IMPLEMENTASI ARSITEKTUR MICROSERVICES PADA BACKEND COMRADES

Cahyanto Setya Budi[1], Adam Mukharil Bachtiar[2]

[1,2]Program Studi Teknik Informatika, Universitas Komputer Indonesia
Jalan Dipatiukur No. 112-116, Coblong, Bandung, Jawa Barat
E-mail : cahyantosetyabudi@gmail.com[1], adam@email.unikom.ac.id[2]

ABSTRAK

Comrades merupakan aplikasi pintar sebagai media informasi dan edukasi tentang HIV/AIDS. Aplikasi *Comrades* membantu para pengidap HIV/AIDS untuk saling bertukar informasi dan dukungan serta dapat saling terhubung baik dengan masyarakat umum maupun dengan sesama pengidap HIV/AIDS. Dalam perkembangannya *Comrades* juga bekerja sama dengan lembaga terkait salah satunya komunitas pengidap HIV/AIDS yang berada didaerah sekitarnya. Direncanakan pengguna dari aplikasi Comrades akan semakin meningkat karena adanya kerja sama tersebut. Untuk mengatasi masalah tersebut maka dibutuhkanlah peningkatan performa yang ada pada aplikasi *Comrades*. Tujuan dari penelitian ini adalah untuk meningkatkan performa dan *availability* pada aplikasi *Comrades*. Hal ini dapat dicapai dengan menerapkan arsitektur *microservices* pada *web services Comrades* dari yang sebelumnya *monolithic*. Pada penelitian ini proses perubahan arsitektur pada aplikasi *Comrades* akan menggunakan pendekatan *Domain Driven Design* pada pemecahan *service* nya. Sedangkan pada sisi arsitektur fisiknya juga akan mengalami perubahan dengan menerapkan beberapa teknologi yang ada saat ini di antaranya seperti: *Docker*, *Kubernetes* dan *API Gateway*.

Kata kunci : *Microservices, Web Services*, API, *Domain Driven Design, Monolithic, Performance*

1. PENDAHULUAN

1.1 Latar Belakang

Comrades merupakan aplikasi pintar sebagai media informasi dan edukasi tentang HIV/AIDS. *Comrades* yang merupakan aplikasi pintar berbasis teknologi, membantu para pengidap HIV/AIDS untuk saling bertukar informasi dan dukungan serta dapat saling terhubung baik dengan masyarakat umum maupun dengan sesama pengidap HIV/AIDS. Pada aplikasi *Comrades* terdapat beberapa fitur di antaranya seperti: chat, informasi artikel, berita dan *event*, *tweet* positif dan sticker yang dapat dikirim sebagai dukungan untuk pengidap ODHA. Dalam prosesnya *Comrades* juga bekerja sama dengan komunitas pengidap HIV/AIDS yang berada didaerah sekitarnya. Disisi lain dengan meningkatnya jumlah komunitas yang ikut bekerja sama di dalam aplikasi ini, direncanakan hal tersebut juga akan meningkatkan jumlah pengguna dari waktu ke waktu. Di samping itu, supaya aplikasi dapat terus melayani secara optimal terdapat beberapa kebutuhan yang harus dipenuhi seperti: performa, *availability*, dan ketahanan saat mengalami tingginya request dari pengguna [1]. Dengan demikian terdapat jenis arsitektur yang sedang berkembang saat ini yang dapat memenuhi kebutuhan tersebut yaitu, *microservices*.

Pada tahap awal pembangunannya aplikasi *Comrades* menggunakan jenis arsitektur *monolithic*. Arsitektur ini menerapkan aspek fungsional dari web *service* yang menggunakan *code base* dan teknologi yang seragam dalam penerapannya [2]. Hal ini terjadi karena tuntutan aplikasi untuk dapat dibangun secara cepat dan dengan keterbatasan jumlah tim yang tersedia. Seiring berjalanya kebutuhan bisnis dalam aplikasi, arsitektur *monolithic* akan menjadi semakin besar disisi lain arsitektur *monolithic* juga akan sulit untuk dikembangkan [3]. Masalah mulai muncul ketika fungsional pada aplikasi yang semakin banyak, dan *traffic* yang terus meningkat. Sehingga saat suatu fungsional mengalami *error* maka akan berdampak pada keseluruhan *web services*. Dari analisis pengujian menggunakan *Blazemeter* yang dilakukan terhadap aplikasi *web services* pada *Comrades*, diperoleh informasi bahwa aplikasi belum bekerja dengan optimal karena masih terdapat kekurangan dari sisi *web services Comrades*. Hal ini dibuktikan dengan *web services* membutuhkan waktu setidaknya 2,5 detik dalam melakukan *response*.

