dinamis. Pengelolaan *state* dalam aplikasi kompleks memegang peranan penting dalam efisiensi pelacakan dan pembaruan *state.*

Penelitian ini membandingkan efisiensi dua metode *state management Flutter,* BloC dan Riverpod, dalam aplikasi informasi gempa bumi yang menampilkan data *real-time*. Melalui pengembangan dua aplikasi serupa, masing-masing menggunakan BloC dan Riverpod, penelitian ini mengukur penggunaan *CPU* dan memori sebagai indikator efisiensi. Pengujian dilakukan menggunakan *Android Studio Profiler* dalam skenario terkontrol.

Hasil menunjukkan perbedaan signifikan dalam penggunaan sumber daya antara kedua metode, memberikan wawasan tentang *state management* yang lebih efisien untuk aplikasi informasi gempa bumi. Temuan ini diharapkan membantu pengembang memilih state *management* yang tepat untuk aplikasi dengan dibutuhkan.

1. PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi perangkat lunak semakin pesat seiring dengan kebutuhan akan aplikasi yang lebih canggih dan efisien. Salah satu platform populer untuk pengembangan perangkat lunak *mobile* adalah *Flutter*, yang dikembangkan oleh *Google* [1]. *Flutter* adalah *framework open-source* yang memungkinkan pengembang membuat perangkat lunak *mobile* dengan antarmuka pengguna yang menarik dan responsif. Perangkat lunak *Flutter* dibangun dari komponen *User Interface (UI)* yang disebut *widget*, yang mematuhi standar desain aplikasi *modern* dan menyederhanakan proses pembuatan antarmuka pengguna. *Widget* ini dapat berubah tampilan sesuai dengan konfigurasi dan *state* yang dimilikinya [2], di mana *state* merupakan representasi dari data dinamis yang mempengaruhi tampilan di layar [3].

Namun, dengan semakin banyaknya *widget* atau kompleksitas perangkat lunak, pelacakan dan pembaruan *state* menjadi lebih sulit [4]. Semakin sedikit *widget* yang perlu dibangun kembali, semakin efisien penggunaan sumber daya [3], yang berdampak positif pada performa perangkat lunak. Penelitian sebelumnya menunjukkan bahwa metode *state management* yang berbeda mempengaruhi performa perangkat lunak [14].

Pentingnya performa aplikasi semakin meningkat pada aplikasi yang menampilkan data *real-time*, seperti aplikasi yang akan dibangun yaitu informasi tentang gempa bumi. Dalam konteks dimana aplikasi harus segera menyajikan informasi gempa bumi kepada pengguna, kinerja yang cepat dan responsif sangat penting untuk memungkinkan pengguna mengambil keputusan yang tepat waktu. Aplikasi harus dapat memproses dan menampilkan data dengan efisiensi dan responsif terhadap perubahan *state* dengan cepat, sehingga pengguna selalu mendapatkan informasi terkini tanpa jeda. Menurut survei Apigee [16], 98% responden menganggap performa sebagai prioritas utama, dengan masalah seperti *freeze, crash,* dan *respons* lambat menjadi penyebab utama ulasan negatif. Keterlambatan dalam menerima informasi penting akibat kinerja aplikasi yang buruk dapat berdampak serius terhadap keselamatan pengguna. Dengan

memanfaatkan teknik-teknik *state management* yang tepat, kita dapat mengurangi beban pada sistem, mempercepat waktu respons, dan meningkatkan pengalaman pengguna secara keseluruhan [5].

Menurut informasi dari *official package repository* resmi *Flutter*, BLoC adalah *library state management* yang populer di kalangan *developer* [6]. Riset sebelumnya [13] menunjukkan BLoC unggul dalam performa dibandingkan Provider dan setState(). Namun, belum ada pengujian terhadap Riverpod sebagai pendekatan *state management modern*. Oleh karena itu, perlu dilakukan pengujian terhadap Riverpod untuk memahami bagaimana performanya dibandingkan BLoC.

Dengan berkembangnya teknologi dan metode *state managemen*t, evaluasi *state management* yang lebih *modern* seperti Riverpod sangat diperlukan [8]. Riverpod menawarkan fleksibilitas dan kesederhanaan yang menarik bagi pengembang, terutama dalam pengembangan aplikasi *real-time* seperti aplikasi informasi gempa bumi, di mana kecepatan dan efisiensi sangat mempengaruhi kepuasan pengguna.

