Logic: Jurnal Penelitian Informatika
Vol. 1, Issue 1, pp. 73–78 (2023)
doi: http://doi.org/10.25124/logic.v1i1.6479

RESEARCH ARTICLE

Analisis Performa GetX dan BLoC *State Management Library* Pada *Flutter* Untuk Perangkat Lunak Berbasis *Android*

Mgs. M. Fakhri Abdillah, Indra Lukmana Sardi* and Aristyo Hadikusuma

Fakultas Informatika, Universitas Telkom, Bandung, 40257, Jawa Barat, Indonesia
*Corresponding author: indraluk@telkomuniversity.ac.id
Received on 08 August 2023; accepted on 05 September 2023

Abstrak

Perkembangan perangkat lunak berbasis android sangatlah cepat, *framework* untuk mengembangkan perangkat lunak berbasis android pun banyak bermunculan, salah satunya *Flutter*. *Flutter* terdiri dari beberapa komponen u*ser interface* yang disebut dengan *widget*. Informasi terkait pembangunan ataupun perubahan *widget* disebut dengan *state*. Para pengembang menyadari bahwa diperlukan suatu cara untuk mengelola *state* yang disebut dengan *state management*. Pada *Flutter* terdapat berbagai *state management*, beberapa *state management* yang paling banyak disukai para pengembang antara lain GetX dan BLoC. Pengimplementasian *state management* yang berbeda dapat mempengaruhi performa perangkat lunak, maka dari itu dilakukan pengujian terhadap kedua *state management* tersebut menggunakan metode *performance testing*. Hasil pengujian menunjukkan bahwa GetX memiliki cpu *usage* 11% dan *memory usage* 111.6333 mb lebih rendah dari pada BLoC, namun memiliki *energy consumption* yang sama. Hal ini dikarenakan perbedaan cara dalam mengelola *state*.

Key words: *Gray Wolf Optimization*, *Hepatobiliary Disorders*, Prediksi, *Support Vector Machine*.

Pendahuluan

Perkembangan perangkat lunak berbasis android sangatlah cepat. Tercatat terdapat sebanyak 2.591.578 perangkat lunak yang tersedia di *play store* pada Maret 2022 [1]. Kerangka kerja ataupun *framework* untuk mengembangkan perangkat lunak berbasis android pun banyak bermunculan, salah satunya adalah *Flutter*. *Flutter* merupakan *framework multi-platform* yang dirilis oleh *google* dan bersifat *open source* [2]. Pengembangan perangkat lunak menggunakan *Flutter* terdiri dari beberapa komponen UI *(user interface)* yang disebut dengan *widget*. *Widget* akan memberikan luaran berupa tampilan yang sesuai dengan konfigurasi dan *state* saat ini. *State* merupakan informasi yang dapat dibaca secara *sinkronus* ketika *widget* dibangun dan dapat berubah selama *widget* tersebut aktif [2]. Pada saat *state* suatu *widget* berubah, maka *widget* akan membangun ulang untuk menampilkan kondisi yang sesuai dengan *state* terbaru [3].

Pada UI suatu perangkat lunak, biasanya terdiri dari beberapa *widgets* yang dapat menyebabkan banyak terjadinya perubahan *state*. Ketika terdapat banyak *widget* atau semakin kompleks suatu perangkat lunak, maka akan menyulitkan proses pelacakan dan pembaharuan *state*. Berdasarkan hal tersebut, para *developer* menyadari bahwa diperlukan suatu cara untuk mengelola *state (State Management)*. *State management* merupakan cara untuk mengelola *state* dari *UI controls* seperti *text fields*, *radio button*, dan lainnya [2]. Menurut penelitian yang dilakukan sebelumnya [4], menunjukkan bahwa penggunaan *state management* yang berbeda dapat menghasilkan perbedaan performa pada suatu perangkat lunak. Performa pada perangkat lunak berbasis android merupakan aspek yang penting, karena dapat menggambarkan bagaimana cara kerja, efisiensi, dan kenyamanan dalam penggunaannya [14]. Performa perangkat lunak yang buruk dapat mempengaruhi *responsiveness*, *launch time*, CPU *consumption*, *memory consumption*, dan *energy consumption* pada suatu perangkat android [5].

Penerapan *state management* dilakukan di beberapa *framework*, terutama pada pengembangan android salah satunya adalah *Flutter*. Pada *Flutter* terdapat beberapa *state management library* yang paling banyak disukai oleh *developer*, antara lain, GetX, *Provider*, dan BloC [6]. Hal ini dapat dilihat dari jumlah *likes* dan *stars* pada laman *resmi library* tersebut. GetX dengan 8475 *likes* [7] dan 6300 *stars* [8], *Provider* dengan 6323 *likes* [9] dan 4200 *stars* [10], dan BloC dengan 3736 *likes* [11] dan 8800 *stars* [12].

