ANALISIS PERFORMA STATE MANAGEMENT PROVIDER DAN GETX PADA APLIKASI FLUTTER

Ibrohim Husain, Purwantoro, Carudin
Program Studi Informatika S1, Fakultas Ilmu Komputer, Universitas Singaperbangsa Karawang
Jl. HS. Ronggo Waluyo, Telukjambe Timur, Karawang, Indonesia
1910631170193@student.unsika.ac.id

ABSTRAK

State Management adalah salah satu aspek penting dalam pengembangan aplikasi *mobile*, terutama pada aplikasi yang kompleks. Berdasarkan hasil pencarian pada *official package repository* untuk Flutter, Provider dan GetX adalah *library state management* Flutter yang paling populer dan disukai oleh para pengembang. Dari kedua *library state management* tersebut, perlu dipertimbangkan mana yang paling cocok untuk digunakan dalam pengembangan aplikasi. Penelitian ini bertujuan untuk menganalisis efektivitas penggunaan Provider dan GetX sebagai *library state management* dalam aplikasi Flutter dengan kriteria pengukuran berdasarkan parameter ukuran aplikasi dan performa (CPU, memori, *frame rate*). Penelitian dilakukan dengan membangun dua versi aplikasi ShowTime menggunakan *state management* Provider dan GetX secara terpisah. Setelah itu, dilakukan pengujian performa menggunakan *tools* Profiler pada Android Studio. Hasil pengujian menunjukkan perbedaan antara aplikasi ShowTime yang menggunakan *state management* Provider dan aplikasi ShowTime yang menggunakan *state management* GetX. Aplikasi dengan *state management* Provider memiliki ukuran aplikasi yang sama, penggunaan CPU yang lebih rendah, konsumsi memori yang lebih rendah, dan *frame rate* yang lebih tinggi dari aplikasi yang menggunakan *state management* GetX.

Kata kunci: *Analisis Performa, Flutter, Provider, GetX*

1. PENDAHULUAN

Flutter adalah *framework* pengembangan aplikasi yang dapat digunakan untuk berbagai platform. Dengan menggunakan Flutter, para pengembang hanya perlu membuat satu basis kode, dan aplikasi tersebut dapat dijalankan pada sistem operasi Android dan iOS dengan performa yang sama dengan aplikasi *native*. Aplikasi yang dikembangkan dengan Flutter dibangun dari blok komponen UI yang disebut *widget*. Flutter menyediakan *widget* yang sesuai dengan standar desain aplikasi yang sudah ada, sehingga memudahkan dalam membuat UI. Tampilan *widget* dapat berubah sesuai dengan konfigurasi dan *state* dari *widget* tersebut [1]. *State* adalah informasi yang dapat dibaca secara sinkron ketika *widget* dibangun dan dapat berubah selama umur *widget* [2]. *State* dari *widget* merupakan representasi dari data dinamis yang mempengaruhi tampilan di layar. Secara *default*, ketika *state* dari *widget* berubah, *widget* perlu dibangun ulang dengan memanggil metode setState. Dengan cara itu, Flutter akan membangun ulang semua *widget* di satu tampilan UI. Berbeda dengan setState yang membangun ulang semua *widget*, *library state management* memungkinkan aplikasi hanya membangun ulang *widget* yang mengalami perubahan *state* [3].

Berdasarkan hasil pencarian pada *official package repository* untuk Flutter seperti yang dapat dilihat pada Tabel 1, Provider dan GetX adalah *library state management* Flutter yang paling populer dan disukai oleh para pengembang [4].

Tabel 1 Hasil Pencarian State Management

No.	*State Management*	*Likes*	*Pub Points*	*Popularity*	*Link*
1	GetX	11952	130/140	100%	get | Flutter Package (pub.dev)
2	Provider	8360	140/140	100%	provider | Flutter Package (pub.dev)

Sumber: Search results for state management (pub.dev) (diakses pada tanggal 18 April 2023)

Provider bekerja dengan membungkus (*wrap*) *inherited widget* dan mengirimkan informasi perubahan *state* melalui komponennya. Provider akan memberitahu *observer* (*Consumer*) ketika terjadi perubahan data dalam kelas pengelolaan status (*ChangeNotifier*). Consumer dan *ChangeNotifier* dapat berkomunikasi melalui *ChangeNotifierProvider*[3]. Sementara itu, GetX merupakan *reactive state management* yang disederhanakan. GetX memproses perubahan *state* pada *controller* secara langsung tanpa menggunakan *Stream* [5]. Dengan menggunakan *library state management* Provider dan GetX, pembangunan ulang *widget* hanya terbatas pada *widget* yang memiliki perubahan *state*. Semakin sedikit *widget* yang dibangun ulang, semakin efisien sumber daya yang digunakan. Efisiensi kinerja dari produk perangkat lunak dapat dilihat berdasarkan beberapa karakteristik, salah satunya adalah penggunaan sumber daya.