1.2. Perumusan Masalah

Berdasarkan latar belakang, performa perangkat lunak yang tidak optimal dapat menyebabkan aplikasi *mobile* mengalami kinerja buruk. Dalam konteks aplikasi informasi gempa bumi, membutuhkan kinerja yang cepat dan responsif sangat penting untuk keselamatan pengguna. Oleh sebab itu, diperlukan analisis untuk menemukan metode *state management* yang optimal. Adapun rumusan masalah dalam penelitian ini adalah:

- Bagaimana performa BloC dan Riverpod dalam mengelola state pada aplikasi Flutter, terutama dalam konteks aplikasi informasi gempa bumi?
- Bagaimana mengevaluasi performa aplikasi *mobile* berbasis *Flutter* yang menggunakan Riverpod dibandingkan dengan BLoC dalam hal efisiensi penggunaan sumber daya?

Untuk batasannya, penelitian ini akan membandingkan metode *state management* Riverpod dan BLoC hanya pada aplikasi *mobile* berbasis Flutter. Aspek performa aplikasi mobile yang diukur dalam penelitian ini mencakup *memory usage* dan *CPU usage.*

1.3. Tujuan

Penelitian ini bertujuan untuk membandingkan dan menganalisis performa perangkat lunak berbasis *Android* dengan menggunakan *library state management* BloC dan Riverpod pada Flutter. Khususnya, penelitian ini berfokus pada penggunaan *CPU* dan memori di berbagai ukuran *dataset* dan skenario pengujian yang mencerminkan berbagai tingkat kompleksitas aplikasi. Hasil analisis performa ini akan memberikan wawasan berharga bagi pengembang aplikasi dalam memilih *state management* yang paling cocok untuk mengembangkan aplikasi mereka, sehingga dapat meningkatkan efisiensi dan efektivitas pengembangan aplikasi *mobile.*

1.4. Rencana Kegiatan

Rencana Kegiatan yang akan dilakukan pada penelitian ini mencakup beberapa tahap, yang meliputi :

1. Kajian Pustaka

   Tahap ini melibatkan pengumpulan dan analisis literatur terkait manajemen *state* dalam pengembangan perangkat lunak Flutter, dengan fokus pada BLoC dan Riverpod. Sumber-sumber seperti jurnal dan dokumentasi resmi akan menjadi acuan utama.
2. Pengumpulan data

   Pengumpulan pada tahap ini diperoleh melalui eksperimen yang dirancang untuk mengukur metrik performa konsumsi memori, penggunaan *CPU*.
3. Rancangan penelitian
   a. Pengembangan sistem : Membangun perangkat lunak menggunakan Flutter dengan implementasi BLoC dan

Riverpod secara terpisah.

b. Pengujian Eksperimental: Melakukan serangkaian pengujian pada perangkat lunak untuk mengumpulkan data performa dari kedua metode *state management.*

c. Analisis Data: Mengolah data yang terkumpul untuk menentukan analisis BLoC dan Riverpod dalam pengelolaan *state.*

d. Evaluasi: Membandingkan hasil pengujian untuk menilai kelebihan dan kekurangan masing-masing *library state management* BloC dan Riverpod.

4. Penyusunan laporan penelitian yang mencakup metodologi, hasil analisis, dan rekomendasi berdasarkan temuan.

1.5. Jadwal Kegiatan

Jadwal kegiatan penelitian ini dirancang untuk memastikan setiap tahapan penelitian dapat diselesaikan tepat waktu. Berikut adalah rincian jadwal kegiatan:

Kegiatan	**Bulan**					
Pengembangan sistem	1	2	3	4	5	6
Pengujian Eksperimental						
Analisis Data						
Evaluasi						
Penyusunan Laporan						

2. KAJIAN PUSTAKA.

2.1. Penelitian Terkait

Penelitian yang dilakukan oleh Mgs. M. Fakhri Abdillah [3] pada tahun 2023, berjudul “Analisis Performa GetX dan BLoC *State Management Library* Pada Flutter Untuk Perangkat Lunak Berbasis Android” memberikan kontribusi penting dalam pemahaman tentang efisiensi penggunaan *library state management* dalam pengembangan perangkat lunak. Hasil yang menunjukkan GetX memiliki penggunaan *CPU* dan memori yang lebih rendah dibandingkan dengan BLoC, tetapi dengan konsumsi energi yang serupa, memberikan dasar untuk pertimbangan lebih lanjut dalam pemilihan *library state management* yang tepat untuk kasus penggunaan tertentu.