Pada penelitian sebelumnya [4] dilakukan pengujian performa terhadap *Provider* dan BloC yang dibandingkan dengan *setState*() sebagai *state management* default pada *Flutter*. Penelitian tersebut menunjukkan bahwa penggunaan BLoC menghasilkan performa yang lebih baik dibandingkan dengan *Provider* dan setState(). Namun penelitian tersebut belum melakukan pengujian terhadap GetX sebagai state management yang paling banyak disukai oleh *developer*. Oleh karena itu perlu dilakukan pengujian terhadap GetX, sehingga dapat diketahui

performansi dari ketiga *state management* yang paling banyak disukai oleh *developer*. Dikarenakan BLoC memiliki performa yang lebih baik jika dibandingkan dengan *Provider* [4], maka pada penelitian ini akan dilakukan pengujian performa terhadap GetX yang akan dibandingkan dengan BloC *state management*.

GetX merupakan *state management* yang ringan, *powerful*, dan *high performance* [13]. Namun pada penelitian yang dilakukan oleh Dmitrii Slepnev [14] menyebutkan bahwa penggunaan GetX yang kurang tepat dapat menimbulkan masalah performansi pada perangkat lunak yang dikembangkan. Penelitian tersebut menyebutkan bahwa penggunaan GetX kurang cocok pada perangkat lunak yang kompleks, namun belum melakukan pengujian performansinya. Hal ini menimbulkan suatu permasalahan baru yaitu kelebihan dari GetX yang diberikan pada laman resmi GetX [13] bertolak belakang dengan penelitian yang dilakukan oleh Dmitrii Slepnev [14] dalam performansi GetX. Untuk mengatasi hal tersebut, perlu dilakukan pengujian dan analisis performansi GetX dan BLoC. Tingkat kompleksitas perangkat lunak yang dimaksud dilihat dari jumlah *widgets*, jumlah data yang diakses, dan alur kerjanya.

Tinjauan Pustaka

Android

Android adalah sistem operasi berbasis linux yang terdiri dari beberapa *layer*, yaitu kernel, *hardware abstraction layer*, *android runtime*, *native libraries*, *application framework*, dan sistem perangkat lunak. Android dirancang untuk penggunaan pada *smartphone* dan *gadget*. Arsitektur perangkat lunak berbasis android dirancang sedemikian rupa sehingga menyederhanakan *reuse* dari komponen – komponennya [15].

Performa

Performa dalam pengembangan suatu perangkat lunak sangatlah penting. Karena performa dapat menggambarkan cara kerja, efisiensi, ataupun tingkat kenyamanan dari penggunaan suatu perangkat lunak. Penggunaan CPU (CPU *usage*), konsumsi baterai *(battery consumption)*, dan penggunaan memori *(memory usage)* merupakan faktor utama untuk menghasilkan performa yang baik pada perangkat lunak berbasis android [14].

CPU *Usage*

Central Processing Unit (CPU) merupakan perangkat keras *(hardware)* yang berfungsi untuk memproses data utama pada sebuah perangkat *(device)*. Sedangkan, CPU *usage* adalah sebuah indikator yang digunakan untuk melihat penggunaan CPU pada sebuah perangkat. Penggunaan CPU berkaitan dengan penggunaan daya pemrosesan ataupun pekerjaan yang dilakukan oleh CPU. CPU *usage* dipengaruhi oleh jaringan, tampilan, dan semua pekerjaan yang terjadi pada perangkat lunak. Dengan meminimalisir CPU *usage* maka dapat memberikan hasil yang baik dalam pengembangan sebuah perangkat lunak. Contohnya, perangkat lunak dapat berjalan dengan baik, cepat, dan dapat menghemat penggunaan baterai sebuah perangkat [14].

Memory Usage

Memori berfungsi untuk membantu CPU dalam menyimpan data dan informasi yang bersifat sementara. Sedangkan *memory usage* merupakan sebuah indikator yang digunakan untuk melihat penggunaan memori pada sebuah perangkat. Membaca dan menulis ke memori akan membuat efisiensi dan kecepatan yang lebih baik untuk sebuah perangkat lunak dibandingkan dengan ke disk [14]. Pada android terdapat sistem perlindungan untuk memori. Sistem ini disebut dengan *low memory killer* (LMK) yang memungkinkan android untuk menutup proses yang tidak aktif, sehingga dapat mengurangi penggunaan memori. Dengan sistem ini, android dapat melakukan penutupan paksa dan melakukan perlambatan kinerja perangkat lunak ketika penggunaan memori telah mencapai batasnya [14].

Energy Consumption

Energy consumption merupakan sumber daya yang penting untuk *smartphone*. *Energy consumption* terkait dengan masa pakai baterai pada *smartphone*. Energi dikonsumsi oleh berbagai komponen pada *smartphone*, termasuk CPU, *memory*, LCD, GPS, audio, layanan WiFi dan lainnya [5]. *Energy* pada *smartphone* terbatas pada baterai saja. Meskipun kemajuan teknologi pada baterai, komponen *smartphone*, dan sistem operasi telah membantu meringankan batasan ini, kemajuan tersebut tidak dapat membantu efisiensi aplikasi dalam penggunaan energi. Oleh karena itu, meningkatkan kemampuan *developer* dalam merekayasa *energy consumption* pada perangkat lunak sangatlah penting [16].