Dengan demikian, berdasarkan permasalahan yang terjadi serta studi literatur yang telah dilakukan sebelumnya maka dilakukanlah sebuah penelitian yaitu berupa mengubah arsitektur pada aplikasi *Comrades* dari *monolithic* ke *microservices*. *Microservices* adalah desain arsitektur aplikasi yang memecah aplikasi menjadi *service – service* kecil yang terpisah sesuai dengan fungsinya [4]. Pemecahan ini bertujuan untuk menciptakan web *services* yang *scalable*, *resilence*, dan *high-availability*. Penerapan arsitektur *microservices* diharapkan dapat menangani masalah yang terjadi pada aplikasi *Comrades*.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang telah dipaparkan di atas maka dapat dirumuskan suatu masalah yaitu seberapa besar pengaruh performa yang dihasilkan dengan menerapkan arsitektur *microservices* pada *web services Comrades*.

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk meningkatkan performa pada aplikasi Comrades.

1.4 Batasan Masalah

Adapun batasan masalah dalam penelitian ini adalah sebagai berikut:

a. Dalam penelitian ini analisis yang dilakukan fokus kepada performa yang tercatat dalam *Quality of Services (QOS) Web Services*.

b. Penerapan arsitektur *microservices* pada penelitian hanya fokus kepada API *web services* pada aplikasi.

c. Pada proses pertukaran informasi antar *client* dengan *server* menggunakan format JSON dengan mekanisme REST, dikarenakan mekanisme ini lebih ringan dibanding yang lain [5].

d. Pada penelitian ini proses dalam memecah aplikasi menjadi *microservices* menggunakan pendekatan *Domain Driven Design*.

2. ISI PENELITIAN

2.1 Landasan Teori

2.1.1 *Web Services*

Web Service adalah salah satu komponen pada sistem terdistribusi yang berbasis *Service Oriented Architecture* (SOA) [6]. SOA merupakan arsitektur perancangan aplikasi yang menggunakan komponen-komponen dari kebutuhan bisnis yang sudah ada sebelumnya. Dalam hal ini aplikasi yang dibangun akan bersifat modular sesuai komponen-komponen tersebut, komponen ini disebut juga sebagai *service*. *Service* yang dibangun memiliki *interface* yang berfungsi memproses dan mengirim pesan dalam bentuk XML atau JSON. *Service* yang dibuat menggunakan konsep *object oriented* dalam proses pengiriman *request* maupun *responsenya*.

2.1.2 *Microservices*

Microservices adalah desain arsitektur aplikasi yang memecah aplikasi menjadi *service – service* kecil yang terpisah sesuai dengan fungsinya. Service akan dibagi menjadi lebih rinci lagi dari segi fungsionalitasnya agar setiap fungsi bekerja secara independen [3]. Dalam hal ini setiap *service* mungkin saja dapat mempunyai teknologi *stack* yang berbeda antara satu dengan yang lain tergantung dari kebutuhan *service*. Ini artinya akan terdapat teknologi yang berbeda – beda dalam satu aplikasi besar.

2.1.3 *Domain Driven Design*

Domain – Driven Design (DDD) merupakan pendekatan pembangunan perangkat lunak dalam modeling suatu aplikasi, DDD membantu dalam proses pengembangan perangkat lunak menjadi lebih akurat sesuai dengan proses bisnis yang akan berjalan pada aplikasi [7]. Salah satu masalah yang sering muncul dalam proses merancang aplikasi adalah komunikasi antar *class* atau fungsi yang seiring berkembangnya aplikasi maka akan semakin besar [8]. Masalah ini akan menimbulkan *dead code* yaitu saat kode aplikasi yang besar tidak berani disentuh developer karena takut akan menimbulkan kesalahan. Semakin lama *dead code* akan semakin besar dan bahkan bertambah, hal ini menjadi sesuatu yang menakutkan bagi developer.

