Terbit *online* pada laman : http://teknosi.fti.unand.ac.id/

Jurnal Nasional Teknologi dan Sistem Informasi

| ISSN (Print) 2460-3465 | ISSN (Online) 2476-8812 |

Artikel Ilmiah

Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik

Rizki Mufrizal[a], *Dina Indarti*[b]

[a]*Magister Manajemen Sistem Informasi, Universitas Gunadarma, Jl. Margonda Raya Pondok Cina, Depok 16424, Indonesia*
[b]*Teknik Informatika, Universitas Gunadarma, Jl. Margonda Raya Pondok Cina, Depok 16424, Indonesia*

INFORMASI ARTIKEL

Sejarah Artikel:
Diterima Redaksi: 02 Februari 2019
Revisi Akhir: 19 April 2019
Diterbitkan *Online*: 30 April 2019

KATA KUNCI

Monolithic

Microservice

Refactoring

Strangler Pattern

Absensi

KORESPONDENSI

Telepon: +62 87889255404

E-mail: mufrizalrizki@gmail.com

A B S T R A C T

Meningkatnya jumlah pengguna aplikasi berbasis arsitektur *monolithic* dapat mempengaruhi proses pemeliharaan, kinerja dan kompleksnya pembaruan aplikasi. *Resilient challenges* merupakan salah satu permasalahan yang sering terjadi pada arsitektur *monolithic*, dimana jika terjadi kegagalan pada saat pembaruan atau penambahan fitur baru pada aplikasi, maka seluruh fitur aplikasi akan mengalami kegagalan sistem. Permasalahan selanjutnya jika terjadi perubahan salah satu modul aplikasi pada arsitektur *monolithic*, maka dibutuhkan proses *restart* seluruh aplikasi. Semakin besar sebuah aplikasi, maka proses *restart* aplikasi menjadi semakin lama sehingga selama proses *restart* berlangsung, aplikasi tidak dapat digunakan. Dengan berbagai permasalahan pada arsitektur *monolithic*, penggunaan arsitektur *microservice* dalam mengembangkan sebuah aplikasi dapat memperbaiki permasalahan pada arsitektur *monolithic* melalui pemisahan servis-servis menjadi kecil. Untuk dapat mengatasi permasalahan aplikasi dengan menggunakan arsitektur *monolithic*, diperlukan proses *refactoring* aplikasi dari arsitektur *monolithic* menjadi arsitektur *microservice*. Pada penelitian ini, dilakukan *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* dengan menggunakan 13 tahapan dan strategi *strangler pattern* pada aplikasi absensi PT. Graha Usaha Teknik. Proses *refactoring* dengan strategi *strangler pattern* hanya digunakan pada 10 tahapan dari 13 tahapan *refactoring*. Tahapan strategi *strangler pattern* dilakukan hingga seluruh servis pada arsitektur *monolithic* berhasil dilakukan *refactoring*. Hasil *refactoring* yang dilakukan pada aplikasi absensi PT. Graha Usaha Teknik menghasilkan tujuh servis yang dapat dikembangkan pada arsitektur *microservice*. Berdasarkan hasil pengujian dengan menggunakan *load test*, arsitektur *microservice* yang telah dibangun lebih optimal dibandingkan arsitektur *monolithic* pada saat jumlah pengguna dinaikan menjadi 15 tps dengan menggunakan spesifikasi komputer yang sama.

1. PENDAHULUAN

Pesatnya perkembangan teknologi pada zaman ini telah memberikan banyak manfaat di berbagai bidang. Salah satu manfaat yang diberikan oleh pesatnya perkembangan teknologi adalah membantu manusia dalam menyelesaikan berbagai macam pekerjaannya. Pesatnya perkembangan teknologi juga didukung oleh peningkatan penggunaan teknologi di kalangan masyarakat. Salah satu contoh peningkatan penggunaan teknologi adalah internet. Meningkatnya penggunaan internet dibuktikan dengan adanya survei yang dilakukan Asosiasi Penyelenggara Jasa Internet Indonesia (APJII). Menurut APJII selama kurun waktu 18 tahun (2000-2017) pengguna internet di Indonesia meningkat dari 1,9 juta menjadi 143,26 juta pengguna [1]. Faktor meningkatnya penggunaan internet disebabkan oleh peningkatan penggunaan teknologi smartphone di kalangan masyarakat. Berdasarkan survei APJII, pada tahun 2016 sebanyak 45,4% pengguna hanya memiliki satu perangkat smartphone [2], sedangkan pada tahun 2017, pengguna yang memiliki satu perangkat smartphone meningkat menjadi 95,75% [1]. Meningkatnya penggunaan teknologi *smartphone* dapat mempengaruhi kinerja sebuah aplikasi yang telah dibangun. Sebagian aplikasi yang telah dirancang dan dibangun saat ini masih menggunakan arsitektur *monolithic*.