Performance Testing

Performance testing merupakan metode yang digunakan untuk menguji performa perangkat lunak. Metode ini merupakan salah satu tahap yang penting dalam proses pengembangan perangkat lunak. Pada metode ini dilakukan perhitungan terhadap kecepatan, efektivitas, *network computer program, software* maupun *hardware* [17]. Terdapat beberapa metode lain yang terkait dengan penelitian ini, antara lain, SLR *(Systematic Literature Review)* dan SDLC *(Software Development Life Cycle)*. Metode SLR bertujuan untuk mengidentifikasi, mengkaji, mengevaluasi, dan menafsirkan literatur yang relevan. Sedangkan, pada metode SDLC berfokus pada proses pengembangan perangkat lunak, dimulai dari tahap perencanaan, analisis, desain, pengembangan, pengujian, implementasi, dan pemeliharaan [18]. Kedua metode tersebut kurang cocok untuk menjadi metode utama dalam penelitian ini, karena penelitian ini berfokus pada pengujian performa perangkat lunak. Pada penelitian ini, secara tidak langsung metode SLR telah diterapkan sehingga tidak perlu dijadikan metode utama. Sedangkan metode SDLC bertujuan untuk pengembangan perangkat lunak secara keseluruhan, sesuai dengan tahapan diatas. Berdasarkan hal tersebut, metode yang digunakan pada penelitian ini adalah *performance testing*. Karena metode tersebut sesuai dengan latar belakang dan tujuan dari penelitian ini.

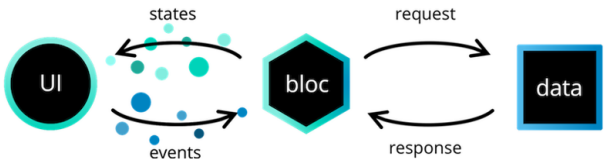
Flutter

Flutter merupakan *open source framework* yang dibangun oleh *google* untuk mengembangkan perangkat lunak yang *multi-platform*, indah, dan di *compile* secara native hanya dengan satu basis kode [2]. *Flutter* ditulis menggunakan bahasa pemrograman *Dart*. Pengembangan perangkat lunak menggunakan *Flutter* terdiri dari beberapa komponen UI *(user interface)* yang disebut dengan *widget*. *Widget* akan memberikan luaran berupa tampilan yang sesuai dengan konfigurasi dan state saat ini. *State* merupakan informasi yang dapat dibaca secara sinkronus ketika *widget* dibangun dan dapat berubah selama *widget* tersebut aktif [2].

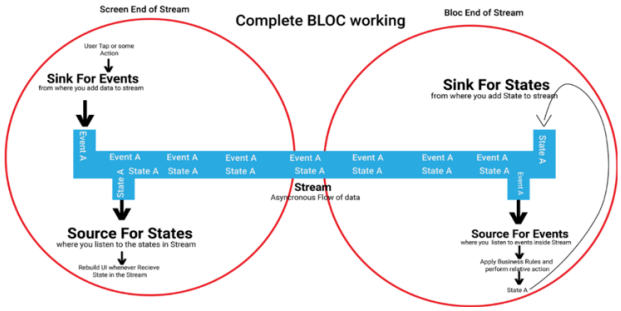
Dart

Dart merupakan bahasa pemrograman yang digunakan dalam pengembangan perangkat lunak pada *flutter*. *Dart* merupakan bahasa pemrograman alternatif yang dibangun oleh *Google* dengan mengaplikasikan *Dart virtual machine* pada *Google Chrome*. Hal ini menyebabkan *web browser* dapat mengeksekusi kode yang ditulis menggunakan bahasa pemrograman *Dart*[14].

Gambar 1. GetX *Flow Pattern.*



Gambar 2. BLoC *Flow Pattern.*



Gambar 3. Detail BLoC *WorkFlow*

State Management

State management merupakan cara untuk mengelola *state* dari UI *controls* seperti *text fields*, radio *button*, dan lainnya [2]. Selain itu, *state management* bisa juga digunakan untuk memisahkan antara *logic* dan *view* agar kode yang dikembangkan bersifat reusable [15]. Terdapat beberapa state management *library* yang tersedia pada *framework Flutter*. Pada penelitian ini, akan dilakukan pengujian terhadap GetX dan BLoC *state management*.

GetX

GetX merupakan solusi dari *reactive state management* yang disederhanakan [14]. Pada GetX terdapat beberapa fitur yang disediakan yaitu, *state management*, *routing*, dan *dependency injection*. GetX merupakan *state management library* yang paling banyak disukai saat ini. GetX memiliki tiga prinsip utama, antara lain: [7] [8].

Performance

GetX berfokus pada performa dan penggunaan sumber daya yang minim. GetX tidak menggunakan *ChangeNotifer*() atau *streams* seperti *state management* lain, sehingga memiliki performa yang baik. Pada GetX terdapat *controller*, pada *state management* lain *controller* tersebut biasanya akan dibuang *(dispose)* secara manual, namun pada GetX proses tersebut dilakukan secara otomatis sehingga dapat menghemat memori, dan baris *code*.