2.1.4 *Performance for Web Service*

Dalam membangun suatu *web service* yang baik dibutuhkan suatu standar sehingga *web service* yang dibangun dapat bekerja dengan baik sesuai dengan tujuannya [9]. Standar ini harus mencakup banyak aspek sehingga *web service* tidak hanya sesuai dengan tujuan, akan tetapi baik dari segi *performansi*, *biaya*, *maintainability*, dsb. sudah dapat dikatakan optimal. Adapun standar-standar yang harus dipenuhi tersebut meliputi [1]:

a. *Response Time*

Response time merupakan total keseluruhan waktu yang digunakan *service* dalam memberikan *respon* terhadap *request* yang datang.

b. *Throughput*

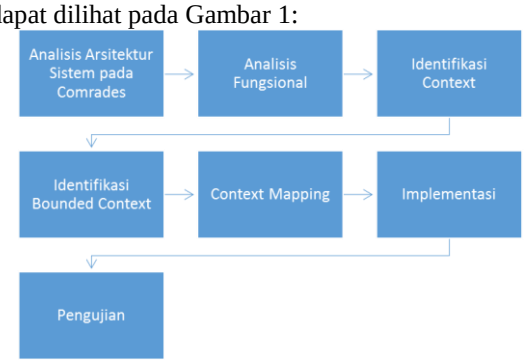
Throughput merupakan jumlah *request* dari *service* sama yang dapat dilayani dalam satuan waktu tertentu.

c. *Latency*

Latency adalah lamanya waktu yang dibutuhkan *service* dalam mengirimkan *response* dari server ke *client*

2.2 Metode

Metodologi penelitian yang digunakan dalam penelitian ini adalah dengan pendekatan kuantitatif yang menggunakan jenis penelitian komparatif. Penelitian ini bertujuan membandingkan antara dua objek dengan variabel tertentu. Dalam kasus ini objek yang dibandingkan merupakan tingkat performa *web services* pada aplikasi *Comrades* yang menggunakan arsitektur sebelumnya dan arsitektur yang akan diterapkan yaitu *microservices*. Berikut adalah tahapan-tahapan pada penelitian ini yang dapat dilihat pada Gambar 1:



Gambar 1 Metodologi Penelitian

Analisis Arsitektur Sistem pada *Comrades*
Pada tahap ini dilakukan analisis mengenai arsitektur dari keseluruhan aplikasi yang sedang berjalan.

Analisis Fungsional
Pada tahapan ini akan di analisis fungsional yang ada pada sistem beserta deskripsinya. Fungsional ini menjelaskan proses bisnis apa saja yang terdapat dalam aplikasi.

Identifikasi *Context*
Pada tahapan ini dilakukan pengelompokan dari beberapa fungsional yang sudah dijelaskan menjadi satu *context* yang sama.

Identifikasi *Bounded Context*
Pada tahapan ini dilakukan untuk mengetahui batasan-batasan yang dimiliki oleh masing-masing *context*.

Context Mapping
Pada tahapan ini dilakukan untuk menggambarkan keseluruhan *context* beserta jenis hubungannya antara satu dengan yang lain.

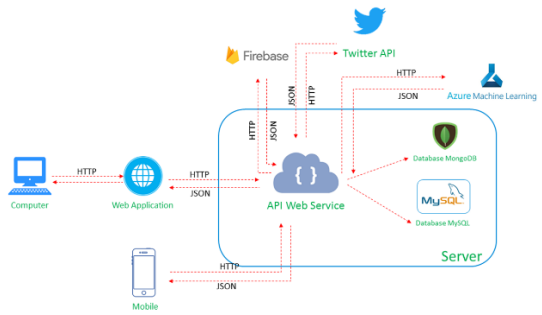
Implementasi
Pada tahapan ini akan dilakukan penerapan *web services* ke dalam kode, sesuai dengan perancangan *microservices* yang sudah telah dilakukan sebelumnya.

Pengujian
Pada tahap ini dilakukan pengujian *web service* yang sudah dibuat. Pengujian ini bertujuan untuk mengetahui apakah *web services* yang sudah dibuat sudah sesuai. Pengujian ini juga berguna untuk mengetahui apakah pada arsitektur yang baru *web services* mengalami peningkatan performa.

2.3 Hasil dan Pembahasan

2.3.1 Analisis Arsitektur Sistem pada *Comrades*

Pada tahap ini dilakukan analisis mengenai arsitektur dari keseluruhan aplikasi yang sedang berjalan. Adapun arsitektur yang sedang berjalan pada aplikasi *Comrades* dapat dilihat pada Gambar 2:



Gambar 2 Arsitektur Sistem *Comrades*

Setelah diketahui arsitektur keseluruhan yang sedang berjalan maka selanjutnya di analisis masalah apa saja yang terdapat pada arsitektur tersebut.

