Perbandingan Penggunaan *Reactive Programming* dan *Object-Oriented Programming* pada Kinerja Aplikasi Sederhana Berbasis Android

Dewanto Joyo P[1], Naufal Rajabi[2], Riyanzani Anggara P[3], Ani Rahmani[4]

[1234]*Jurusan Teknik Informatika - Politeknik Negeri Bandung Bandung 40012*
E-mail : {dewanto.joyo.tif418, naufal.rajabi.tif418, riyanzani.anggara.tif418, anirahma}@polban.ac.id

ABSTRAK

Sistem antarmuka reaktif pada saat ini dibutuhkan untuk menjawab isu melimpahnya data dan kejadian yang terjadi pada suatu antarmuka pengguna. Hal ini dimaksudkan agar pengguna nyaman saat menggunakan aplikasi. Suatu metode bernama *Reactive Programming* (RP) hadir sebagai metode baru dalam mengembangkan *user interface* yang memiliki aliran data dan kejadian/*event* yang kompleks. RP bekerja dengan merepresentasikan hal dunia nyata menjadi suatu aliran kejadian/*event* yang akan dikelola secara *asynchronous*. Di sisi lain, Object Oriented (OO) memiliki pendekatan dengan merepresentasikan wujud dunia nyata ke dalam objek objek sebagai entitas utama suatu program. Tulisan ini menjelaskan hasil studi perbandingan kinerja aplikasi sederhana berbasis Android yang dikembangkan dengan paradigma berbeda yaitu *reactive* dan *pure* OO. Perbandingan kinerja aplikasi diukur dari konsumsi sumber daya aplikasi identik melalui *profiler* pada Android Studio. Dari tiga parameter pengukuran (penggunaan memori, CPU, dan energi), aplikasi dengan paradigma RP cenderung memiliki konsumsi sumber daya 1,8-8,1% lebih tinggi dibandingkan aplikasi dengan paradigma *pure* OO. Hasil ini menunjukkan bahwa pemrograman dengan paradigma *pure* Object Oriented memiliki kinerja yang lebih ramah terhadap penggunaan sumber daya/*resources* pada kasus implementasi aplikasi Android sederhana.

Kata Kunci
Reactive programming, paradigma, object-oriented, android, asynchronous

1. PENDAHULUAN

Reactive programming (RP) merupakan sebuah trend dalam pemrograman pada pengembangan aplikasi. RP diusulkan sebagai solusi untuk menyederhanakan pengembangan sistem reaktif [1] [2].

Pada umumnya aplikasi Android dibuat dengan bahasa berbasis *Object Oriented* seperti Java dan Kotlin. *Object Oriented Programming* (OOP) adalah paradigma pemrograman yang merepre-sentasikan dunia nyata ke dalam objek di mana objek adalah entitas fundamental suatu program [3]. Pada paradigma OOP semua fungsi, data, dan berbagai pengolahan data akan dimasukkan ke dalam kelas-kelas dan *object-object*. Setiap *object* memiliki tugas dan sifat yang berbeda. *Object-object* tersebut dapat bekerja sendiri ataupun saling berkaitan satu sama lain.

Ada beberapa konsep dasar yang harus dipahami dalam OOP yaitu, *object*, *attribute*, *method*, *class*, *encapsulation*, *inheritance*, dan *polymorphism* [3].

Pada implementasi level bahasa pemrograman, *reactive programing* diaktivasi dengan suatu *reactive extension* yaitu sebuah library untuk melakukan proses *asynchronous* berdasarkan kejadian/*event* menggunakan *observable sequences* [4]. Setiap aplikasi memiliki antarmuka pengguna yang memungkinkan pengguna berinteraksi dengan aplikasi tersebut [1]. Dapat disimpulkan bahwa keberadaan *User Interface* (UI) sudah menjadi sebuah kebutuhan dalam membuat perangkat lunak. Sebuah UI yang baik adalah kunci suksesnya pengembangan aplikasi. UI yang baik harus memiliki latensi jarak antara permulaan operasi dan perbaruan/ *update* yang kecil. Perangkat lunak dengan UI yang buruk (latensi yang besar) dapat menyebabkan besarnya keluhan pengguna [5].