Productivity

GetX memiliki gaya penulisan yang mudah untuk diimplementasikan. GetX juga memiliki *boilerplate code* yang sedikit, sehingga dapat mempermudah *developer* dalam pengaplikasiannya [16]. Hal ini dapat menghemat waktu dalam pengembangan dan memberikan performa yang maksimal pada perangkat lunak. Dengan menggunakan GetX semua sumber daya yang tidak digunakan akan dibuang dari memori secara otomatis.

Organization

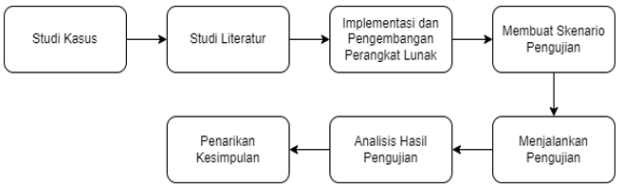
GetX memungkinkan pemisahan antara *view*, *presentation logic*, *business logic*, *dependency injection*, dan *navigation*. Dengan menggunakan GetX, pengembang dapat dengan mudah mencari setiap fitur yang tersedia pada perangkat lunak. Oleh karena itu, penulisan menggunakan GetX akan bersifat *clean code*. *Business Logic Class* pada GetX disebut dengan *controller*. Pada Gambar 1 diatas menunjukkan bahwa *controller* sebagai perantara antara UI dan data *access layer*. Pada *class* ini *variable* dapat dibuat menjadi observable atau selalu mengikuti perubahan yang terjadi pada state *(reactive)* dengan cara menambahkan ".obs" di akhir *variable*. Cara untuk mengimplementasikan observable variables yaitu dengan menggunakan class Obx. Obx merupakan *lightweight implementation* yang akan menggunakan *anonymous function* yaitu mengembalikan *widget* dan dapat digunakan dengan cara "Obx (=> *MyWidget*)". *Controller* yang telah dibuat harus dilakukan injeksi manual dengan cara *Get.find*() [14].

Kelebihan GetX, antara lain, menyediakan *reactive programming*, pengimplementasian *reactive pattern* dengan *boilerplate code* yang sedikit, relatif mudah untuk dipelajari serta diimplementasikan, mendukung *reactive* dan *non-reactive state management*, terdapat *dependency injector, navigator,* dan memberikan pemisahan yang jelas antara *business logic* dan *user interface code* [14]. Sedangkan kekurangan GetX, antara lain, dalam pendekatan reactive GetX membutuhkan sumber daya yang cukup banyak. Sehingga, kesalahan dalam pengimplementasian GetX dapat menimbulkan masalah performa pada beberapa perangkat, dan tingkat testability yang rata – rata [12]. Berdasarkan kelebihannya, GetX cocok diimplementasikan oleh *developer* yang baru mengenal *flutter* dan belum menemukan *state management* yang cocok untuknya. Tetapi, tetap mendapatkan keuntungan dari *reactive programming* dengan cara yang sederhana. Sedangkan, berdasarkan kekurangannya GetX tidak cocok untuk digunakan pada proyek perangkat lunak yang kompleks [14].

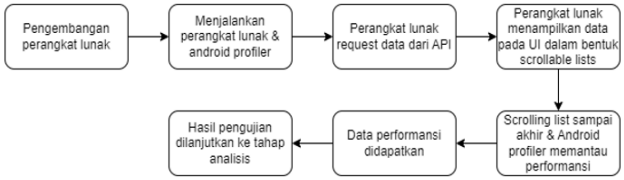
BLoC

Business Logic Component (BLoC) merupakan *state management library* yang bekerja dengan cara menerima *event* yang akan memicu perubahan *state* lalu dipancarkan dari BLoC. Sehingga BLoC dapat memisahkan antara *input* dan *output*. Setelah memberikan *event* ke input lalu BLoC akan menerima *state* dari *output*. *Output* tersebut dapat diamati oleh *widget UI* yang bereaksi terhadap perubahan *state*. BLoC sering dibandingkan dengan MVVM *pattern*, namun *ViewModel* pada MVVM digantikan dengan BLoC 2 BLoC berperan sebagai perantara antara UI dan data *access layer*. BLoC mulai bekerja ketika ada event yang memicu *business logic*. Contohnya ketika terdapat event yang meminta *repository* untuk meminta *request* API, lalu BLoC akan memproses data tersebut dan memancarkan *state* sesuai dengan data yang diterima [14].