Penelitian kedua yang dilakukan oleh Regawa Rama Prayoga et al [13] pada tahun 2021, berjudul "*Performance Analysis of BLoC and Provider State Management Library on Flutter*" memberikan kontribusi penting dalam pemahaman tentang efisiensi penggunaan library *state management* dalam pengembangan perangkat lunak Flutter untuk Android. Hasil yang menunjukkan bahwa BLoC dan Provider memiliki efisiensi yang lebih baik dalam penggunaan *CPU* dan memori serta waktu eksekusi dibandingkan dengan metode *setState* bawaan, memberikan dasar untuk pertimbangan lebih lanjut dalam pemilihan *library state management* yang tepat untuk kasus penggunaan tertentu.

2.2. BloC

Business Logic Component (BloC) merupakan pola desain yang digunakan dalam pengembangan Flutter untuk memisahkan logika bisnis dari antarmuka pengguna [17]. Tujuan utamanya adalah untuk meningkatkan *modularity, testability, dan reusability* pada kode. BloC bekerja dengan menerima event yang memicu perubahan state, kemudian mengirimkan state yang diperbarui dari BloC.

Dengan demikian, BloC mampu memisahkan antara *input* dan *output*. Setelah *event* diberikan ke *input*, BloC akan menghasilkan *state* pada *output* yang kemudian dapat diamati oleh *widget UI* yang bereaksi terhadap perubahan tersebut .

Salah satu persamaan utama antara BloC dan Riverpod yang mempengaruhi performa aplikasi adalah pendekatan reaktif dalam mengelola state. BloC menggunakan *streams* untuk mengirim dan menerima perubahan *state,* yang memungkinkan pengelolaan *state* secara efisien dan responsif terhadap interaksi pengguna [18]. Hal ini penting untuk menjaga performa aplikasi tetap optimal, karena perubahan *state* dapat diproses dan disampaikan ke *UI* tanpa penundaan yang signifikan .

Selain itu, dukungan terhadap *dependency injection* juga menjadi faktor penting dalam performa aplikasi. Dengan menggunakan *context.read* atau *context.watch* , BloC memungkinkan pemisahan logika bisnis dari *UI,* yang membuat komponen-komponen aplikasi menjadi lebih modular dan mudah untuk diuji [20]. *Dependency injection* membantu dalam mengelola dependensi secara efisien, sehingga mengurangi *overhead* dan meningkatkan performa keseluruhan aplikasi. Pengelolaan *state* yang terstruktur melalui *cubit* atau *BloC* juga mempengaruhi performa aplikasi. Dengan memisahkan logika bisnis dari layer presentasi, BloC memastikan bahwa UI hanya di *update* ketika ada perubahan *state* yang relevan, menghindari rendering ulang yang tidak perlu dan meningkatkan efisiensi aplikasi .

2.3. Riverpod

Riverpod adalah sebuah *library* yang dikembangkan untuk mengelola *state* dalam aplikasi Flutter. Dengan menggunakan Riverpod, pengembang dapat menulis logika bisnis dengan cara yang mirip dengan *Stateless widgets*, memungkinkan pengembang untuk memiliki permintaan jaringan yang otomatis di rekomputasi ketika diperlukan dan membuat logika yang mudah digunakan

kembali, *kompatibel*, dan *maintainable.* Riverpod memecahkan masalah yang ada pada library *Provider*, yaitu tidak mendukung *multiple providers* dengan tipe data yang sama [8] .

Sama seperti BloC, salah satu faktor utama yang mempengaruhi performa aplikasi adalah pendekatan reaktif dalam pengelolaan state. Riverpod menggunakan *Provider* untuk mengelola *state* secara reaktif, memungkinkan aplikasi merespons perubahan *state* dengan cepat dan efisien [8] .Riverpod juga mendukung *dependency injection*, yang merupakan persamaan lain dengan BloC yang mempengaruhi performa aplikasi. Dengan menggunakan *Provider*, Riverpod memungkinkan pengelolaan dependensi secara modular, sehingga komponen aplikasi menjadi lebih terisolasi dan mudah untuk diuji. *Dependency injection* yang efisien membantu dalam mengurangi *overhead* yang terkait dengan pengelolaan dependensi, meningkatkan performa aplikasi secara keseluruhan .