Sumber daya pada perangkat mobile adalah CPU, memori, baterai, dan jaringan [3].

Dalam mengembangkan aplikasi Flutter, pengembang harus memperhatikan bagaimana aplikasi tersebut akan mengelola dan menyimpan data dan *state*. Oleh karena itu, penting bagi setiap pengembang untuk memahami cara menggunakan dan memilih *state management* yang tepat untuk aplikasi yang akan dikembangkan. *State Management* adalah salah satu aspek penting dalam pengembangan aplikasi mobile, terutama pada aplikasi yang kompleks. Dari kedua *library state management* yang paling disukai dan populer untuk Flutter, perlu dipertimbangkan mana yang paling cocok untuk digunakan dalam pengembangan aplikasi. Setiap *state management* memiliki kelebihan dan kekurangan. Dalam penelitian ini, penulis akan mengevaluasi efektivitas penggunaan Provider dan GetX sebagai *library state management* dalam aplikasi Flutter dengan kriteria pengukuran berdasarkan parameter ukuran aplikasi dan performa (CPU, memori, *frame rate)*. Hasil penelitian ini diharapkan dapat membantu para pengembang aplikasi Flutter dalam memilih *state management* yang paling tepat untuk proyek mereka, sehingga dapat meningkatkan kualitas dan performa aplikasi yang dihasilkan.

2. TINJAUAN PUSTAKA

2.1. Flutter

Flutter adalah *framework open source* yang dikembangkan oleh Google dan merupakan *toolkit* UI portabel untuk membangun aplikasi yang indah dan dikompilasi secara *native* untuk perangkat *mobile*, web, desktop, dari satu basis kode. Flutter pertama kali dirilis pada tahun 2017, namun untuk versi stabilnya diluncurkan pada Desember 2018. Sejak peluncurannya, lebih dari 400.000 aplikasi telah dibuat menggunakan Flutter dan digunakan oleh ratusan juta perangkat [6]. Flutter menyediakan fitur *hot-reload* yang merupakan salah satu fitur unggulan dalam mendorong siklus pengembangan dalam pembuatan UI, penambahan fitur, dan perbaikan *bugs*. Cara kerja dari *hot-reload* adalah dengan memasukkan berkas kode yang telah diperbarui ke dalam Dart Virtual Machine (VM) yang sedang berjalan. Setelah VM memperbarui *class* dengan versi yang baru. Flutter *framework* secara otomatis akan melakukan *build* ulang seluruh widget, dan memungkinkan untuk menampilkan perubahan secara langsung [7].

2.2. Dart

Dart adalah bahasa pemrograman yang *open source* dan *general purpose*. Dart dikembangkan oleh Google dan ditujukan untuk membuat aplikasi lintas platform seperti mobile, desktop, dan web. Dart awalnya dikenalkan pada konferensi GOTO pada tahun 2011. Project ini didirikan oleh Lars Bak dan Kasper Lund dari Google, sampai akhirnya versi Dart 1.0 dirilis pada 14 November 2013. Bulan Agustus 2018, Dart 2.0 dirilis dengan perubahan bahasa seperti perubahan *type system* [8].

2.3. Widget

Penggunaan *widget* pada Flutter awalnya terinspirasi dari *framework* lain yaitu React. Ide utamanya adalah bagaimana membuat UI dari *widget*. *Widget* menggambarkan seperti apa tampilan berdasarkan konfigurasi dan *state*nya saat ini. Ketika *state widget* berubah, maka *widget* akan dibangun ulang sesuai dengan konfigurasi yang baru [1]. *Widget* tidak hanya mengontrol dan mempengaruhi perilaku tampilan, tetapi juga merespons aksi yang dilakukan pengguna. Oleh karena itu sebuah *widget* haruslah memiliki performa yang cepat, termasuk pada proses *rendering* dan *animating* [9].