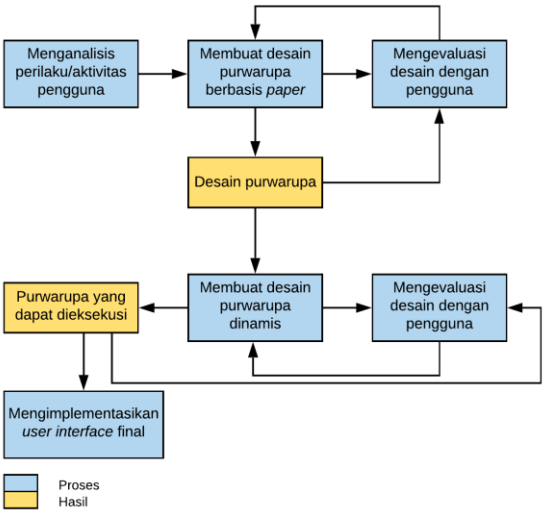
Saat ini, *user interface* yang paling banyak digunakan dalam membuat sebuah perangkat lunak/*software* adalah Graphical *User Interface* (GUI) [1][6][7]. GUI digunakan dengan menambah elemen lain seperti icon, gambar/*graphic*, atau bahan lain yang dapat meningkatkan pengalaman user saat menggunakan software. Keuntungan menggunakan GUI dalam software adalah [6]:

a. Mudah digunakan oleh user yang memiliki pengetahuan komputer yang minim.
b. Meminimalkan kehilangan informasi saat berpindah layar.
c. Memiliki akses penuh pada layar dengan segera untuk beberapa macam tugas/keperluan.

Tahapan-tahapan yang dilakukan dalam merancang sebuah *user interface adalah* [6]:

a. Menganalisis dan memahami kegiatan pengguna
b. Membuat desain purwarupa/*prototype* berbasis makalah/*paper*
c. Mengevaluasi desain dengan pengguna
d. Membuat purwarupa dinamis
e. Mengimplementasikan *user interface* final

Tahapan tersebut dapat digambarkan dalam diagram alur seperti pada Gambar 1.



Gambar 1. Tahapan merancang *user interface*

Proses perancangan *user interface* mengikuti prinsip-prinsip desain sebagai berikut [6]:

a. *User familiarity*: antarmuka menggunakan bahasa dan istilah yang mudah untuk dikenali dan dimengerti.
b. *Consistency*: antarmuka memiliki cara operasi yang sama pada operasi yang sebanding (jika memungkinkan).
c. *Minimal surprise*: antarmuka tidak membuat pengguna terkejut dengan cara kerja sistem.
d. *Recoverability* / antarmuka memiliki mekanisme pemulihan *error.*
e. *User guidance*: antarmuka memberi timbal balik yang dapat dimengerti pengguna.
f. *User diversity*: antarmuka mendukung keberagaman tipe pengguna.

Prinsip tersebut diharapkan dapat memenuhi serta meningkatkan pengalaman pengguna dalam berinteraksi dengan software. Dalam berinteraksi dengan pengguna, *user interface* yang terbagi menjadi lima tipe/gaya interaksi [6]. Kelima tipe interaksi tersebut adalah:

a. **Direct manipulation**: interaksi langsung dengan objek pada layar.
b. **Menu selection**: pemilihan menu dalam daftar yang disediakan.
c. **Form fill-in**: mengisi area-area pada form yang disediakan.
d. **Command language**: menuliskan perintah yang sudah ditentukan oleh program.
e. **Natural language**: penggunaan bahasa manusia untuk mendapatkan hasil yang diinginkan.

Selain interaksi antara user dengan software, sebuah interface yang baik juga perlu menyajikan informasi yang baik dan akurat kepada pengguna [6].