Kunci utama dalam BLoC adalah stream yang akan melanjutkan alur data secara *asynchronous*. Data yang telah ditambahkan ke stream juga dapat di listen secara bersamaan. Gambar diatas 3 menjelaskan secara detail bagaimana cara kerja BLoC. Secara sederhana pada BLoC terdapat stream yang akan menyalurkan *event* apapun yang diterima lalu dikembalikan ke UI *layer*. *Stream* terdiri dari dua bagian, yang pertama sebagai penerima *event* berdasarkan interaksi *user* ataupun *event* yang didapatkan dari UI. Dan yang kedua digunakan oleh BLoC untuk *listen new state* yang bertujuan untuk memperbaharui UI ataupun menambahkan *state* baru berdasarkan



Gambar 4. Tahapan Penelitian



Gambar 5. Skenario Pengujian

event yang diterima [21]. Kelebihan dari BLoC, antara lain, memiliki pemisahan *class* yang lebih baik karena tidak hanya *business logic* yang dipisahkan secara menyeluruh, namun juga pemisahan antara *input* dan *output*, memberlakukan penggunaan arsitektur yang mirip seperti MVVM, menyediakan mekanisme *built-in dependency injection* untuk BLoC, memungkinkan *reusable-code* pada perangkat lunak yang sama ataupun untuk perangkat lunak yang berbeda, sangat baik dalam *scalability*, dan sangat baik dalam *testability* [14].

Sedangkan, kekurangan dari BLoC, antara lain, dalam pendekatan *reactive* BLoC menghabiskan lebih banyak sumber daya pada sistem, memiliki banyak *boilerplate code* dan cara penulisan serta proses pemahaman yang cukup kompleks [14]. BLoC dapat dianggap sebagai pendekatan tingkat lanjut, karena cukup sulit untuk diimplementasikan dan memiliki konsep yang cukup rumit. Namun, dengan tingkat kompleksitasnya yang tinggi dan banyaknya *boilerplate code*, penulisan code menjadi *testable, reusable, scalable* dan juga penggunaan BLoC cocok untuk *clean architecture*. Oleh karena itu, penggunaan yang tepat untuk *state management* ini adalah ketika mengembangkan perangkat lunak yang kompleks [14].

Android Profiler

Android Profiler pada IDE *Android Studio* merupakan *tools* yang menyediakan data *real-time* untuk memahami penggunaan *resource* CPU, *memory*, jaringan, dan baterai pada suatu perangkat lunak [19]. Ketiga *resource* ini akan dipantau oleh *Android Profiler* secara detail sehingga dapat diketahui cara aplikasi dalam berinteraksi dengan sistem.

Metodologi Penelitian

Berdasarkan Gambar 4 berikut tahapan yang akan dilakukan dalam penelitian ini.

Implementasi dan Pengembangan Perangkat Lunak

Tahap ini melakukan implementasi dan pengembangan perangkat lunak. Pengembangan perangkat lunak menggunakan *framework Flutter* dan bahasa pemrograman *Dart*. Perangkat lunak dikembangkan sebanyak dua buah dengan mengimplementasikan *state management* yang berbeda. Pada perangkat lunak pertama mengimplementasikan *BLoC state management* dan perangkat lunak kedua mengimplementasikan *GetX state management*. Kedua perangkat lunak ini akan dikembangkan dengan tampilan dan fungsionalitas yang sama. Hal ini bertujuan untuk meminimalisir perbedaan pada perangkat lunak agar mendapatkan hasil yang maksimal. Tampilan perangkat lunak dapat dilihat pada Lampiran 1 halaman 12. Perangkat lunak yang dikembangkan menerapkan API *(Application Programming Interface)*. API yang digunakan diambil dari laman *api.tvmaze.com.* API ini dipilih karena menyediakan kecepatan dalam REST API, open source, dan berupa data JSON *(JavaScript Object Notation)*. API tersebut memberikan 240 data *tv shows*. Perangkat lunak yang dikembangkan memiliki fitur *scrollable list* dan dibuat sederhana agar dapat berfokus pada *State Management* [14]. Variabel pengujian yang akan diukur adalah CPU *usage, memory usage,* dan *energy consumption*.

Membuat Skenario Pengujian

Pada tahap ini, dilakukan pembuatan skenario pengujian. Kedua perangkat lunak yang telah dikembangkan pada bagian 3.3 akan dilakukan pengujian untuk mendapatkan data performansinya menggunakan *Android Profiler*. Setiap perangkat lunak dilakukan pengujian untuk setiap kategori. Kategori dibedakan berdasarkan jumlah data yang digunakan yaitu 50, 100, 240, 2400, 2.400.000, dan 4.800.000. Jumlah data ini dipilih dengan tujuan untuk melihat performa *state management* untuk setiap tingkatan data. Berikut skenario pengujian perangkat lunak: 5