2.4. State management

State adalah informasi yang dapat dibaca secara sinkron ketika *widget* dibangun dan dapat berubah selama umur *widget* [2]. *State management* adalah teknik strukturisasi, pengiriman, dan sinkronisasi *state* aplikasi melalui komponen-komponen dari aplikasi tersebut. Dalam situasi ketika jumlah *state* pada aplikasi bertambah, biasanya akan semakin sulit untuk mempertahankan kerapiannya. Oleh karena itu, solusi *state management* lebih banyak ditujukan untuk aplikasi yang kompleks daripada yang kecil. Untuk mengembangkan aplikasi tanpa kehilangan kendali atas kompleksitasnya, diperlukan pendekatan standar satu atau lebih untuk berinteraksi dengan *state*, mutasi, dan sinkronisasinya. Pendekatan-pendekatan ini dapat bervariasi dari pola desain, alat, paket/*library*, atau hanya panduan [10]

2.5. Provider

Provider merupakan sebuah pembungkus (*wrapper*) dari *InheritedWidget* yang membuat penggunaannya lebih mudah dan *reusable* [11]. Provider dibuat oleh Remi Rousselet dan terdiri dari sekumpulan class yang dirancang untuk menyederhanakan pengelolaan state pada Flutter. Metode pengelolaan *state* dengan Provider adalah metode yang direkomendasikan oleh tim Flutter. Tim Flutter mengatakan bahwa metode ini adalah yang paling mudah dipahami dan cukup *powerful*. Provider, sama seperti InheritedWidget, pada dasarnya adalah mekanisme *dependency injection*. Provider dijanjikan akan memberikan *boilerplate* yang jauh lebih sedikit daripada membuat kelas baru setiap saat dan skalabilitas yang lebih baik [12].

2.6. GetX

GetX adalah solusi yang ekstra ringan dan andal bagi aplikasi Flutter. GetX merupakan sebuah kombinasi antara *state management* dengan performa tinggi, *dependency injection* yang cerdas, dan *route management* yang cepat dan praktis [5]. Prinsip dasar GetX terdiri dari tiga elemen, yaitu produktivitas, performa, dan organisasi. Fokus utama GetX adalah

pada performa dan konsumsi sumber daya yang minimal. Dalam hal ini, GetX memperbaiki produktivitas pengembangan dengan sintaks yang mudah dan menyenangkan, serta memberikan performa maksimal pada aplikasi. Selain itu, GetX memungkinkan pemisahan total antara View, *presentation logic*, *business logic*, *dependency injection*, dan navigasi, sehingga memungkinkan pengorganisasian yang lebih baik. Ekosistem GetX sangat luas dan didukung oleh komunitas yang besar, serta akan terus diperbarui selama Flutter masih relevan [5].

2.7. Rapid Application Development (RAD)

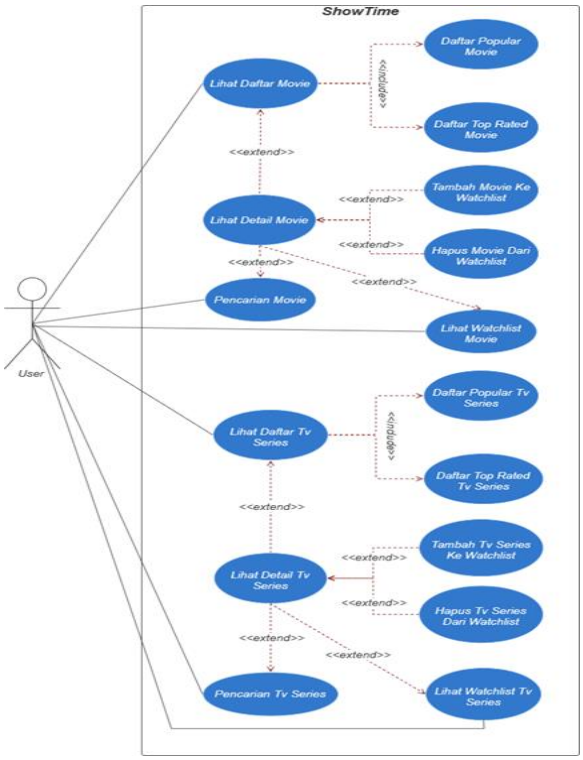
Rapid Application Development adalah sebuah metode yang menggabungkan beberapa pendekatan terstruktur. Metode ini menggunakan pendekatan iteratif dalam pengembangan sistem, dimana model sistem dibangun pada tahap awal pengembangan untuk mengidentifikasi kebutuhan pengguna. Metode RAD fokus pada pemodelan bisnis, pemodelan data, pemodelan proses, pembuatan aplikasi, dan pengujian [13]. Metode RAD telah menjadi pilihan populer dalam jurnal penelitian karena kemampuannya dalam membangun sistem dan prototipe aplikasi dengan cepat dan efisien, sehingga tujuan utamanya dapat dicapai dengan cepat. Karena waktu pengerjaan yang singkat dalam metode RAD, pengembang sering melakukan komunikasi yang intensif dengan klien untuk menganalisis kebutuhan yang dapat disesuaikan dengan situasi atau kondisi yang dapat berubah-ubah. [14].