Arsitektur *monolithic* adalah suatu arsitektur yang mengambarkan sebuah aplikasi yang menjalankan semua logika dalam satu *server* aplikasi [3]. Peningkatan jumlah pengguna aplikasi dapat mempengaruhi proses pemeliharaan aplikasi, kinerja aplikasi dan kompleksnya proses pembaruan aplikasi yang dibangun dengan arsitektur *monolithic* [3]. Proses pemeliharaan aplikasi yang dibangun dengan arsitektur *monolithic* yang berukuran besar sangat sulit dikarenakan kompleksitasnya [4]. Salah satu kompleksitas yang sering terjadi pada arsitektur *monolithic* adalah *resilient challenges*, dimana jika terjadi kegagalan pada saat proses pembaruan atau penambahan fitur baru pada aplikasi, maka seluruh fitur aplikasi akan mengalami kegagalan sistem [5]. Di dalam penelitian [4] juga dijelaskan jika terjadi perubahan salah satu

https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

modul aplikasi dengan arsitektur *monolithic*, maka dibutuhkan proses *restart* seluruh aplikasi. Semakin besar sebuah aplikasi yang dibangun dengan arsitektur *monolithic*, maka proses *restart* aplikasi menjadi semakin lama sehingga selama proses *restart* berlangsung, aplikasi tersebut tidak dapat digunakan.

Dengan adanya permasalahan sebagaimana yang dijelaskan diatas, maka muncul sebuah arsitektur baru, yaitu arsitektur *microservice*. Arsitektur *microservice* adalah gaya arsitektur perangkat lunak yang memerlukan pemecahan aplikasi secara bisnis [6]. Pengembangan aplikasi dengan menggunakan arsitektur *microservice* dapat memperbaiki permasalahan diatas, yaitu adanya pemisahan servis-servis menjadi kecil.

Aplikasi absensi PT. Graha Usaha Teknik merupakan sebuah aplikasi berbasis web yang dikembangkan untuk memenuhi kebutuhan absensi karyawan pada PT. Graha Usaha Teknik. Terdapat beberapa fungsi dari aplikasi absensi PT. Graha Usaha Teknik diantaranya adalah memudahkan karyawan dalam melakukan absensi, untuk memantau kedisplinan dari masing-masing karyawan, memudahkan pembuatan laporan data karyawan, jam kerja karyawan dan peminjaman APD (alat pelindung diri) dari masing-masing karyawan. Sebagai aplikasi yang berfungsi untuk menyimpan berbagai macam data karyawan, aplikasi absensi PT. Graha Usaha Teknik masih mengadopsi arsitektur *monolithic*.

Seiring dengan bertambahnya karyawan dan berkembanganya PT. Graha Usaha Teknik, maka fitur yang dikembangkan semakin bertambah dan jumlah trafik yang diproses oleh aplikasi semakin besar sehingga proses pemeliharaan aplikasi menjadi semakin sulit, kinerja aplikasi semakin menurun, dan proses pembaruan yang semakin sulit. Untuk menangani permasalahan tersebut, maka dilakukan *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada aplikasi absensi pada PT. Graha Usaha Teknik. Untuk melakukan proses *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada PT. Graha Usaha Teknik digunakan 13 tahapan yang diusulkan oleh penelitian [7] dan strategi *strangler pattern* sehingga dapat mengurangi kegagalan sistem pada saat proses *refactoring*.

1.1. Arsitektur Monolithic

Arsitektur *monolithic* adalah suatu arsitektur yang mengambarkan sebuah aplikasi yang menjalankan semua logika dalam satu *server* aplikasi [3]. Aplikasi yang dibangun dengan arsitektur *monolithic* hanya dijalankan dengan menggunakan satu *server* aplikasi sehingga hanya memerlukan proses pemeliharaan aplikasi pada satu server aplikasi. Di dalam penelitian [4], dijelaskan bahwa arsitektur *monolithic* merupakan aplikasi perangkat lunak yang modulnya tidak dapat dijalankan secara independen. Dengan menggunakan arsitektur *monolithic*, setiap aplikasi dijalankan secara bersamaan dikarenakan seluruh modul-modul aplikasi terdapat di dalam sebuah aplikasi yang sangat besar. Terdapat beberapa kekurangan dari arsitektur *monolitic*, yaitu [4] :

- Berukuran besar sehingga sangat sulit untuk melakukan pemeliharaan dan pengembangan aplikasi.
- Dapat terjadi konflik *dependency library* saat proses penambahan modul baru.
- Setiap penambahan modul baru membutuhkan proses *restart* aplikasi.
- Munculnya konflik kode aplikasi pada saat dilakukan *deployment* aplikasi secara bersamaan.
- Arsitektur *monolithic* memiliki keterbatasan untuk dilakukan *scalability*.
- Hanya menggunakan satu teknologi untuk membuat aplikasi dengan menggunakan arsitektur *monolithic*.