Selain itu, Riverpod menggunakan *state notifier* atau *provider* untuk mengelola *state* dan logika bisnis, yang membantu dalam memisahkan logika bisnis dari layer presentasi. Ini memastikan bahwa *UI* hanya di *update* ketika ada perubahan *state* yang relevan, menghindari *rendering* ulang yang tidak perlu dan meningkatkan efisiensi aplikasi. Dengan pendekatan *declarative programming,* Riverpod memastikan bahwa logika bisnis ditulis dengan cara yang mirip dengan *Stateless widgets*, yang dapat meningkatkan performa dan *maintainability* aplikasi [21].

2.4. State Management

State management adalah praktik penting dalam pengembangan perangkat lunak Flutter yang bertujuan untuk mengatur state *UI controls* seperti *text fields*, *radio button*, dan lainnya [9]. Hal ini diperlukan terutama dalam aplikasi yang kompleks dengan fungsionalitas seperti pemanggilan *API*, penggunaan *database*, atau *integrasi* dengan *Firebase*. Tujuannya adalah untuk memastikan efisiensi dalam melacak dan memperbarui status dari berbagai elemen *UI*, sambil menjaga agar kode tetap mudah dibaca, diuji, dan dipelihara. Terdapat berbagai pendekatan *state management* dengan masing-masing kelebihan dan kekurangannya. Sebagai

contoh, *Redux* menawarkan performa tinggi tetapi kompleks, sementara *Scoped Model* lebih mudah dipahami namun kurang efisien dalam performa dan skalabilitas. Selain itu, *state management* juga memungkinkan pemisahan antara *logic* dan *view* untuk memastikan kode dapat digunakan kembali [4]. Dengan memanfaatkan berbagai *library state management* yang tersedia di Flutter, pengembang dapat lebih efisien mengelola status perangkat lunak mereka.

2.5. Mobile Applications performance Metrics

Menurut penelitian terkait performa *metrics* untuk perangkat lunak mencakup berbagai aspek penting diantaranya [12] :

- *CPU Usage*

  Processor atau *Central Processing Unit (CPU)* merupakan komponen krusial yang mengelola instruksi dari perangkat lunak aplikasi. Dalam evolusi telepon seluler, terjadi transisi signifikan dari penggunaan *prosesor single-core* menuju konfigurasi *dual-core* dan *quad-core*. Implementasi chip *multi-core* menawarkan keuntungan substansial, memungkinkan distribusi beban kerja di antara berbagai inti *prosesor*, sehingga meningkatkan kemampuan *multitasking* dan efisiensi pemrosesan. Walaupun penambahan inti *processor* berpotensi meningkatkan kecepatan pemrosesan, performa keseluruhan prosesor dan perangkat tidak hanya ditentukan oleh jumlah inti. Faktor-faktor lain seperti kapasitas *RAM* dan optimisasi perangkat lunak juga berperan penting dalam menentukan kecepatan perangkat. Oleh karena itu, dalam pengembangan aplikasi *mobile*, sangat penting untuk merancang aplikasi yang dioptimalkan untuk memanfaatkan kemampuan *processor multi-core*, guna mencapai efisiensi pemrosesan yang optimal.
- *Memory Usage*

  Kapasitas penyimpanan pada perangkat menentukan jumlah aplikasi yang dapat diinstal serta volume konten digital yang dapat disimpan. Secara paralel, *RAM* yang berfungsi menyimpan data sementara saat aplikasi berjalan, juga memiliki batasan kapasitas yang mempengaruhi

performa aplikasi. Oleh karena itu, penting bagi pengembang perangkat lunak *mobile* untuk mengoptimalkan penggunaan memori agar efisien. Perangkat lunak yang memerlukan memori berlebih dapat membebani perangkat keras, yang pada gilirannya dapat menurunkan kinerja dan responsivitas aplikasi. Dalam konteks pengembangan perangkat lunak *mobile*, efisiensi memori adalah kunci untuk menjaga performa aplikasi tetap tinggi dan pengalaman pengguna yang lancar.