Hal yang menarik adalah pada GUI itu tersendiri, yang biasanya harus bereaksi dan mengkoordinasikan beberapa kejadian yang terjadi. Aplikasi yang diprogram dengan cara biasa akan kesulitan menangani hal ini karena urutan kedatangan *external events* yang tidak dapat diprediksi dan diatur (kapan kejadian terjadi bukan ditentukan oleh *programmer*). Ditambah lagi dengan perubahan data yang harus terus diperbarui secara tepat (urutan dan waktu) pada setiap bagian yang bergantung pada data tersebut. Untuk mengatasi hal tersebut, paradigma *reactive* dinilai sebagai solusi yang tepat dari permasalahan tersebut [1].

Klaim bahwa paradigma *reactive* lebih unggul ketimbang paradigma *pure Object Oriented* sudah pernah diungkapkan sebelumnya pada [8], yang menyimpulkan bahwa program yang ditulis dengan paradigma *reactive* lebih mudah dimengerti ketimbang program dengan paradigma *pure Object Oriented.* Namun, perbandingan tersebut tidak mengungkapkan keunggulan paradigma *reactive* secara teknis pada performa/kinerja suatu aplikasi melainkan hanya keunggulan dari segi kualitatif seorang *programmer* ketika memahami kode program yang dibuat. Sejauh ini belum ditemukan referensi yang mengungkapkan keunggulan paradigma *reactive* dengan membandingkan performa suatu aplikasi identik pada paradigma yang berbeda. Oleh karena itu, dibutuhkan bukti empiris untuk mengungkapkan hal tersebut.

Penelitian mengenai perbandingan paradigma *reactive* dan *pure object oriented* yang dilakukan didasarkan pada *research question* sebagai berikut:

RQ1. Mengapa aplikasi Android memerlukan desain UI yang *reactive*?
RQ2. Apakah paradigma *reactive* lebih baik digunakan untuk aplikasi di Android dibandingkan dengan paradigma *Object Oriented*?
RQ3. Apa saja unsur yang membedakan paradigma *reactive* dengan *pure object-oriented*?

2. KONSEP REACTIVE PROGRAMMING

Reactive Programming terdapat di berbagai bahasa pemrograman. Paradigma ini tersedia sebagai framework/ekstensi dari bahasa pemrograman contohnya RxJava, RxRuby, RxKotlin, dan lain-lain [4].

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* [9]. *Framework* ini digunakan untuk menangani beberapa informasi secara reaktif agar proses manipulasi yang rumit pada *user interface* dapat ditangani dengan mudah [10].

Unsur-unsur pada RxJava adalah [10]:

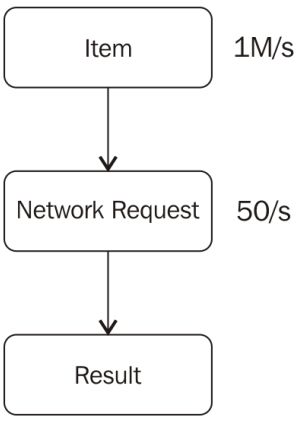
a. **Observable**: sumber data yang dapat diamati.
b. **Subscriptions**: penanganan untuk observable penerima data.
c. **Schedulers**: suatu cara untuk menentukan di mana data akan diproses.

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 dimana kode akan dieksekusi dengan memilih beberapa thread tertentu.

Adapun jenis khusus dari observable, yaitu flowable. Flowable dapat memproses item yang dipancarkan lebih cepat dari sumber daripada kapasitas yang dapat ditangani pada langkah berikutnya. Hal inilah yang menjadi perbedaan antara flowable dan observable. Sebagai contoh kasusnya adalah propagasi informasi dari sebuah sumber yang mengirimkan data sebanyak 1 juta per detik (1M/s). Langkah selanjutnya adalah melakukan *network request*. Namun, *network request* hanya mampu menjalankan 50 request per detik seperti diilustrasikan pada Gambar 2.