2.3.2 Analisis Masalah

Pada saat ini beberapa *web services* masih lama dalam melakukan *response* hal ini dibuktikan melalui pengujian menggunakan tools *Blazemeter*. Adapun hasil pengujian yang telah dilakukan dapat dilihat pada Tabel 1:

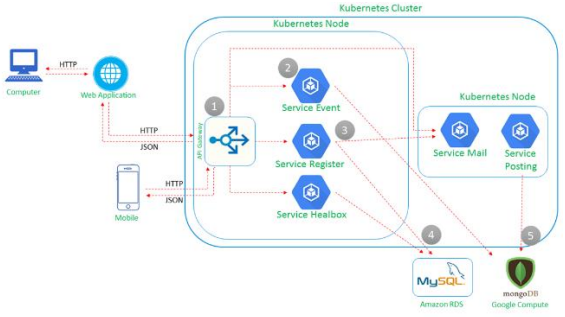
Tabel 1 Hasil Pengujian Arsitektur Lama

Nama Service	Respon Rata – Rata (ms)	Respon Minimal (ms)	Respon Maksimal (ms)
Admap All Posting	4932.926	4043	10036
Admapp All Artikel	3273.912	2483	4190
List All Artikel	3131.818	2135	5268
List All Berita	2721.784	2127	4005
Admapp Berita by Admin	1856.038	1455	3189
Ambil Tweet	1713.467	1603	2314
List 5 Artikel	1731.357	1032	4197

Setelah diketahui masalah yang terdapat pada arsitektur *Comrades* maka dilakukanlah perubahan arsitektur dari *monolithic* menjadi *microservices*. Perubahan ini diharapkan nantinya dapat meningkatkan performa dari *web service Comrades*.

2.3.3 Analisis Arsitektur Sistem Baru *Comrades*

Pada tahap ini bertujuan menjelaskan arsitektur sistem yang akan diterapkan pada *web services Comrades*. Adapun gambaran keseluruhan dari arsitektur yang akan diterapkan dapat dilihat pada Gambar 3:



Gambar 3 Arsitektur Keseluruhan *Microservices*

Berikut adalah deskripsi mengenai arsitektur baru yang akan diterapkan pada *web services Comrades*:

a. Pada sisi *API Gateway* akan menggunakan *Ambassador*, *ambassador* berguna menangani *request* yang datang dan melanjutkan kepada *services* dituju yang terdapat di dalam *kubernetes service*. Ambassador juga sudah dilengkapi dengan *load balancer* yang akan membagi *request* jika dianggap melebihi kapasitas.

b. Pada sisi *service*, setiap *service* mewakili *service* yang berada pada *web service*. Di dalam *service* mungkin akan berisi beberapa *pods*, hal itu tergantung kebutuhan *service* dalam melayani *request* dari pengguna.

c. Pada sisi *service*, setiap *service* memungkinkan untuk saling berkomunikasi jika memang dibutuhkan ini berarti terjadi ketika suatu *microservices* saling berkomunikasi dengan *microservices* lain. Komunikasi tidak hanya dapat dilakukan di dalam *node* saja namun dapat juga dilakukan di luar *node*. Mekanisme yang digunakan adalah menggunakan http dan format JSON. Untuk itu dibutuhkan *Service Discovery*, *service discovery* adalah mekanisme yang dilakukan untuk menemukan *service* yang tersedia beserta alamatnya [10].

d. Pada *database MySQL*, seluruh *service* yang ada akan langsung menuju *database* yang digunakan.

e. Pada *database MongoDB*, seluruh *service* yang ada akan langsung menuju *MongoDB* yang digunakan.

2.3.4 Analisis Fungsional

Pada tahapan ini akan di analisis fungsional yang ada pada sistem beserta deskripsinya. Fungsional ini menjelaskan proses bisnis apa saja yang terdapat dalam aplikasi [11]. Pada analisis fungsional akan menggunakan pemodelan *use case* diagram. Dari hasil analisis ini didapatkan 23 fungsional yang terdapat pada aplikasi *Comrades*. Berikut adalah beberapa fungsional yang terdapat pada aplikasi *Comrades* yang dapat dilihat pada Tabel 2:

