1. Pendahuluan

Latar Belakang

Aplikasi *mobile* merupakan suatu program atau software yang dijalankan pada perangkat *mobile*. Awalnya aplikasi *mobile* ditawarkan untuk tujuan informasi dan produktivitas seperti email, kalender, dan kontak. Namun seiring berkembangnya zaman penggunaan aplikasi *mobile* semakin meluas ke area lain seperti *mobile games*, GPS, *order-tracking*, *banking*, media sosial, pembelian tiket, dan banyak aplikasi *mobile* lainnya yang tersedia [1]. Dengan banyaknya aplikasi yang tersedia dapat memudahkan setiap orang untuk melakukan pekerjaan seperti cek email, atau menghubungi partner bisnis kapanpun sehingga produktivitas semakin meningkat. Tetapi tidak semua aplikasi *mobile* dapat berjalan sesuai dengan harapan pengguna. Permasalahan yang muncul ketika aplikasi *mobile* tidak berjalan dengan baik biasanya pengguna akan menghapus aplikasi *mobile* tersebut dan juga cenderung akan memberikan ulasan negatif terhadap aplikasi *mobile* tersebut.

Menurut survei tentang ulasan negatif yang dilakukan oleh Apigee [2], penyebab pengguna aplikasi *mobile* memberikan ulasan negatif diantaranya karena masalah *freezes*, *crashes*, *slow responsive*, penggunaan baterai yang berlebihan, dan terlalu banyak iklan. Ketika pengguna aplikasi *mobile* mengalami hal tersebut, "44% responden secara lisan mengatakan bahwa mereka (pengguna) akan segera menghapus aplikasi *mobile* jika aplikasi tersebut tidak berfungsi seperti yang diharapkan". Survei tersebut juga mencatat bahwa hampir setiap responden (98%) menganggap bahwa *performance* itu merupakan prioritas utamanya. Kemudian menurut survei The App Attention Index 2019 oleh tim AppDynamics mengatakan bahwa masalah *performance* terjadi di semua jenis layanan digital setiap hari seperti *Social Media*, *Entertainment*, *Productivity*, *Retail*, *Finance*, dan lainnya. Survei ini mencatat bahwa sebanyak 55% pengguna merasa frustasi ketika dihadapkan dengan masalah pada *performance* [22]. Aplikasi dengan efisiensi kinerja yang rendah dapat menyebabkan masalah saat mengakses aplikasi seperti waktu muat yang lambat, Crashes, atau Freezes [2]. Dengan kemajuan teknologi setiap harinya di industri *mobile*, sangat diperlukan untuk fokus pada kualitas perangkat lunak aplikasi *mobile* terutama dalam hal *performance* [3]. Oleh karena itu, pentingnya melakukan optimasi dalam hal *performance* pada pengembangan aplikasi *mobile*.

Berdasarkan penelitian yang dilakukan oleh Shahbudin [4] dengan menerapkan desain arsitektur pada aplikasi *mobile* yang dibuat dapat meningkatkan efisiensi dari aplikasi tersebut. Rendahnya efisiensi aplikasi dapat menyebabkan masalah ketika mengakses aplikasi tersebut seperti *slow load time*, *crash*, atau *froze* [4]. Efisiensi dalam aplikasi *mobile* dapat dilihat dari beberapa karakteristik seperti *time efficiency* (time consumption dan network time), *resource efficiency* (penggunaan memori, baterai, dan CPU) [4].

Arsitektur yang digunakan pada penelitian ini adalah Model-View-Presenter (MVP). Model MVP diperkenalkan pertama kali oleh Taligent pada bahasa pemrograman C++ dan Java [8]. Konsep dasar MVP berasal dari MVC, akan tetapi perbedaannya berada pada Presenter. Komponen utama dari pola MVP adalah Presenter yang secara langsung mengakses View dan Model lalu mengkoordinasikan interaksi ketiga komponen tersebut [7]. MVP juga digunakan karena pemakaian memori yang lebih sedikit dibandingkan dengan MVC dan MVVM yang mengindikasikan bahwa *performance* yang dihasilkan lebih baik [5]. Selain itu MVP juga memiliki keunggulan coupling level yang rendah sehingga memudahkan untuk melakukan perubahan [5].