Gambar 2. Contoh kasus implementasi *flowable* pada propagasi informasi

Hal di atas menimbulkan masalah yaitu setelah 60 detik, akan ada 60 juta item dalam antrian yang menunggu untuk diproses. Item akan terakumulasi saat 1 juta item per detik antara langkah pertama dan kedua karena langkah kedua memprosesnya lebih lambat daripada langkah pertama .

Selain jenis observable dan flowable, terdapat tiga jenis lain yang disediakan oleh RxJava, yaitu [10]:

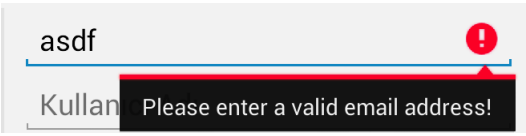
- **Completable**, merupakan tindakan tanpa hasil yang akan diselesaikan nanti.
- **Single**, sama seperti *observable* (atau *flowable*) yang mengembalikan satu item sebagai pengganti *stream*.
- **Maybe**, merupakan tindakan singkat yang dapat menyelesaikan (atau gagal) tanpa mengembalikan nilai apapun (seperti dapat selesai) tetapi juga dapat mengembalikan item seperti pada *single*.

3. METODE

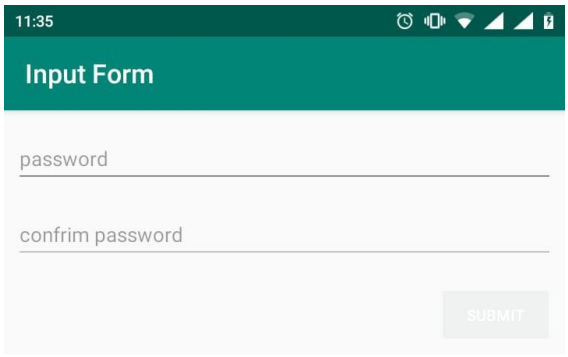
Bagian ini menjelaskan beberapa kegiatan dalam pelaksanaan penelitian.

3.1 Karakteristik Software untuk Studi Kasus

Penelitian dilakukan dengan mengamati aplikasi sederhana yang berisi *form fill-in* dengan dua *field* berupa *text box/text field*: (*password* dan *confirm password*) serta satu *button*. *Text box* password akan mengeluarkan pesan *error* (seperti pada Gambar 3) ketika jumlah karakter input dari pengguna belum mencapai 8 karakter. *Text box confirm password* akan mengeluarkan pesan error jika input dari user belum sama dengan input pada *text box password*. Tampilan akhir aplikasi ditunjukkan pada Gambar 4.



Gambar 3. Contoh pesan *error* pada *text box*



Gambar 4. Tampilan aplikasi

3.2 Teknis Implementasi Paradigma pada Software

Kedua paradigma dipasang aplikasi yang sama dengan implementasi kode yang berbeda. Pada paradigma *pure Object-Oriented* kejadian input user akan ditangkap oleh suatu objek berjenis *EditText* untuk setiap *text field* yang diberi sebuah *listener* berupa objek *TextWatcher* seperti pada Program 1.

```
etPassword.addTextChangedListener(
            new TextWatcher() {...});

etConfirmPass.addTextChangedListener(
            new TextWatcher() {...});
```

Program 1. Potongan implementasi paradigma *pure Object-Oriented*

Sementara pada RP kejadian input diamati pada objek *Observable* oleh *Observer* yang mana setiap perubahan kejadian/*events* diterima oleh suatu *Subscriber* yang menangkap keluaran dari *Observer* seperti pada Program 2.

```
Observer<Boolean>invalidFieldObserver =
        new Observer<Boolean>() {...};
passwordStream.subscribe(
        passwordObserver);
passwordConfirmationStream.subscribe(
        passwordConfirmationObserver);
invalidFieldsStream.subscribe(
        invalidFieldsObserver);
```

Program 2. Potongan implementasi paradigma *reactive programming*

3.3 Parameter Uji dan Teknis Pengukuran

Nilai kinerja aplikasi dibandingkan melalui tiga parameter uji yaitu pemakaian CPU, memory, dan konsumsi energi pada saat *runtime*. Ketiga parameter uji ini penting diukur untuk menunjang antarmuka pengguna yang nyaman dipakai pengguna serta ramah dalam pemakaian sumber daya. Hal tersebut dicapai dengan mengoptimalkan penggunaan CPU dan alokasi memori.