Tabel 2 Fungsional pada Aplikasi *Comrades*

Use Case	Deskripsi
Melakukan Pendaftaran	Fungsionalitas ini digunakan oleh pengguna umum untuk mendaftarkan dirinya ke dalam sistem
Melihat *Tweet* Dukungan	Fungsionalitas ini digunakan untuk mengetahui *tweet-tweet* dukungan untuk ODHA
Melihat Lokasi dan Pelayanan Obat	Fungsionalitas ini digunakan untuk mengetahui tempat pelayanan dan obat ODHA yang terdekat
Merekomendasi Tempat	Fungsionalitas ini digunakan untuk merekomendasikan tempat yang dapat digunakan oleh ODHA untuk mendapatkan pelayan mengenai HIV/AIDS

Setelah fungsional pada aplikasi *Comrades* diketahui, maka selanjutnya fungsional-fungsional yang sama akan dikelompokkan ke dalam context.

2.3.5 Identifikasi *Context*

Pada tahap ini pengelompokan bertujuan untuk mengetahui fungsional mana saja yang sejenis. Adapun *context* yang terdapat pada *web services Comrades* dapat dilihat pada Tabel 3:

Tabel 3 Daftar *Context* pada Aplikasi *Comrades*

No.	Nama Context
1.	*Event*
2.	*Friends*
3.	*Healbox*
4.	Lokasi Obat
5.	Notifikasi
6.	*Posting*
7.	*SMS*
8.	*Twitter*
9.	*User*
10.	*Email*
11.	*AES*
12.	*Sticker*

Dari hasil identifikasi *context* yang telah dilakukan maka didapatkan 12 *context* yang terdapat aplikasi *Comrades*.

2.3.6 Identifikasi *Bounded Context*

Ciri khas yang dimiliki oleh arsitektur *microservices* terdapat pada *autonomus* setiap *servicenya*, untuk itu dilakukanlah analisis *bounded context*. Analisis ini berfungsi untuk mengetahui batasan apa saja yang terdapat di dalam *context*. Adapun daftar *bounded context* yang terdapat pada *context event Comrades* dapat dilihat pada Tabel 4:

Tabel 4 *Bounded Context* pada Aplikasi *Comrades*

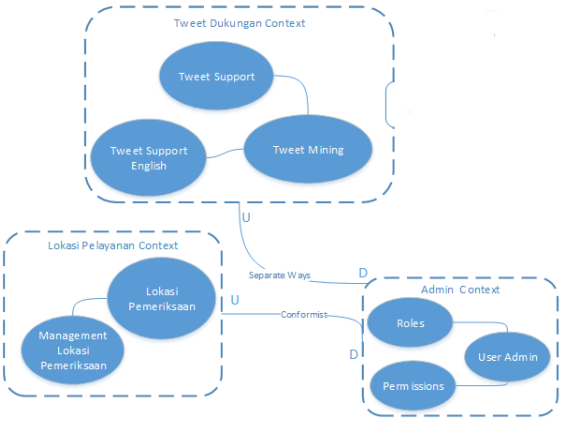
Bounded Context	Deskripsi
Event	Mengatur data acara pada *web service*
Notifikasi	Mengirim notifikasi *event* baru pada aplikasi *mobile*

Dari hasil identifikasi total didapatkan 21 *bounded context* yang tersebar pada aplikasi *Comrades*.

2.3.7 *Context Mapping*

Pada tahapan ini seluruh *bounded context* yang sudah di identifikasi akan digambarkan secara

keseluruhan. Adapun potongan *context* map yang terdapat pada aplikasi *Comrades* dapat dilihat pada Gambar 4:

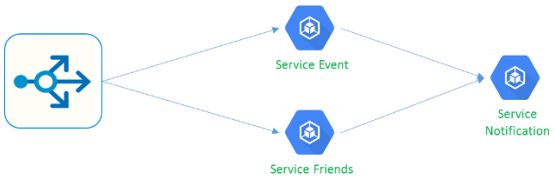


Gambar 4 *Context Map* pada Aplikasi *Comrades*

Berdasarkan hasil *context map* yang telah digambarkan, maka selanjutnya akan di implementasikan *microservices* yang terdapat pada aplikasi *Comrades*.