1.2. Arsitektur Microservice

Arsitektur *microservice* adalah gaya arsitektur perangkat lunak yang memerlukan pemecahan aplikasi secara bisnis [6], sedangkan di dalam penelitian [8] dijelaskan bahwa arsitektur *microservice* adalah sebuah arsitektur baru dimana pengembangan aplikasi dilakukan dalam bentuk web *service* kecil yang saling berkomunikasi satu sama lain. Terdapat beberapa kelebihan dari arsitektur *microservice*, yaitu [4] :

- Arsitektur *microservice* membuat kode aplikasi lebih sedikit dan bersifat independen sehingga dapat dilakukan pengujian aplikasi secara independent.
- Memudahkan pemeliharaan perangkat lunak.
- Dapat melakukan proses distrubusi perangkat lunak secara independen.
- Mudahnya dalam melakukan *scalability*.
- Developer dapat bebas dalam mengembangkan aplikasi dengan berbagai bahasa pemrograman dan *framework*.

1.3. Strangler Pattern

Strangler pattern adalah strategi yang digunakan untuk melakukan *refactoring* arsitektur *monolithic* menjadi arsitektur microservice [9]. *Strangler pattern* merupakan strategi yang dikembangkan oleh Martin Fowler dan digunakan untuk melakukan *refactoring* arsitektur *monolithic* [10]. Di dalam *strangler pattern* tersebut, Martin Fowler menggambarkan *strangler pattern* dengan cara membangun ulang sistem yang sedang berjalan secara bertahap tanpa menghapus sistem yang lama.

Membangun ulang seluruh sistem dari sebuah sistem yang telah berjalan memliki banyak risiko, misalnya resiko kegagalan pada saat melakukan migrasi sistem dan waktu migrasi sistem yang lama. Dengan menggunakan strategi *strangler pattern,* maka dapat mengurangi resiko-resiko kegagalan sistem [10]. Dengan menggunakan strategi *strangler pattern*, proses *refactoring* dapat dilakukan secara bertahap sehingga tidak akan menggangu aplikasi yang sedang berjalan.

1.4. Refactoring Microservice

Proses refactoring dari arsitektur *monolithic* menjadi arsitektur *microservice* dilakukan dengan menggunakan 15 tahapan, yaitu [7] :

1. Penambahan konfigurasi *continuous integration.*
2. Analisa arsitektur yang sedang berjalan.

 https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

3. Pemecahan arsitektur *monolithic* berdasarkan proses bisnis.
4. Pemecahan arsitektur *monolithic* berdasarkan kepemilikan data.
5. Perubahan pemanggilan kode aplikasi dengan pemanggilan servis
6. Penambahan konfigurasi *service discovery server*.
7. Penambahan konfigurasi *service discovery client*.
8. Penambahan konfigurasi internal *load balancer*.
9. Penambahan konfigurasi eksternal *load balancer*.
10. Penambahan konfigurasi *circuit breaker*.
11. Penambahan konfigurasi *server*.
12. Penambahan konfigurasi *edge server*.
13. Pengaturan servis dengan *docker container*.
14. Implementasi servis pada *server cluster*.
15. Pemantauan sistem.

1.5. Refactoring Database

Terdapat 3 pola *database* yang dapat digunakan pada arsitektur *microservice*, yaitu [6] :

- *Private-tables-per-service*, yaitu semua servis menggunakan *database* yang sama, akan tetapi setiap servis hanya dapat mengakses tabel yang telah ditentukan.
- *Schema-per-service*, yaitu semua servis memiliki *database* masing-masing akan tetapi *database* tersebut berada pada satu *node database server*.
- *Database-server-per-service*, yaitu semua servis memiliki *database* masing-masing dan berada pada masing-masing *node database server*.

Untuk melakukan proses *refactoring database* dapat dilakukan dengan dua tahapan, yaitu [11] :

1. Mendeskripsikan skema *database* yang sedang digunakan untuk mengetahui tabel-tabel dan keterhubungan antar tabel pada database yang sedang digunakan.
2. Mengubah skema *database* dan memindahkan data. Pengubahan skema *database* dilakukan dengan mengubah dan membuat tabel-tabel dari database yang sedang digunakan. Skema *database* yang baru dilakukan proses pemindahan data dengan menggunakan perintah SQL.