Ketiga parameter uji diukur menggunakan *profiler* pada Android Studio. *Profiler* adalah alat bantu ukur kinerja aplikasi secara langsung (real-time) yang tersemat pada Android Studio. Aplikasi dijalankan pada perangkat Nokia 3 tahun 2017 dengan dengan API Level 28 (Android Pie), besar memori utama 2 Gigabyte (2048 Megabyte), dan chipset MediaTek 6737 quad-core 1,3 GHz.

Aplikasi dijalankan dengan cara melakukan *running* pada Android Studio yang sudah terhubung dengan perangkat uji melalui kabel USB. Aplikasi didiamkan sampai masuk ke MainActivity di mana tampilan pada Gambar 3 sudah muncul, kemudian diketikkan kalimat "suratnuh71" sebagai *string* uji yang dimasukkan ke dalam *field password* dan *confirm password,* kemudian tombol submit ditekan sebanyak 10 kali.

Perilaku tersebut valid dan objektif untuk mengukur kinerja tiap paradigma karena perilaku berpengaruh tepat pada bagian implementasi aplikasi yang merespon input dengan paradigma yang berbeda (*textbox password* dan *confirm password*) seperti yang sudah dibahas pada penjelasan Program 1 dan Program 2.

4. ANALISIS HASIL DAN PEMBAHASAN

Bagian ini menjelaskan temuan dan hasil pengamatan terhadap perilaku uji pada aplikasi.

4.1 Kejadian pada Pengamatan

Berdasarkan metode perlakuan yang sama pada kedua aplikasi, diperoleh data untuk setiap parameter uji (penggunaan CPU, memori, dan konsumsi energi). Kejadian yang diamati dibatasi hanya pada tiga kejadian yaitu:

a. *Initial Minimal Value*: Nilai minimum penggunaan sumber daya saat awal *running* aplikasi.
b. *Initial Peak*: Nilai puncak penggunaan sumber daya saat awal *running* aplikasi.
c. *Range when user doing input*: Rentang penggunaan sumber daya ketika pengguna mengetikkan *string* uji.

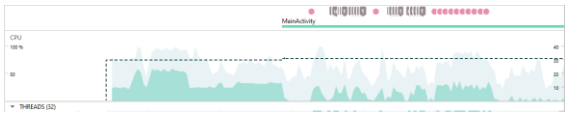
4.2 Hasil Pengamatan terhadap Parameter Uji

Berikut adalah hasil pengamatan pada masing-masing parameter uji.

4.2.1 Penggunaan CPU

Gambar 5 dan 6 menunjukan grafik penggunaan CPU oleh aplikasi di mana sumbu X adalah satuan waktu (detik) dan sumbu Y adalah persentase penggunaan CPU. Data dari gambar 5 dan 6 terangkum pada Tabel 1.

Gambar 5. Pemakaian CPU pada aplikasi dengan paradigma *pure-OO*



Gambar 6. Pemakaian CPU pada aplikasi dengan paradigma *reactive programming*

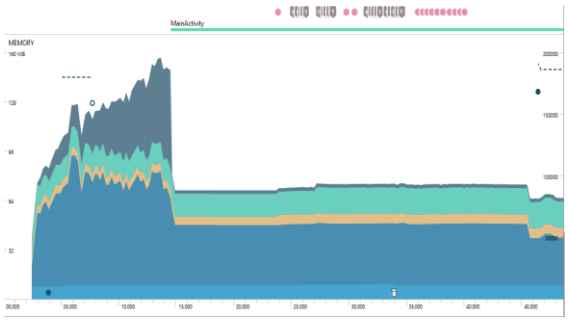
Tabel 1. Persentase pemakaian CPU oleh aplikasi