Ada 4 tahapan yang harus dilalui ketika menggunakan metode RAD, yaitu perencanaan kebutuhan, desain sistem, proses pengembangan dan implementasi atau penyesuaian produk.

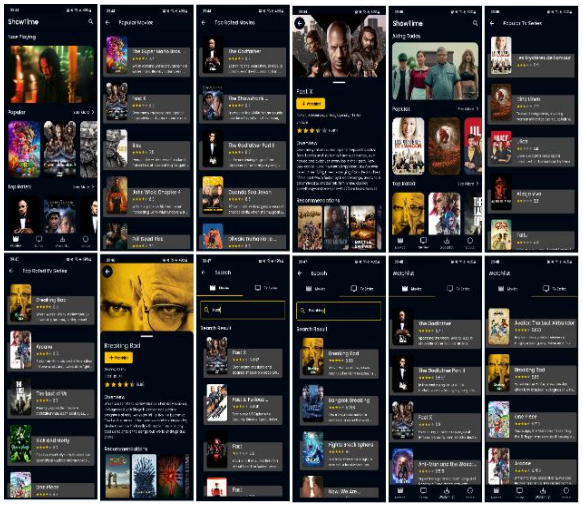
3. METODE PENELITIAN

Penelitian ini memfokuskan pada analisis efektivitas penggunaan Provider dan GetX sebagai library state management dalam aplikasi Flutter. Untuk mengetahui efektivitas penggunaan state management tersebut pada aplikasi Flutter, dilakukan pengujian aplikasi menggunakan parameter seperti ukuran aplikasi, dan performa (penggunaan CPU, konsumsi memori, dan frame rate). Studi kasus aplikasi yang digunakan pada penelitian ini adalah prototipe aplikasi yang fungsinya menyediakan informasi seperti judul, genre, poster hingga sinopsis dari berbagai film dan tv series. Pengembangan aplikasi prototipe ini menggunakan metode Rapid Application Development (RAD) dengan empat tahapan, yaitu Perencanaan Kebutuhan, Desain Sistem, Proses Pengembangan Aplikasi, dan Implementasi dan Finalisasi Produk.

Gambar 1. Diagram Rancangan Pengembangan Aplikasi



Gambar 2. Use Case Diagram Aplikasi ShowTime

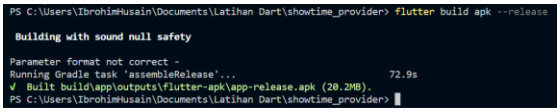


Gambar 3. Tampilan Aplikasi ShowTime

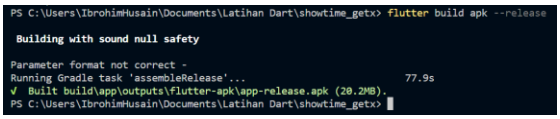
Setelah kedua aplikasi selesai dibuat, selanjutnya akan dilakukan pengujian untuk mengetahui efektivitas penggunaan state management Provider dan GetX pada aplikasi Flutter dengan memperhatikan parameter pengujian seperti ukuran aplikasi, dan performa (penggunaan CPU, konsumsi memori, dan frame rate).

4. HASIL DAN PEMBAHASAN

Pengujian ukuran aplikasi dilakukan dengan membangun aplikasi dalam mode release. Hasil pengujian ukuran aplikasi menunjukkan bahwa aplikasi Flutter yang mengimplementasikan penggunaan state management Provider dan GetX tidak memiliki perbedaan ukuran aplikasi yang signifikan, dengan hasil build aplikasi masing-masing menunjukkan ukuran 20.2 MB.