1.6. Domain Driven Design

Domain Driven Design (DDD) merupakan pengetahuan mengenai proses bisnis dan cara menggunakan sebuah teknologi untuk dapat membantu proses bisnis [12]. DDD dapat membantu dalam proses pengembangan perangkat lunak menjadi lebih akurat sesuai dengan proses bisnis yang akan berjalan pada sebuah aplikasi. Adapun tahapan-tahapan yang digunakan pada metode DDD, yaitu [13] :

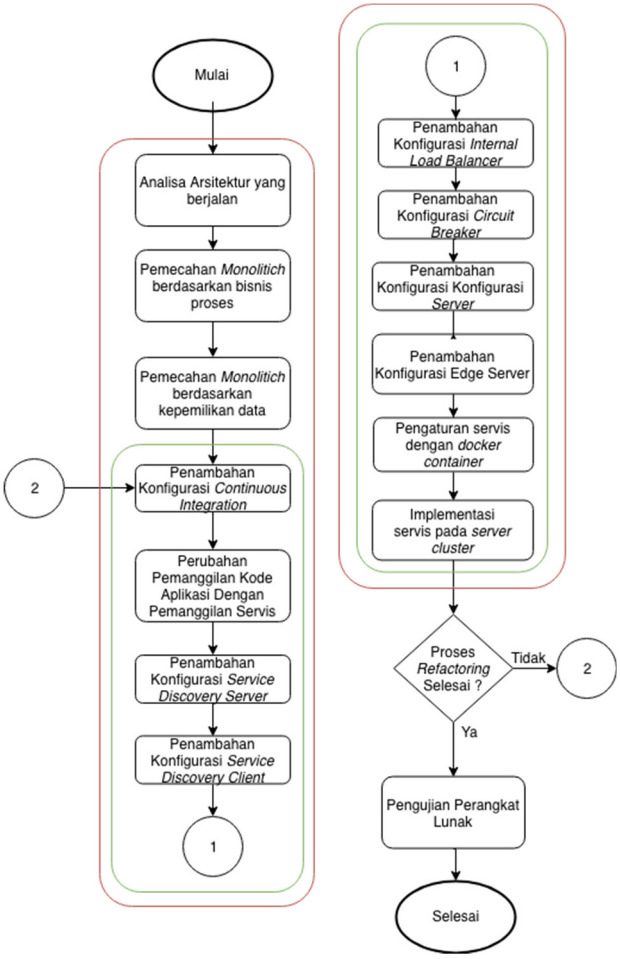
1. Analisa fungsional sistem yang sedang berjalan dilakukan untuk mengetahui proses bisnis yang terdapat pada aplikasi *monolithic*.
2. Pengelompokan fungsional sistem yang sedang berjalan dilakukan untuk mengetahui fungsional-fungsional yang memiliki persamaan antar proses bisnis.
3. Identifikasi hubungan fungsional sistem yang sedang berjalan dilakukan untuk mengetahui keterhubungan antara satu proses bisnis dengan proses bisnis yang lain.

2. METODE

Proses *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada penelitian ini dilakukan dengan menggunakan 2 tahapan, yaitu strategi *strangler pattern* dan 15 tahap seperti pada penelitian [7]. Pada penelitian ini digunakan 13 tahapan untuk melakukan proses *refactoring* dari 15 tahap berdasarkan metode yang diusulkan oleh penelitian [7]. Terdapat 2 tahapan yang tidak digunakan dikarenakan tahapan tersebut tidak dibutuhkan pada saat proses *refactoring* dilakukan, yaitu :

1. Penambahan konfigurasi eksternal *load balancer*.
2. Memantau sistem *microservice* yang telah dilakukan proses *refactoring*.

Setelah dilakukan pemilihan 13 tahapan untuk proses *refactoring*, maka dibuatlah tahapan penelitian yang mencakup kedua tahapan tersebut. Tahapan penelitian yang digunakan dapat dilihat pada gambar 1.



Gambar 1. Tahapan Penelitian

Pada gambar 1, 13 tahapan yang telah dipilih dari 15 tahapan pada penelitian [7] ditandai dengan garis berwarna merah, sedangkan strategi *strangler pattern* ditandai dengan garis berwarna hijau. Strategi *strangler pattern* hanya digunakan pada 10 tahapan *refactoring*, yaitu penambahan konfigurasi *continuous integration*, perubahan pemanggilan kode aplikasi dengan pemanggilan servis, penambahan konfigurasi *service discovery server*, penambahan konfigurasi *service discovery client*, penambahan konfigurasi internal *load balancer*, penambahan konfigurasi *circuit breaker*, penambahan konfigurasi *server*, penambahan konfigurasi *edge server*, pengaturan servis dengan *docker container* dan implementasi servis pada *server cluster*. Tahapan strategi *strangler pattern* yang dilakukan pada penelitian ini, yaitu :

1. Penambahan *server proxy* untuk menghubungkan pengguna dengan aplikasi pada arsitektur *monolithic*. Pada tahap ini, penambahan sebuah *server proxy* diletakkan diantara pengguna aplikasi dengan aplikasi pada arsitektur *monolithic*. Penambahan *server proxy* berfungsi untuk mengarahkan trafik ke servis yang telah berhasil dilakukan proses *refactoring*.
2. Proses *refactoring* dilakukan pada salah satu servis yang terdapat di aplikasi dengan arsitektur *monolithic*. Tahapan proses *refactoring* untuk masing-masing servis dilakukan dengan menerapkan 10 tahap seperti pada penelitian [7].
3. Melakukan pengarahan trafik masuk ke servis yang telah selesai dilakukan *refactoring*. Setelah proses *refactoring* selesai, trafik yang masuk diarahkan ke servis baru, sedangkan servis yang belum dilakukan *refactoring* diarahkan ke aplikasi dengan arsitektur *monolithic*.