1. Perangkat lunak yang dikembangkan memiliki fitur *scrollable list* yang akan menampilkan list data dari API dan dilakukan duplikasi data agar sesuai dengan kategori yang telah ditetapkan sebelumnya. Berdasarkan hal tersebut, pengujian dilakukan dengan cara menjalankan perangkat lunak yang diinstal pada *device samsung* S9 dan diukur performansinya menggunakan *Android Profiler*. Dapat dilihat pada Lampiran 1 halaman 12.
2. Pada perangkat lunak yang dikembangkan terdapat fitur *loading* yang akan berakhir ketika semua data dari API telah berhasil di *request*, kemudian data tersebut akan ditampilkan dalam bentuk *list* pada perangkat lunak. Dapat dilihat pada Lampiran 1 halaman 12.
3. Proses pengujian dimulai ketika semua data telah ditampilkan. Lalu dilakukan *scrolling* hingga *list* yang ditampilkan mencapai batas akhirnya. Kemudian, *Android Profiler* akan memantau nilai performansi perangkat lunak tersebut. Dapat dilihat pada Lampiran 1 halaman 12.
4. Setelah menjalankan perangkat lunak dan *Android Profiler*, lalu akan didapatkan data performansi berupa CPU *usage, memory usage,* dan *energy consumption*. Dapat dilihat pada Lampiran 2 halaman 13.
5. Hasil pengujian akan dilanjutkan ke tahap analisis.

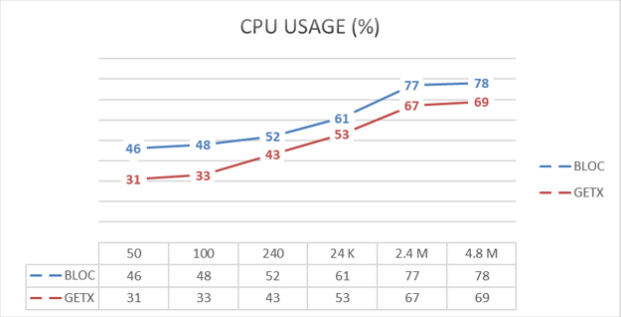
Menjalankan Pengujian

Pengujian dilakukan terhadap kedua perangkat lunak yang telah dikembangkan dengan mengimplementasikan BLoC dan GetX *state management*. Metode yang digunakan pada penelitian ini berasal dari penelitian yang dilakukan oleh Ilham Alamsyah [17] yaitu *performance testing*. Metode ini dipilih karena dapat mengukur *speed, effectiveness, network*, dan lainnya pada perangkat lunak. Pengujian dilakukan untuk mendapatkan nilai variabel performansi yang dibutuhkan dengan menerapkan skenario pengujian yang telah dijelaskan pada bagian 3.4. Kemudian data hasil pengujian dilanjutkan ke tahap analisis.

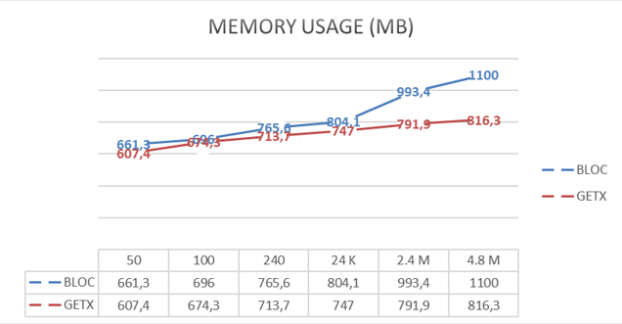
Hasil dan Pembahasan

CPU *Usage*

Berdasarkan hasil pengujian, dapat dilihat 6 bahwa ketika terjadi peningkatan jumlah data, maka CPU *usage* pun meningkat. Pada BLoC untuk kategori data berjumlah 50, 100 dan 240 memiliki CPU *usage* yang relatif stabil antara 46% - 52%. Lalu CPU *usage* mulai melonjak



Gambar 6. Grafik CPU *Usage*



Gambar 7. Grafik *Memory Usage*

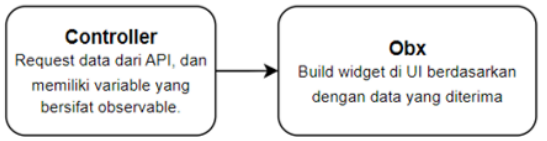
Table 1. *Energy Consumption*

	ENERGY CONSUMPTION					
	50	**100**	**240**	**24 K**	**2.4 M**	**4.8 M**
BLOC	Light	Light	Light	Light	Light	Light
GETX	Light	Light	Light	Light	Light	Light

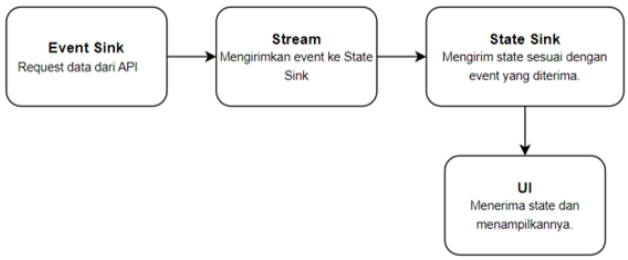
ketika jumlah data yang diterima antara 24.000 dan 2.400.000. Dan CPU *usage* mulai stabil antara 2.400.000 dan 4.800.000 yaitu 77% - 88%. Sedangkan GetX memiliki CPU *usage* yang lebih rendah dibandingkan BLoC pada setiap kategori. Berbeda dari BLoC, CPU *usage* minimum didapatkan pada kategori data 50 dan 100 yang relatif stabil antara 31-33%. CPU *usage* pada GetX mulai melonjak ketika menerima data berjumlah 240 hingga 2.400.000. Sama seperti BLoC, GetX juga memiliki CPU *usage* yang stabil ketika menerima data berjumlah 2.400.000 dan 4.800.000 dengan nilai CPU *usage* antara 67% - 69%.