- *Delay*

  Dalam konteks pengalaman pengguna pada perangkat *mobile*, responsivitas aplikasi merupakan faktor kritis. Berdasarkan hasil survei, *respons* yang diperoleh dalam waktu kurang dari 100 milidetik cenderung dipersepsikan sebagai instan oleh pengguna. Di sisi lain, respons yang memerlukan waktu satu detik atau lebih dapat dirasakan sebagai penundaan dalam eksekusi aplikasi. Penundaan ini dapat diakibatkan oleh berbagai faktor, termasuk volume data yang ditransfer dari perangkat *mobile* ke *server* dan efisiensi *server* dalam menangani permintaan tersebut. Oleh karena itu, penting bagi *developer* untuk memperhatikan aspek-aspek ini saat merancang dan mengoptimalkan aplikasi guna memastikan pengalaman pengguna yang lancar dan responsif.

- *Battery Lifetime*

  Manajemen konsumsi energi merupakan aspek kritikal dalam pengoperasian perangkat *mobile* yang bergantung pada baterai sebagai sumber daya. Aktivasi aplikasi memicu pengaktifan berbagai komponen perangkat, yang mana baterai kemudian dialokasikan untuk mendukung proses yang berjalan. Konsumsi energi ini secara signifikan dipengaruhi oleh bagaimana aplikasi dirancang dan bagaimana ia memanfaatkan sumber daya perangkat. Penggunaan intensif siklus *CPU* atau *GPU*, terutama selama tugas-tugas yang memerlukan pemrosesan grafis atau perhitungan kompleks, dapat meningkatkan konsumsi energi dan menghasilkan panas berlebih. Oleh karena itu, penting bagi *developer* untuk mengoptimalkan perangkat

lunak dengan mempertimbangkan efisiensi energi, sehingga meminimalkan beban pada *CPU* dan *GPU* dan memperpanjang umur baterai perangkat.