Tahapan dari strategi *strangler pattern* dilakukan sampai seluruh servis pada arsitektur *monolithic* berhasil dilakukan *refactoring*. Setelah semua servis berhasil dilakukan *refactoring*, aplikasi dengan arsitektur *monolithic* akan dihapus sehingga semua trafik yang masuk diarahkan ke aplikasi dengan arsitektur *microservice* melalui *server proxy*.

Setelah dilakukan proses *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* berdasarkan strategi *strangler pattern* dan 13 tahapan seperti pada penelitian [7], untuk memastikan perangkat lunak berjalan sesuai dengan harapan, maka diperlukan pengujian perangkat lunak. Pengujian perangkat lunak pada penelitian ini menggunakan pengujian *load test*. Pengujian *load test* dilakukan pada dua aplikasi absensi PT. Graha Usaha Teknik, yaitu aplikasi absensi dengan arsitektur *monolithic* dan aplikasi absensi dengan arsitektur *microservice*.

3. HASIL DAN PEMBAHASAN

3.1. Analisa Arsitektur Yang Sedang Berjalan

Proses analisa arsitektur yang sedang berjalan terdiri dari 3 tahapan, yaitu [7] :

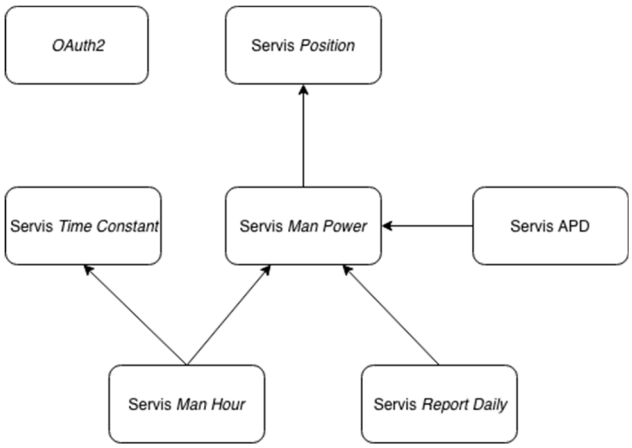
1. Analisa arsitektur servis yang sedang berjalan.
2. Analisa arsitektur teknologi yang sedang berjalan.
3. Analisa arsitektur implementasi aplikasi ke *server*.

3.1.1. Analisa Arsitektur Servis Yang Sedang Berjalan

Proses analisa arsitektur servis dilakukan dengan mengambarkan keterhubungan antar servis yang sedang berjalan. Servis-servis yang terdapat pada aplikasi absensi PT. Graha Usaha Teknik, yaitu :

- Servis *position*, berfungsi sebagai servis untuk mengelola data posisi setiap karyawan.
- Servis *time constant*, berfungsi sebagai servis untuk mengelola data ketentuan atau batasan jam kerja.
- Servis *man power*, berfungsi sebagai servis untuk mengelola data setiap karyawan.
- Servis *man hour*, berfungsi sebagai servis untuk mengelola data jam kerja setiap karyawan.
- Servis *report daily*, berfungsi sebagai servis untuk mengelola data laporan harian yang terjadi di lapangan kerja.
- Servis APD (alat pelindung diri), berfungsi sebagai servis untuk mengelola data APD.
- Servis *OAuth2*, berfungsi sebagai servis untuk pengecekan pengguna saat melakukan proses *login* ke aplikasi.

Untuk mengetahui hubungan antar servis, maka dibuatlah sebuah diagram keterhubungan antar servis yang sedang berjalan. Gambar 2 merupakan keterhubungan antar servis yang sedang berjalan.



Gambar 2. Hubungan antar servis

3.1.2. Analisa arsitektur teknologi yang sedang berjalan

Analisa arsitektur teknologi dilakukan dengan mengidentifikasikan teknologi-teknologi yang digunakan pada sistem yang sedang berjalan. Analisa teknologi dilakukan pada bagian perangkat keras dan perangkat lunak yang digunakan pada sistem yang sedang berjalan. Tabel 1 dan tabel 2 merupakan informasi mengenai teknologi perangkat keras dan perangkat lunak yang digunakan pada sistem yang sedang berjalan.