Memory Usage

Berdasarkan hasil pengujian, dapat dilihat bahwa *memory usage* 7 meningkat seiring dengan jumlah data yang bertambah. Pada setiap kategori GetX memiliki *memory usage* yang lebih rendah jika dibandingkan dengan BLoC. Dan GetX memiliki *memory usage* yang relatif stabil di setiap kategori, namun memiliki lonjakan tertinggi pada data antara 50 dan 100 dengan peningkatan sebesar 66,9 MB. Sedangkan, pada BLoC *memory usage* yang relatif stabil hanya pada kategori data 50 hingga 24.000, lalu mulai melonjak pada kategori 2.400.000 hingga 4.800.000. Peningkatan *memory usage* tertinggi terdapat pada data antara 24.000 dan 2.400.000 yaitu sebesar 189,3 MB.



Gambar 8. Proses GetX



Gambar 9. Proses BLoC

Energy Consumption

Berdasarkan hasil pengujian, dapat dilihat bahwa jumlah data dan penggunaan *state management* yang berbeda tidak mempengaruhi nilai *energy consumption*1.

Analisis Hasil Pengujian

Pengujian menunjukkan bahwa jumlah data dapat mempengaruhi performa perangkat lunak berbasis *android* yang mengimplementasikan *state management* pada *Flutter*. Hal ini dikarenakan perangkat lunak yang dikembangkan menggunakan *Flutter* terdiri dari komponen UI yang disebut dengan *widget*, sehingga jumlah data yang diimplementasikan dapat mempengaruhi jumlah *widget*. Ketika *widget* dibangun dan berubah, *state* akan membaca informasi tersebut secara sinkronus. Disinilah peran *state management* yang membutuhkan *resource* untuk mengelola *state* tersebut. Pada GetX terdapat *keyword* ".obs" yang dapat membuat sebuah *variable* atau *class* menjadi *observable object*. Sehingga ketika terjadi perubahan pada *object*, *widget* yang menggunakan *object* tersebut akan langsung menerimanya dan menampilkan perubahan tersebut. Selain itu, GetX memungkinkan untuk menghapus fungsi ataupun *controller* yang sedang tidak digunakan, sehingga dapat mengurangi penggunaan *resources* yang tidak diperlukan. Pada perangkat lunak yang dikembangkan dalam pengujian, GetX *state management* berperan untuk mengolah dan menampilkan data menjadi *widget* yang akan ditampilkan pada UI. Proses GetX dalam mengelola *state* dapat dilihat pada gambar 8.

Pada gambar 8 dapat dilihat proses BLoC dalam mengelola *state*. Pada perangkat lunak yang dikembangkan, proses *request* API terletak pada *event sink*. *Event* tersebut akan dikirim ke *state sink* melalui *stream*, kemudian *state sink* akan mengelola *event* yang diterima lalu mengirimkan *state* sesuai dengan *event* tersebut dan mengirimkannya ke UI. Pada UI terdapat *class Stream Builder* yang berfungsi untuk membangun *widget* sesuai dengan *state* yang diterima berupa *list* data dari API.Kemudian di UI akan menggunakan class Obx() untuk membangun widget sesuai dengan data yang diterima dari observable variable. Proses BLoC dalam mengelola *state* sangatlah berbeda, jika pada GetX dapat membuat variable ataupun class menjadi observable object, sedangkan BLoC bekerja dengan menggunakan *streams*. *Streams* memungkinkan BLoC untuk menerima *events* dari UI lalu mengirimkan *state* ke UI dengan tujuan untuk memperbaharui perubahan berdasarkan interaksi perangkat. Pada gambar 9 dapat dilihat proses BLoC dalam mengelola *state*. Pada perangkat lunak yang

Table 2. Data Hasil Pengujian

Variabel	Hasil (avg)	
	GetX	**BLoC**
CPU Usage (%)	49.3333	60.3333
Memory Usage (mb)	725.1	836.7333
Energy Consumption	Light	Light

dikembangkan, proses *request* API terletak pada *event sink*. *Event* tersebut akan dikirim ke *state sink* melalui *stream*, kemudian *state sink* akan mengelola *event* yang diterima lalu mengirimkan state sesuai dengan *event* tersebut dan mengirimkannya ke UI. Pada UI terdapat *class Stream Builder* yang berfungsi untuk membangun *widget* sesuai dengan *state* yang diterima berupa *list* data dari API. Berdasarkan pengujian, nilai CPU dan *memory usage* meningkat seiring dengan jumlah data yang digunakan. Data hasil pengujian tersebut diambil rata – rata dari enam kategori yang dapat dilihat pada tabel 2.