Gambar 4. Hasil Build APK dengan state management Provider



Gambar 5. Hasil Build APK dengan state management GetX

Sedangkan pengujian performa dilakukan dengan mengambil nilai puncak penggunaan CPU, memori, dan frame rate ketika menggunakan aplikasi, adapun skenario pengujian aplikasi yang akan digunakan terdapat pada Tabel 2.

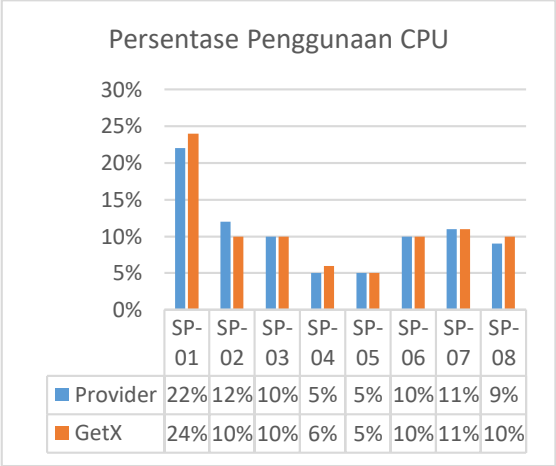
Tabel 2. Skenario Pengujian Aplikasi

No.	Skenario Pengujian
SP-01	Klik ikon aplikasi untuk membuka aplikasi dan menampilkan halaman movie
SP-02	Menggulirkan daftar movie
SP-03	Buka detail movie
SP-04	Kembali ke daftar movie
SP-05	Klik ikon pencarian untuk membuka halaman pencarian
SP-06	Klik kolom pencarian movie dan masukkan judul “Fast”
SP-07	Buka detail movie hasil pencarian
SP-08	Kembali ke menu pencarian

Pengujian ini menggunakan tools Profiler bawaan Android Studio dan terbatas pada platform Android, serta untuk menyimulasikan penggunaan nyata dari aplikasi, kedua aplikasi dijalankan pada perangkat fisik Samsung Galaxy A52s.

4.1. Penggunaan CPU

Pengujian penggunaan CPU pada aplikasi ShowTime dilakukan dengan menggunakan *tools* Profiler pada Android Studio. Proses pengujian dilakukan dengan menjalankan skenario pengujian secara berurutan. Hasil penggunaan CPU untuk masing-masing skenario dapat dilihat pada Gambar 6 Dari gambar tersebut, dapat dilihat bahwa aplikasi yang menggunakan *state management* Provider memiliki persentase penggunaan CPU antara 5% hingga 22%. Sementara itu, aplikasi yang menggunakan *state management* GetX memiliki persentase penggunaan CPU antara 5% hingga 24%. Data tersebut mencerminkan variasi dalam penggunaan CPU pada setiap skenario pengujian.



Gambar 6 Presentase Penggunaan CPU

Untuk mempermudah dalam mengevaluasi hasil pengujian CPU, data persentase penggunaan CPU yang tercantum dalam Gambar 6 akan diolah dan disajikan dalam Tabel 3 yang mencakup nilai rata-rata, tertinggi, dan terendah.

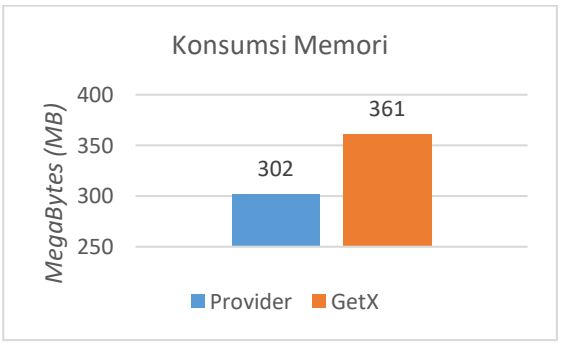
Tabel 3 Hasil pengujian CPU

Nilai	Provider	GetX
AVG	10.5%	10.75%
MAX	22%	24%
MIN	5%	5%

Data pada Tabel 3 menunjukkan adanya perbedaan dalam rata-rata (AVG) dan penggunaan CPU tertinggi (MAX) antara kedua aplikasi, *state management* Provider memiliki rata-rata penggunaan CPU sebesar 10.5% dan penggunaan CPU tertinggi sebesar 22%, sedangkan *state management* GetX memiliki rata-rata penggunaan CPU sebesar 10.75% dan penggunaan CPU tertinggi sebesar 24%. Dengan demikian, dapat disimpulkan bahwa secara keseluruhan, aplikasi ShowTime yang menggunakan *state management* Provider memiliki penggunaan CPU yang lebih rendah dari aplikasi ShowTime yang menggunakan *state management* GetX.