 https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

Tabel 1. Teknologi perangkat keras

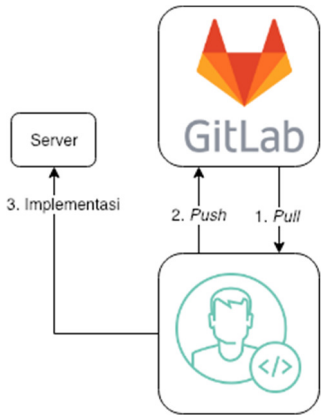
Nama perangkat keras	Spesifikasi
Prosesor	Intel Core i5 2,7 GHz
RAM	8 GB
Disk	500 GB

Tabel 2. Teknologi perangkat lunak

Nama perangkat lunak	Spesifikasi
Bahasa pemrograman	Java
Backend framework	Spring framework
Database	PostgreSQL
Front end framework	Angular JS
UI framework	Bootstrap Material

3.1.3. Analisa arsitektur implementasi aplikasi ke server

Pada tahapan ini, analisa dilakukan untuk mengetahui tahapan-tahapan yang digunakan pada saat proses pengembangan hingga implementasi aplikasi ke *server*. Tahapan pengembangan dan implementasi aplikasi ke *server* dapat dilihat pada gambar 3.



Gambar 3. Tahapan pengembangan dan implementasi aplikasi ke *server*

Adapun tahapan-tahapan yang dilakukan berdasarkan gambar 3, yaitu :

1. Melakukan *pull* kode aplikasi dari tempat penyimpanan. Tempat penyimpanan kode yang digunakan yaitu *gitlab*.
2. Melakukan penambahan, perubahan dan *push* kode aplikasi ke tempat penyimpanan kode.
3. Melakukan pengujian aplikasi, pembuatan artifak dan implementasi aplikasi ke *server*.

3.2. Pemecahan Arsitektur Monolithic Berdasarkan Proses Bisnis

Proses pemecahan arsitektur *monolithic* berdasarkan proses bisnis dilakukan menggunakan metode DDD dengan tiga tahapan, yaitu :

1. Analisa fungsional sistem yang sedang berjalan.
2. Pengelompokan fungsional sistem yang sedang berjalan
3. Identifikasi hubungan fungsional sistem yang sedang berjalan.

3.2.1. Analisa Fungsional Sistem Yang Sedang Berjalan

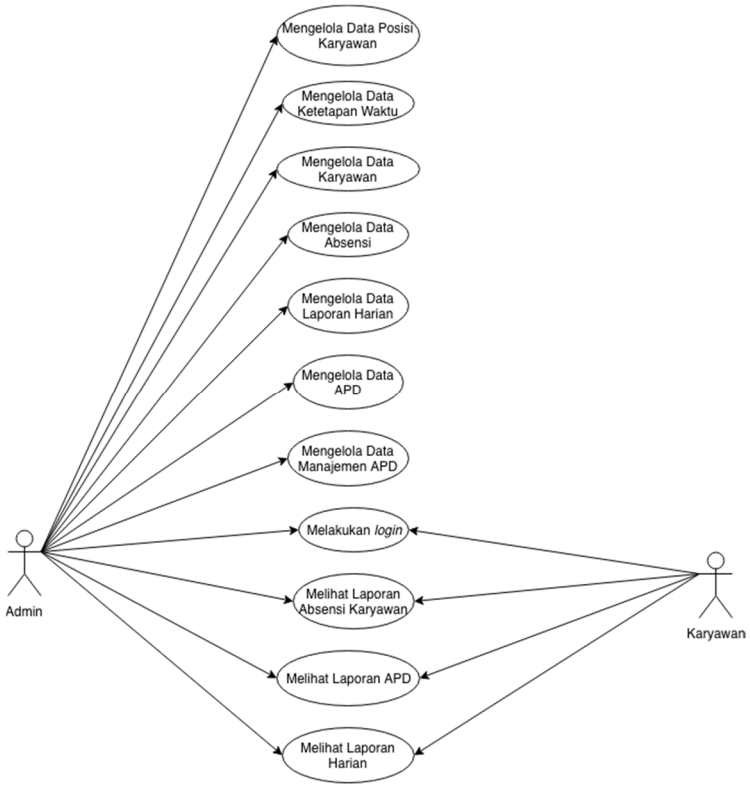
Proses analisa fungsional sistem dilakukan dengan mengidentifikasikan setiap proses bisnis yang terdapat pada aplikasi *monolithic*. Proses analisa fungsional dilakukan dengan menggunakan permodelan *use case diagram*. Gambar 4 merupakan pemodelan *use diagram* aplikasi absensi PT. Graha Usaha Teknik.

3.2.2. Pengelompokan Fungsional Sistem Yang Sedang Berjalan

Pengelompokkan fungsional sistem dilakukan dengan mengelompokkan fungsional berdasarkan fungsional yang sejenis. Pengelompokan fungsional yang terdapat pada aplikasi absensi PT. Graha Usaha Teknik, yaitu proses bisnis posisi karyawan, ketetapan waktu, karyawan, absensi, laporan harian dan APD. Tabel 3 merupakan informasi mengenai pengelompokkan dan pemetaan fungsional pada aplikasi absensi PT. Graha Usaha Teknik.