Terdapat banyak *framework* dalam pengembangan aplikasi *mobile*, salah satunya ialah Flutter. Flutter merupakan *framework* yang dikembangkan oleh Google yang mendukung teknologi *cross-platform*. Flutter memiliki tujuan untuk memungkinkan developer membuat aplikasi berkinerja tinggi untuk berbagai platform [10] sehingga sangat cocok digunakan untuk pengembangan aplikasi yang memprioritaskan aspek *performance*. Aplikasi yang dikembangkan dengan Flutter dibangun dari komponen-komponen User Interface yang disebut dengan widget. Tampilan widget dapat berubah sesuai dengan konfigurasi dan state [11]. *State management* diperlukan agar widget yang memiliki perubahan state saja yang akan di-rebuild dan semakin sedikit widget yang dibangun, maka akan semakin efisien juga penggunaan resources nya [12].

Terdapat banyak *state management* yang dapat digunakan diantaranya *state management* yang populer dan banyak disukai saat ini ialah Provider, GetX, dan BLoC [13]. Ketiga *state management* tersebut masing-masing memiliki kelebihan dan kekurangan tergantung pemakaiannya. BLoC memiliki pattern tersendiri yang dimana BLoC pattern mirip dengan Model-View-ViewModel (MVVM) namun ViewModel diganti menjadi BLoC [13]. Oleh karena itu *state management* BLoC tidak cocok jika ingin digunakan bersamaan dengan MVP dikarenakan BLoC sendiri sudah memiliki pattern. Selain itu *state management* Provider dan GetX keduanya memiliki beberapa keunggulan yang sama seperti mudah digunakan dan dapat meminimalisir kode [13]. Namun jika melihat dari aspek *performance*, GetX lebih baik dalam hal *performance* karena tim GetX sendiri mengklaim bahwa GetX memiliki prinsip salah satunya berfokus terhadap *performance* dan penggunaan resources yang minim [14]. Oleh karena itu, *state management* yang dipilih ialah GetX, karena penelitian ini berfokus terhadap optimasi dari sisi *performance*. Dengan penerapan pola arsitektur MVP dan *state management* GetX yang keduanya memiliki keunggulan dalam sisi *performance*, memungkinkan aplikasi *mobile* yang dibuat memiliki kinerja yang tinggi.

Topik dan Batasannya

Berdasarkan latar belakang, salah satu masalah yang menyebabkan aplikasi *mobile* mendapatkan banyak ulasan negatif ialah karena aplikasi *mobile* tersebut tidak dapat berfungsi seperti yang diharapkan, terutama dalam hal *performance* aplikasi [2], oleh sebab itu diperlukannya optimasi *performance*. Selain itu adapun rumusan masalah dalam penelitian ini diantaranya:

- Bagaimana menerapkan pola arsitektur MVP bersamaan dengan *state management* GetX pada aplikasi *mobile* berbasis Flutter?
- Bagaimana mengukur *performance* pada aplikasi *mobile*?
- Bagaimana mengevaluasi *performance* pada aplikasi *mobile* berbasis Flutter yang menerapkan kombinasi MVP-GetX maupun aplikasi *mobile* berbasis Flutter tanpa penerapan pola arsitektur dan *state management*?

Untuk batasannya, pola arsitektur MVP dan *state management* GetX yang dirancang hanya untuk aplikasi *mobile* berbasis Flutter saja. Aspek *performance* aplikasi *mobile* yang diukur pada penelitian ini ialah *memory* dan CPU *usage* sebagaimana seperti yang dilakukan pada penelitian terkait [9].

Tujuan

Penelitian ini memiliki tujuan untuk menerapkan kombinasi pola arsitektur MVP dan *state management* GetX (MVP-GetX) dan mengevaluasi efisiensi kombinasi MVP-GetX dalam mengoptimalkan kinerja aplikasi, khususnya berfokus pada penggunaan CPU dan penggunaan memori di berbagai ukuran dataset dan skenario pengujian.

Organisasi Tulisan

Tulisan ini dibagi menjadi beberapa bagian. Sebagai awalan, diberikan pemaparan terkait penerapan pola arsitektur MVP dan *state management* GetX pada aplikasi *mobile*. Kemudian dijelaskan alur penelitian yang meliputi studi literatur, pengembangan dan pengujian aplikasi, evaluasi, dan yang terakhir ialah kesimpulan dari hasil penelitian ini.