4.2. Konsumsi Memori

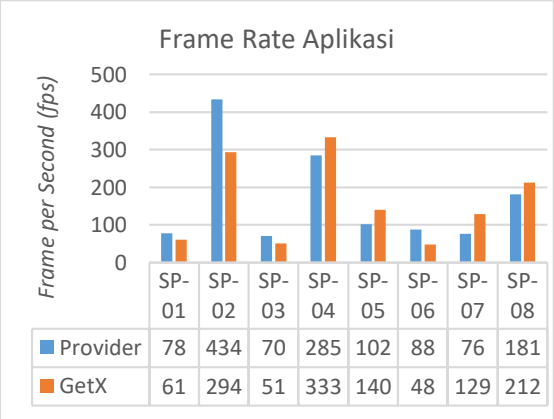
Pengujian konsumsi memori pada aplikasi ShowTime menggunakan *tools* Profiler dari Android Studio, pengujian dilakukan dengan menjalankan skenario 1 – 8 selama kurang lebih 1 menit. Hasilnya adalah aplikasi dengan *state management* Provider mengonsumsi memori sebesar 302.4 MB dan aplikasi dengan *state management* GetX mengonsumsi memori sebesar 361.2 MB. Hasil konsumsi memori untuk masing-masing *state management* dapat dilihat pada Gambar 7.



Gambar 7 Konsumsi Memori

4.3. Frame Rate

Pengujian untuk mengukur *frame rate* pada aplikasi menggunakan *tools performance overlay* yang telah disediakan oleh *framework* Flutter, sama seperti tahap sebelumnya pengujian dilakukan dengan mengikuti skenario secara berurutan. Berbeda dengan pengujian penggunaan CPU dan konsumsi memori dimana nilai yang lebih rendah adalah yang lebih baik, pengujian pada *frame rate* dinilai lebih baik apabila mendekati atau sama dengan nilai *refresh rate* yang didukung oleh perangkat tersebut, dalam studi kasus ini perangkat yang digunakan adalah ponsel Samsung Galaxy A52s yang sudah mendukung *refresh rate* tinggi sampai dengan 120 Hz. Semakin tinggi nilai *frame rate* aplikasi atau mendekati *refresh rate* perangkat maka tampilan yang dihasilkan lebih lancar (*smooth*). Hasil *frame rate* aplikasi untuk masing-masing skenario dapat dilihat pada Gambar 8.



	SP-01	SP-02	SP-03	SP-04	SP-05	SP-06	SP-07	SP-08
Provider	78	434	70	285	102	88	76	181
GetX	61	294	51	333	140	48	129	212

Gambar 8. Frame Rate Aplikasi

Hasil pengujian menunjukkan adanya perbedaan yang cukup signifikan dalam *frame rate* yang dihasilkan antara kedua aplikasi. Rata-rata *frame rate* aplikasi ShowTime yang menggunakan *state management* Provider adalah 164 *fps*, sementara aplikasi ShowTime yang menggunakan *state management* GetX memiliki rata-rata *frame rate* sebesar 158 *fps*. Dengan demikian, dapat disimpulkan bahwa aplikasi ShowTime yang menggunakan *state management* Provider memiliki kinerja *frame rate* yang lebih baik daripada aplikasi ShowTime yang menggunakan *state management* GetX.