Tabel 3. Pengelompokan dan pemetaan fungsional aplikasi absensi PT. Graha Usaha Teknik

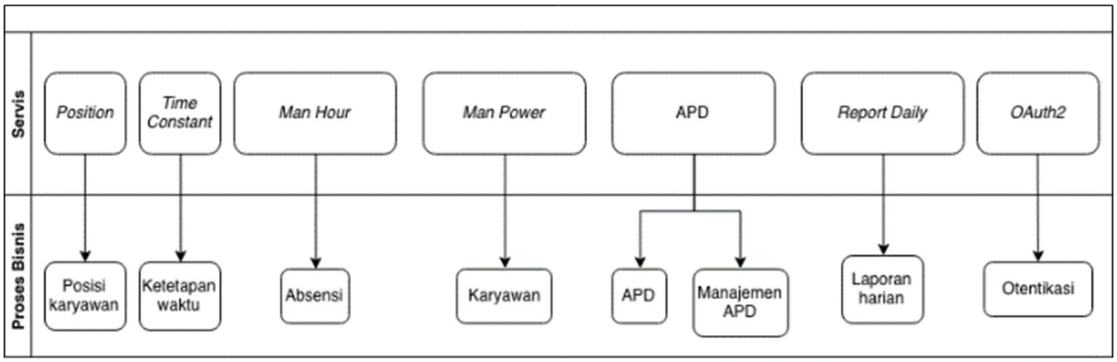
Kelompok	Fungsional
Proses bisnis posisi karyawan	Mengelola data posisi karyawan
Proses bisnis ketetapan waktu	Mengelola data ketetapan waktu
Proses bisnis karyawan	Mengelola data karyawan
Proses bisnis absensi	Mengelola data absensi
	Melihat laporan absensi karyawan
Proses bisnis laporan harian	Mengelola data laporan harian
	Melihat laporan harian
Proses bisnis APD	Mengelola data APD
	Mengelola data manajemen APD
	Melihat laporan APD
Proses bisnis otentikasi	Melakukan *login*



Gambar 4. *Use case diagram* aplikasi absensi PT. Graha Usaha Teknik

3.2.3. Identifikasi Hubungan Fungsional Sistem Yang Sedang Berjalan

Proses identifikasi pada tahap ini berkaitan dengan proses analisa arsitektur servis yang sedang berjalan. Masing-masing servis dapat mewakili satu atau lebih proses bisnis, sehingga dilakukan proses identifikasi fungsional berdasarkan hubungan servis yang dilakukan pada proses analisa arsitektur servis. Berdasarkan proses identifikasi hubungan fungsional, dihasilkan 7 proses bisnis yang telah berhasil dipecah. Masing-masing proses bisnis yang telah dipecah mewakili satu servis kecuali proses bisnis APD dan manajemen APD. Proses bisnis APD dan manajemen APD masih memiliki proses bisnis yang sejenis sehingga kedua proses bisnis tersebut digabungkan menjadi satu servis, yaitu servis APD. Gambar 5 merupakan hasil pemecahan arsitektur *monolithic* berdasarkan proses bisnis.



Gambar 5. Hasil pemecahan arsitektur *monolithic* berdasarkan proses bisnis

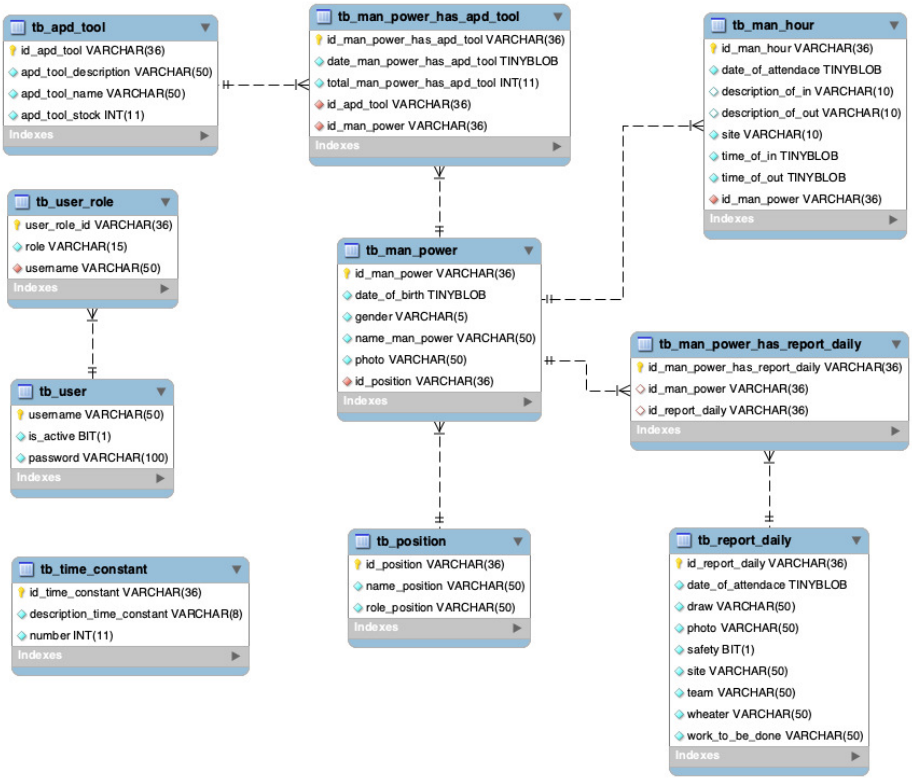
3.3. Pemecahan Arsitektur Monolithic Berdasarkan Kepemilikan Data