Kejadian	Dengan RP	Tanpa RP
Init. Min. Value	30.9%	21%
Init. Peak	59.8%	57.9%
Range when user doing input	36.8%	35%

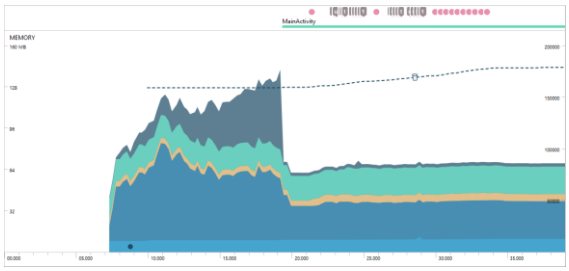
Pada Tabel 1, terangkum data mengenai penggunaan CPU oleh aplikasi. Secara singkat dapat diamati bahwa aplikasi dengan paradigma *reactive* mengkonsumsi lebih banyak kapasitas CPU dibanding yang menggunakan paradigma *pure-OO*. Pada awal aplikasi *running* (*initiate-main activity run*) terlihat aplikasi yang menggunakan paradigma *reactive* menggunakan kapasitas CPU paling kecil 30.9% (*Init. Min. Value*) dan memuncak di 59.8% (*Init. Peak*). Ketika pengguna mengetikkan *string* uji, aplikasi dengan paradigma *reactive* mencapai penggunaan CPU tertinggi sebesar 36.8%. Penggunaan kapasitas CPU ini lebih besar ketimbang program dengan paradigma *pure-OO* yang hanya memakai 35% CPU saat pengguna melakukan input *string* uji.

4.2.2 Penggunaan Memory

Hal yang sama terjadi pada penggunaan memori utama secara keseluruhan. Gambar 7 dan 8 menunjukkan grafik pemakaian memori utama oleh aplikasi dengan sumbu X sebagai besaran waktu (detik) dan sumbu Y sebagai besaran memori utama yang terpakai dalam satuan Megabyte (Mb).



Gambar 7. Pemakaian memory pada aplikasi dengan paradigma pure-OO



Gambar 8. Pemakaian memory pada aplikasi dengan paradigma *reactive programming*

Data dari Gambar 7 dan 8 dirangkum pada Tabel 2. Total pemakaian memori utama secara keseluruhan diambil dari grafik berwarna abu gelap berupa akumulasi dari tumpukan grafik di bawahnya.

Tabel 2. Jumlah total memori utama terpakai dalam Megabyte (Mb)

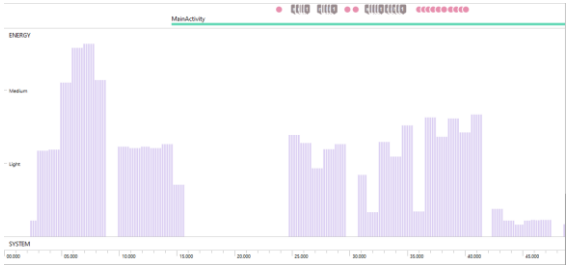
Kejadian	Dengan RP	Tanpa RP
Init. Min. Value	43.4	35.9
Init. Peak	141.8	138.2
Range when user doing input	62.9 - 70.8	62 - 66.2

Tabel 2 menunjukkan total memori yang terpakai saat *runtime* aplikasi uji berlangsung. Program dengan paradigma *pure Object Oriented* memakai jumlah total memori yang lebih kecil dibanding dengan program dengan paradigma *reactive*.