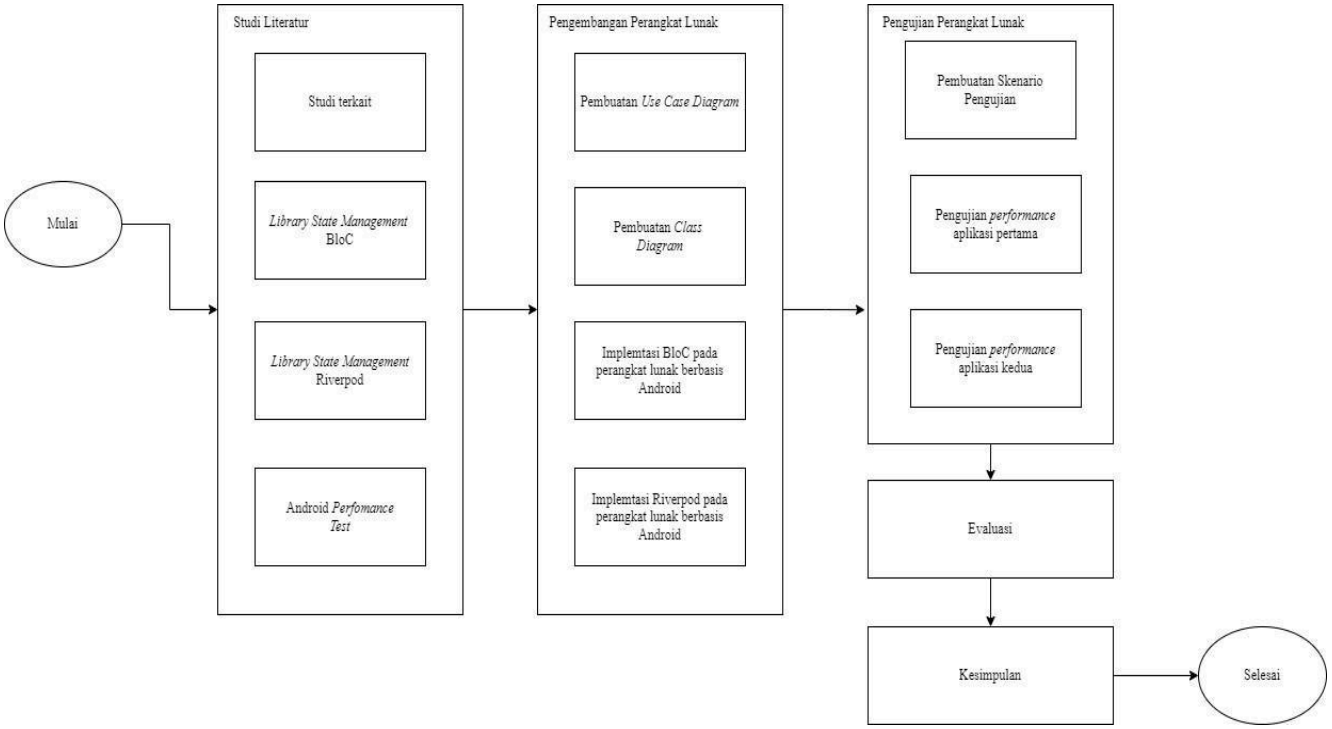
- *Network*

  Mengingat perangkat *mobile* sering dibawa kemana-mana, koneksi internet yang stabil dan terus-menerus tidak selalu mungkin. Perangkat ini sering berganti antara berbagai jenis koneksi seperti *3G, 4G, dan Wi-Fi*, yang memiliki kecepatan yang berbeda-beda. Koneksi yang tidak stabil, terutama di jaringan seluler yang sibuk, dapat menyebabkan masalah pada aplikasi *mobile*, termasuk pemuatan gambar yang lambat, aplikasi yang berhenti merespons, atau bahkan aplikasi yang tiba-tiba tertutup. Penting bagi pengembang untuk mempertimbangkan kondisi koneksi yang berubah-ubah ini saat merancang aplikasi, agar aplikasi tetap dapat berfungsi dengan baik dalam berbagai kondisi jaringan.

2.6. Android Profiler

Android Profiler dalam *Android Studio* merupakan sebuah alat yang menyediakan informasi secara *real-time* mengenai penggunaan sumber daya *CPU*, memori, jaringan, dan baterai oleh sebuah perangkat lunak [11]. Alat ini secara akurat memonitor ketiga sumber daya tersebut, yang memungkinkan pengembang untuk mendapatkan pemahaman yang lebih baik tentang interaksi aplikasi dengan sistem operasi.

3. PERANCANGAN SISTEM



3.1. Studi Literatur

Studi literatur dilakukan sepanjang pengerjaan penelitian ini untuk mempelajari konsep dan penerapan *state management* dalam pengembangan aplikasi *mobile* menggunakan *Flutter.* Literatur yang dipelajari mencakup metode BloC dan Riverpod, serta penelitian terkait yang membahas performa dan efisiensi penggunaan *CPU* dan memori. Dengan mempelajari penelitian sebelumnya, diharapkan dapat diperoleh pemahaman yang mendalam mengenai kelebihan dan kekurangan dari masing-masing *metode state management*.

3.2. Pengembangan Aplikasi

Dalam tahap ini, dua aplikasi *mobile* berbasis *Android* dikembangkan menggunakan Flutter:

- Aplikasi pertama: Dikembangkan dengan menggunakan metode BloC sebagai *state management.*

- Aplikasi kedua: Dikembangkan dengan menggunakan metode Riverpod sebagai *state management.*

Aplikasi yang dikembangkan adalah aplikasi informasi gempa bumi, yang menampilkan data *real-time* mengenai kejadian gempa. Aplikasi ini membutuhkan kinerja yang cepat dan responsif untuk memungkinkan pengguna mengambil keputusan yang tepat waktu. Kedua aplikasi dirancang dengan fitur yang sama untuk memastikan kesetaraan dalam pengujian. *Dataset* yang digunakan diambil dari *Public API* untuk menyediakan data yang konsisten dan relevan bagi aplikasi

3.3. Pengujian pada Aplikasi

Penelitian ini akan menguji dua perangkat lunak *mobile* yang telah dikembangkan. Sebelum pengujian dimulai, penulis akan menyusun skenario pengujian. Pengujian akan berfokus pada aspek performa, dengan data yang diambil meliputi penggunaan *CPU* dan memori. *Android Studio Profiler* akan digunakan untuk menguji performa, dipilih karena kemampuannya merekam penggunaan *CPU* dan memori selama aplikasi beroperasi.