2.3.8 Implementasi

Pada tahapan ini dilakukan penerapan *microservices* yang akan dilakukan pada aplikasi *Comrades*. Arsitektur dibuat berdasarkan *context* yang di identifikasi sebelumnya pada subbab Identifikasi *Context*. Adapun beberapa hasil dari implementasi *microservices* yang telah dilakukan dapat dilihat pada Gambar 5:



Gambar 5 Implementasi *Microservices* pada *Web Services Comrades*

2.3.9 Pengujian

Pengujian sistem berguna untuk melihat dan menemukan kekurangan-kekurangan yang dimiliki oleh sistem baru yang telah diimplementasikan, sehingga dapat membantu pengembangan sistem di masa yang akan datang. Selain itu pengujian sistem juga dilakukan untuk mengetahui apakah sistem yang baru lebih baik dari sebelumnya.

3. PENUTUP

3.1 Kesimpulan

Berdasarkan hasil pengujian, maka diperoleh kesimpulan sebagai berikut:

a. Pada pengujian integrasi arsitektur sistem yang baru dapat disimpulkan bahwa fungsional-fungsinal *web services* sudah berjalan sesuai dengan tugasnya masing-masing

b. Pada pengujian performa *web services* dengan jumlah *sample* yang sama, arsitektur *web services* yang menggunakan *microservices* terbukti lebih unggul dibandingkan *monolithic*. Adapun hasil dari pengujian yang telah dilakukan dapat dilihat pada Tabel 5 dan Tabel 6**Error! Reference source not found.**.

Tabel 5 Hasil Pengujian pada Arsitektur *Monolithic*

Fungsional	Avg. Response Time (ms)	Avg. Latency (ms)	Avg, Throughput (second)
Admapp Artikel	3273	2419	0.051
Admapp Berita	2720	1995	0.05
Admapp Berita by Admin	1856	1425	0.05
Artikel	1713	1506	0.052
Berita	1504	1296	0.051
Detail Lokasi Obat	1299	1091	0.05

Tabel 6 Hasil Pengujian pada Arsitektur *Microservices*

Fungsional	Avg. Response Time (ms)	Avg. Latency (ms)	Avg, Throughput (second)
Admapp Artikel	654.78	483.99	0.01
Admapp Berita	544.00	399.14	0.01
Admapp Berita by Admin	371.21	285.03	0.01
Artikel	300.85	259.27	0.01
Berita	259.87	218.36	0.01
Detail Lokasi Obat	87.53	87.51	0.01

3.2 Saran

Pada penelitian tingkat performa dari *web services* diukur berdasarkan *Quality of Web Services* (QOS) sebagai acuannya. Namun dari keseluruhan poin yang terdapat dalam QOS hanya beberapa poin saja yang diterapkan pada penelitian ini. Dengan demikian saran yang menggunakan topik penelitian yang sama adalah sebagai berikut:

a. Pada QOS sebagai acuannya, poin yang akan digunakan dalam perbandingan ditambah.

b. Pada penerapan arsitektur *microservices* tidak hanya berfokus pada *web services* saja, tetapi juga dari sisi lain.

DAFTAR PUSTAKA

[1] T. Rajendran dan P. Balasubramanie, "Analysis on the Study of QoS-Aware Web Services Discovery," *J. Comput.*, vol. 1, no. 1, hal. 2151–9617, 2009.

[2] A. Messina, R. Rizzo, P. Storniolo, A. Urso, dan I. Cnr, "A Simplified Database Pattern for the Microservice Architecture," no. c, hal. 35–40, 2016.

[3] K. Katuwal, "Microservices : A Flexible Architecture for the Digital Age Version 1 . 1," vol. 3, no. 4, hal. 23–28, 2016.

[4] B. Familiar, *Microservice Architecture*. 2015.

[5] A. Balalaie, A. Heydarnoori, dan P. Jamshidi, "Microservices Migration Patterns," no. 1, hal. 1–21, 2015.

[6] W3C.org, "Web Services Architecture." [Daring]. Tersedia pada: https://www.w3.org/TR/ws-arch/#whatis. [Diakses: 25-Agu-2018].

[7] V. Vernon, *Domain-Driven Design Distilled*. 2015.

[8] T. N. Millett Scott, *Patterns, Principles and Practices of Domain-Driven Design*. 2015.

[9] M. S. Sharmila dan E. Ramadevi, "Analysis of Performance Testing on Web Applications," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 3, no. 3, hal. 5258–5260, 2014.

[10] F. Montesi dan J. Weber, "Circuit Breakers, Discovery, and API Gateways in Microservices," no. September, 2016.

[11] A. Suhendra dan A. M. Bachtiar, "Migration Code Pada Backend Crimezone Dari Php Ke Scala," vol. 12, no. 2, hal. 74–81, 2016.