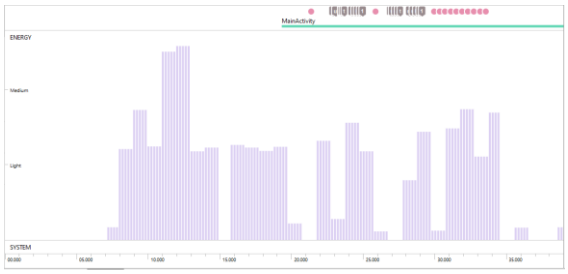
Pada awal aplikasi dijalankan, aplikasi dengan paradigma *pure-OO* hanya mencapai pemakaian maksimal di 138.2 Mb atau lebih kecil sebesar 2.5% dari aplikasi yang menggunakan paradigma *reactive.*

Hal serupa terjadi saat kejadian memasukkan *string* uji dari pengguna. Terlihat bahwa kisaran memori yang digunakan pada aplikasi dengan paradigma *pure Object Oriented* hanya berkisar di 62 s.d. 66.2 Mb, sementara aplikasi dengan paradigma *reactive* memiliki rentang penggunaan memori sebesar 62.9 s.d. 70.8 Mb.

4.2.3 Konsumsi energi



Gambar 9: Konsumsi energi tanpa RP



Gambar 10: Konsumsi energi dengan RP

Berbeda dengan energi yang dihabiskan oleh aplikasi yang digambarkan pada Gambar 9 dan Gambar 10 di mana sumbu X adalah satuan waktu dan sumbu Y adalah intensitas konsumsi energi (Low, Medium, Heavy).

Dari Gambar 9 dan 10, dapat diamati bahwa pada saat inisiasi keduanya sama-sama mencapai penggunaan energi yang tinggi/*heavy*.

Saat pengguna mengetikkan *string* uji ke dalam *field* yang tersedia. Terlihat bahwa aplikasi dengan paradigma *reactive* lebih unggul dan memiliki kemunculan *bar* (penggunaan energi) yang lebih sedikit dibandingkan dengan aplikasi dengan paradigma *pure Object Oriented*, meskipun dapat diamati juga bahwa keduanya tidak memakan energi yang besar dan hanya berada di level konsumsi energi menengah/*medium*.

Sampai dengan hasil pengamatan ini, dapat dilihat bahwa keunggulan performa aplikasi Android sederhana (berdasarkan parameter uji) masih dimiliki oleh aplikasi dengan paradigma *pure Object Oriented*. Sebagai tambahan, jika banyaknya kode program (*line of code*) menjadi ukuran suatu program lebih mudah dipahami, maka pernyataan bahwa program dengan paradigma *reactive* lebih mudah dipahami [8], tidak selalu benar karena Program 1 memiliki *line of code* yang lebih sedikit.

Hasil pengamatan pada tiga parameter uji tersebut digunakan untuk menjawab RQ2 yang dijelaskan pada bagian 5 (Kesimpulan).

5. KESIMPULAN

Penjelasan mengenai konsep *reactive programming* pada bagian 2 dan isu implementasi GUI pada bagian 1 dapat menjadi **jawaban atas RQ1**. Implementasi RP pada GUI aplikasi selaras dengan pernyataan pada bagian 1 bahwa GUI harus merespon kejadian yang terjadi. Implementasi RP dibutuhkan karena penanganan secara konvensional dinilai sulit untuk mensinkronisasi perubahan yang terjadi pada data secara global.

Penjelasan konsep *reactive programming* pada bagian 2 **juga menjawab RQ3** di mana pada pemrograman *reactive* digunakan *design pattern* khusus yaitu Observer Pattern (dan *object* yang terkait dengannya). Selebihnya dijawab pada bagian implementasi di mana paradigma *pure Object-Oriented* mengamati perubahan pada objek dengan menyematkan objek berupa *listener* sementara paradigma *reactive* mengamati perubahan pada objek melalui *Observer*.

Pada dasarnya *framework reactive* digunakan untuk mengatasi permasalahan-permasalahan dalam memanipulasi kejadian dan data. Contohnya saat aplikasi memerlukan data yang selalu akurat maka framework ini akan sangat membantu sekali. Akan tetapi *framework reactive* direalisasikan dengan penggunaan *multithreading* (untuk mendukung pemrosesan secara asynchronous) sehingga mengakibatkan penggunaan CPU/*CPU usage* yang lebih tinggi. Konsep dari *multithreading* secara singkat adalah dengan memanfaatkan jumlah *thread* yang ada dalam CPU untuk mengerjakan tugas secara bersamaan.