3.4. Analisis Hasil Pengujian dan Kesimpulan

Hasil pengujian dianalisis untuk membandingkan efisiensi penggunaan sumber daya antara metode BloC dan Riverpod. Analisis dilakukan dengan melihat data penggunaan *CPU* dan memori yang diperoleh selama pengujian. Tujuan dari analisis ini adalah untuk menentukan apakah terdapat perbedaan signifikan dalam efisiensi performa antara kedua metode *state management* tersebut. Hasil analisis diharapkan dapat memberikan rekomendasi mengenai metode *state management* yang lebih efisien digunakan dalam pengembangan aplikasi mobile berbasis Flutter.

DAFTAR PUSTAKA

[1] "Flutter. (2024). Build apps for any screen." *https://flutter.dev/* [Accessed May. 15, 2024].

[2] "Introduction to widgets | Flutter." *https://docs.flutter.dev/development/ui/widgets -intro* [Accessed May. 17, 2024].

[3] Husain, I., Purwantoro, P., & Carudin, C. (2023). Analisis Performa State Management Provider Dan Getx Pada Aplikasi Flutter. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *7*(2), 1417-1422.

[4] Abdillah, M. M. F., Sardi, I. L., & Hadikusuma, A. (2023). Analisis Performa GetX dan BLoC State Management Library Pada Flutter Untuk Perangkat Lunak Berbasis Android. *LOGIC: Jurnal Penelitian Informatika*, *1*(1), 73-78.

[5] Slepnev D. State management approaches in Flutter. 2020.

[6] List of state management approaches — docs.flutter.dev;. [Accessed 15-03-2024]. https://docs.flutter.dev/data-and-backend/state-mgmt/options

[7] Zulistiyan, M., Adrian, M., & Wibowo, Y. F. A. (2024). performance Analysis of BLoC and GetX State Management Library on Flutter. *Journal of Information System Research (JOSH)*, *5*(2), 583-591.

[8] "Why Riverpod | Flutter." https://riverpod.dev/docs/introduction/why_riverpod [Accessed 17-05-2024].

[9] "State class - widgets library - Dart API — api.flutter.dev" https://api.flutter.dev/flutter/widgets/State-class.html#:~:text=State%20is%20information%20that%20(1,such%20state%20changes%2C%20using%20State [Accessed 16-05-2024].

[10] "Differentiate between ephemeral state and app state - Flutter." [Online]. https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app [Accessed 16-05-2024].

[11] "Profile your app performance — Android Studio https://developer.android.com/studio/profile [Accessed 16-05-2024].

[12] Jakimoski, Kire. (2018). Performance Evaluation of Mobile Applications.

[13] Prayoga RR, Syalsabila A, Munawar G, Jumiyani R. Performance Analysis of BLoC and Provider State Management Library on Flutter. Jurnal Mantik. 2021;5(3):1591-7

[14] Azis, M. H. N., Pinandito, A., & Maghfiroh, I. S. E. (2023). Analisis Perbandingan Penggunaan State Management pada Aplikasi Ditonton menggunakan Framework Flutter. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *7*(1), 148-153.

[15] Hort M, Kechagia M, Sarro F, Harman M. A Survey of Performance Optimization for Mobile Applications. IEEE Transactions on Software Engineering. 2022 aug;48(8):2879-904. Available from: https://doi.org/10.1109%2Ftse.2021.3071193

[16] Apigee Survey: Users Reveal Top Frustrations that Lead to Bad Mobile App Reviews." [Online]. Available: https://finance.yahoo.com/news/apigee-survey-users-reveal-top120200656.html [Accessed 18-06-2024]

[17] "Architecture BloC | Flutter." https://bloclibrary.dev/architecture/ [Accessed 19-06-2024].

[18] "BloC Package"| Flutter" https://pub.dev/packages/bloc [Accessed 19-06-2024].

[20] GitHub - felangel/bloc: A predictable state management library that helps implement the BLoC design pattern — github.com;. [Accessed 19-06-2024]. https://github.com/felangel/bloc.

[21] GitHub - rrouseelGitl/riverpod:A reactive caching and data-binding framework. Riverpod makes working with asynchronous code a breeze— github.com;. [Accessed 19-06-2024]. https://github.com/rrousselGit/riverpod.