GetX memiliki performa CPU dan *memory usage* yang lebih baik jika dibandingkan dengan BLoC. GetX memiliki rata - rata CPU *Usage* senilai 49.3333% dan BLoC senilai 60.3333%, dengan perbedaan senilai 11%. Sedangkan untuk *memory usage* GetX memiliki rata – rata senilai 725.1 mb dan BLoC senilai 636.7333 , dengan perbedaan senilai 111.6333 mb. Namun, memiliki rata – rata *energy consumption* yang sama. Perbedaan hasil performansi ini disebabkan oleh perbedaan cara kedua *state management* tersebut dalam mengelola *state* seperti yang telah dijelaskan sebelumnya. Sehingga, penerapan GetX *state management* pada perangakat lunak berbasis *android* akan lebih baik dalam segi performansi jika dibandingkan dengan BLoC *state management*.

Kesimpulan

Pada penelitian ini dilakukan pengujian performa terhadap GetX dan BLoC *state management* pada *Flutter* yang diimplementasikan pada perangkat lunak berbasis *android*. Pengujian skenario dilakukan untuk menentukan *state management* yang memiliki performa terbaik. Hasil pengujian menunjukkan bahwa penggunaan *state management* yang berbeda dan jumlah data yang berbeda dapat mempengaruhi cpu *usage* dan *memory usage* perangkat lunak, namun tidak mempengaruhi *energy usage*. GetX memiliki performa cpu *usage* dan *memory usage* yang lebih baik dibandingkan dengan BLoC dengan perbedaan CPU usage senilai 11% dan *memory usage* senilai 111.6333 mb. Sedangankan kedua kedua *state management* tersebut memiliki nilai *energy consumption* yang sama yaitu *light*. Diharapkan pada penelitian selanjutnya dilakukan pengujian performa pada *state management* yang berbeda contohnya MobX ataupun Redux sehingga dapat diketahui performa *state management* lain yang tersedia di *Flutter*. Dan diharapkan juga untuk melakukan pengujian dengan skenario yang berbeda sehingga dapat diketahui performa dari sudut pandang yang berbeda.

Daftar Pustaka

1. Google Play Store: number of apps 2023 — Statista — statista.com;. [Accessed 24-08-2023]. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.
2. State class - widgets library - Dart API — api.flutter.dev;. [Accessed 24-08-2023]. https://api.flutter.dev/flutter/widgets/State-class.html#:~:text=State%20is%20information%20that%20(1,such%20state%20changes%2C%20using%20State.
3. Flutter - Build apps for any screen — flutter.dev;. [Accessed 24-08-2023]. https://flutter.dev/.
4. Flutter - Build apps for any screen — flutter.dev;. [Accessed 24-08-2023]. https://flutter.dev/.
5. 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.
6. List of state management approaches — docs.flutter.dev;. [Accessed 24-08-2023]. https://docs.flutter.dev/data-and-backend/state-mgmt/options.
7. GitHub - jonataslaw/getx: Open screens/snackbars/dialogs/bottomSheets without context, manage states and inject dependencies easily with Get. — github.com;. [Accessed 24-08-2023]. https://github.com/jonataslaw/getx.
8. GitHub - jonataslaw/getx: Open screens/snackbars/dialogs/bottomSheets without context, manage states and inject dependencies easily with Get. — github.com;. [Accessed 24-08-2023]. https://github.com/jonataslaw/getx.
9. 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.
10. GitHub - rrousselGit/provider: InheritedWidgets, but simple — github.com;. [Accessed 24-08-2023]. https://github.com/rrousselGit/provider/.
11. Flutter - Build apps for any screen — flutter.dev;. [Accessed 24-08-2023]. https://flutter.dev/.
12. GitHub - felangel/bloc: A predictable state management library that helps implement the BLoC design pattern — github.com;. [Accessed 24-08-2023]. https://github.com/felangel/bloc.
13. get — Flutter Package — pub.dev;. [Accessed 24-08-2023]. https://pub.dev/packages/get.
14. Slepnev D. State management approaches in Flutter. 2020.
15. Garg S, Baliyan N. Comparative analysis of Android and iOS from security viewpoint. Computer Science Review. 2021 may;40:100372. Available from: https://doi.org/10.1016%2Fj.cosrev.2021.100372.
16. Li D, Hao S, Gui J, Halfond WG. An empirical study of the energy consumption of android applications. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE; 2014. p. 121-30.
17. Putri MA, Hadi HN, Ramdani F. Performance testing analysis on web application: Study case student admission web system. In: 2017 International Conference on Sustainable Information Engineering and Technology (SIET). IEEE; 2017. Available from: https://doi.org/10.1109%2Fsiet.2017.8304099.
18. Triandini E, Jayanatha S, Indrawan A, Putra GW, Iswara B. Metode Systematic Literature Review untuk Identifikasi Platform dan Metode Pengembangan Sistem Informasi di Indonesia. Indonesian Journal of Information Systems. 2019 feb;1(2):63. Available from: https://doi.org/10.24002%2Fijis.v1i2.1916.
19. Profile your app performance — Android Studio — Android Developers — developer.android.com;. [Accessed 24-08-2023]. https://developer.android.com/studio/profile.