Perbedaan antara pemrograman dengan paradigma *reactive* dan *pure Object Oriented* tidak akan terlihat dari segi antarmuka. Hal ini dikarenakan yang membedakan pemrograman dengan paradigma *reactive* dan *pure-Object Oriented* adalah bagaimana suatu program tersebut diimplementasikan dan bukan bagaimana suatu data disajikan.

Sebagai jawaban atas RQ2, *Reactive Programming* pada aplikasi Android belum tentu memiliki pengaruh yang signifikan terhadap kinerja aplikasi ketika dibandingkan dengan aplikasi yang dibangun dengan paradigma pure Object Oriented. Penggunaan paradigma *reactive* pada kasus tertentu bisa jadi lebih baik dan optimal, meskipun sebenarnya paradigma *reactive* ditujukan bukan untuk mengoptimasi suatu sistem agar lebih baik dalam performa, tetapi untuk membuat aplikasi lebih mudah ketika menangani banyak kejadian/*event* yang memungkinkan terjadi secara *asynchronous* dengan mengorbankan penggunaan kapasitas CPU lebih karena membutuhkan thread yang lebih banyak. Namun, penggunaan thread pada *reactive programming* dapat diatur sehingga diwujudkan suatu *thread-safe reactive programming* seperti yang dilakukan pada [11].

Sebagai kesimpulan akhir (dari seluruh jawaban RQ), penggunaan *reactive extension* pada UI aplikasi Android sangat bergantung pada kebutuhan aplikasi yang dibangun. Ketika aplikasi membutuhkan suatu sistem yang dapat menangani aliran kejadian dan data yang melimpah secara *asynchronous* maka pengembangan dengan paradigma *reactive* dinilai menjadi solusi. Ketika aplikasi yang dibangun tidak memiliki limpahan aliran kejadian dan data, implementasi dengan *pure Object Oriented* dinilai cukup untuk memenuhi kebutuhan aplikasi yang dibangun.

DAFTAR PUSTAKA

[1] E. Bainomugisha, A. Carreton, T. Cutsem, S. Mostinckx and W. Meuter, "A survey on reactive

programming", *ACM Computing Surveys*, vol. 45, no. 4, pp. 1-34, 2013. Available: 10.1145/2501654.2501666.

[2] M. Alessandro and S. Guido, "On the Semantics of Distributed Reactive Programming: the Cost of Consistency," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* 2018.

[3] J. Lewis and W. Loftus, *Java Software Solutions Foundations of Program Design*, 8th ed. Harlow: Pearson, 2015, ch. 1, pp. 30-72.

[4] "ReactiveX/RxJava", *GitHub*, 2016. [Online]. Available: https://github.com/ReactiveX/RxJava/wiki. [Accessed: 06- Aug- 2020].

[5] Y. Kang, Y. Zhou, M. Gao, Y. Sun and M. Lyu, "Experience Report: Detecting Poor-Responsive UI in Android Applications", *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, p. 1, 2016.

[6] A. Memon, "User Interface Design Designing Effective Interfaces for Software Systems", Maryland, 2003.

[7] P. Valero-Mora and R. Ledesma, "Graphical User Interfaces for R", *Journal of Statistical Software*, vol. 49, no. 1, pp. 1-8, 2012.

[8] G. Salvaneschi, S. Proksch, S. Amann, S. Nadi and M. Mezini, "On the Positive Effect of Reactive Programming on Software Comprehension: An Empirical Study", *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1-19, 2017. Available: 10.1109/tse.2017.2655524.

[9] C. Arriola and A. Huang, Reactive Programming on Android with RxJava, MYNAH Software, 2017.

[10] T. Subonis, *Reactive Android programming*. Packt Publishing Ltd., 2017, pp. 23-35.

[11] D. Joscha, M. Ragnar, S. Guido and M. Mira, "Thread-Safe Reactive Programming," *OOPSLA,* vol. II, 2018.