5. KESIMPULAN DAN SARAN

Berdasarkan hasil penelitian yang telah dilakukan dapat disimpulkan bahwa aplikasi ShowTime yang menggunakan *state management* Provider memiliki ukuran aplikasi sebesar 20.2 MB, rata-rata penggunaan CPU sebesar 10.5%, konsumsi memori sebesar 302.4 MB, dan rata-rata *frame rate* sebesar 164 *fps*. Sementara itu, aplikasi ShowTime yang menggunakan *state management* GetX memiliki ukuran aplikasi sebesar 20.2 MB, rata-rata penggunaan CPU sebesar 10.75%, konsumsi memori sebesar 361.2 MB, dan rata-rata *frame rate* sebesar 158 *fps*. Berdasarkan hasil pengujian, aplikasi dengan state management Provider memiliki ukuran aplikasi yang sama, penggunaan CPU yang lebih kecil, konsumsi memori yang lebih rendah, dan frame rate yang lebih tinggi dari aplikasi yang menggunakan state management GetX.

Untuk penelitian selanjutnya disarankan tidak hanya menganalisis performa antar dua state management tetapi dibandingkan juga antar beberapa state management yang berbeda, serta melakukan pengujian pada platform iOS untuk mengetahui bagaimana aplikasi Flutter yang dibangun menggunakan state management Provider atau GetX berjalan pada platform tersebut, lalu disarankan untuk menggunakan metode pengujian yang berbeda untuk membandingkan kedua state management tersebut, sebagai contoh pengujian untuk menampilkan visual yang lebih dinamis atau melakukan pengujian pada aplikasi yang lebih kompleks dan membutuhkan daya komputasi yang lebih berat.

DAFTAR PUSTAKA

[1] "Introduction to widgets | Flutter." https://docs.flutter.dev/development/ui/widgets-intro (accessed Mar. 10, 2023).

[2] "State class - widgets library - Dart API." https://api.flutter.dev/flutter/widgets/State-class.html (accessed Mar. 10, 2023).

[3] R. Rama Prayoga, G. Munawar, R. Jumiyani, and A. Syalsabila, "Performance Analysis of BLoC and Provider State Management Library on Flutter | Jurnal Mantik," *Mantik*, vol. 5, no. 3, pp. 1591–1597, Nov. 2021, Accessed: Mar. 10, 2023. [Online]. Available: https://www.iocscience.org/ejournal/index.php/mantik/article/view/1539

[4] "Search results for state management." https://pub.dev/packages?q=state+management&sort=like (accessed Mar. 10, 2023).

[5] "get | Flutter Package." https://pub.dev/packages/get#about-get (accessed Mar. 11, 2023).

[6] "FAQ | Flutter." https://docs.flutter.dev/resources/faq (accessed Mar. 16, 2023).

[7] "Hot reload | Flutter." https://docs.flutter.dev/development/tools/hot-reload (accessed Mar. 16, 2023).

[8] "Apa itu Dart | Memulai Pemrograman Dengan Dart | Dicoding Indonesia." https://www.dicoding.com/academies/191/tutorials/7450 (accessed Mar. 16, 2023).

[9] F. J. Fayzullaev, X. Supervisor, and T. Mynttinen, "Native-like Cross-Platform Mobile Development : Multi-OS Engine & Kotlin Native vs Flutter," 2018, Accessed: Mar. 16, 2023. [Online]. Available: http://www.theseus.fi/handle/10024/148975

[10] L. VENTURA, "Analysis of Redux, MobX and BLoC and how they solve the state management problem," 2022. Accessed: Mar. 10, 2023. [Online]. Available: http://hdl.handle.net/10589/190202

[11] "provider | Flutter Package." https://pub.dev/packages/provider (accessed Apr. 17, 2023).

[12] D. Slepnev, "State management approaches in Flutter," South-Eastern Finland University of Applied Sciences, 2020. Accessed: Mar. 10, 2023. [Online]. Available: http://www.theseus.fi/handle/10024/355086

[13] D. Karim and H. B. Santoso, "Perancangan Dan Usability Evaluation Prototipe Informasi Akademik Menggunakan Metode Rapid Application Development," *Jurnal Ilmiah ILKOMINFO - Ilmu Komputer & Informatika*, vol. 2, no. 2, pp. 68–79, Jul. 2019, doi: 10.47324/ILKOMINFO.V2I2.33.

[14] J. Martin, A. R. Tanaamah, and K. S. Wacana, "Perancangan Dan Implementasi Sistem Informasi Penjualan Berbasis Desktop Website Menggunakan Framework Bootstrap Dengan Metode Rapid Application Development, Studi Kasus Toko Peralatan Bayi 'Eeng Baby Shop,'" *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 1, pp. 57–68, Mar. 2018, doi: 10.25126/JTIIK.201851547.