Proses pemecahan arsitektur *monolithic* dilakukan dengan mendefinisikan terlebih dahulu tabel-tabel yang terdapat pada *database* yang sedang digunakan. Gambar 6 merupakan struktur tabel yang digunakan pada aplikasi absensi PT. Graha Usaha Teknik.

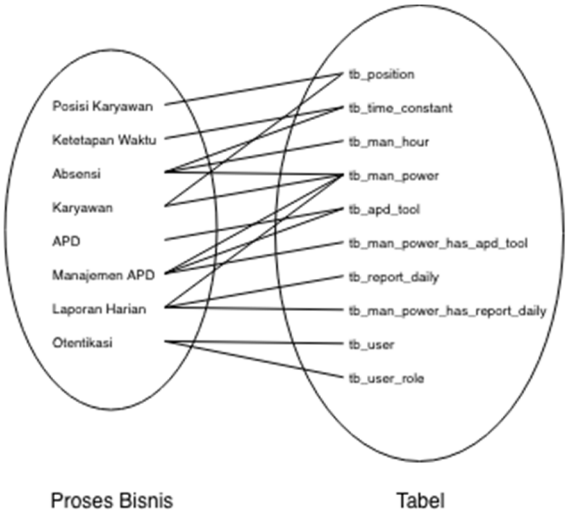
Dari tabel-tabel pada gambar 6 dilakukan proses pemetaan dengan menggunakan diagram pemetaan. Proses pemetaan dilakukan berdasarkan kepemilikan data oleh sebuah proses bisnis, dimana

 https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

setiap proses bisnis bisa memiliki lebih dari satu kepemilikan data.
Gambar 7 merupakan diagram pemetaan tabel dan proses bisnis.



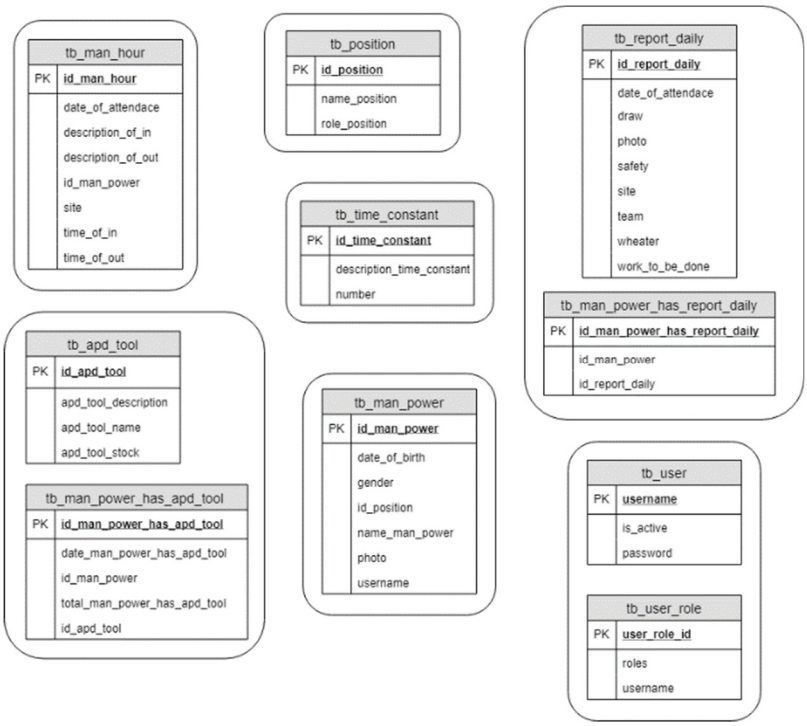
Gambar 6. Struktur tabel pada aplikasi absensi PT. Graha Usaha Teknik



Gambar 7. Diagram pemetaan tabel dan proses bisnis

Pada penelitian ini, *schema-per-service* digunakan sebagai pola *database* untuk pengembangan arsitektur *microservice* pada PT. Graha Usaha Teknik. Penggunaan pola *schema-per-service* dikarenakan untuk mengurangi penggunaan *resource* yang berlebihan, sehingga hanya diperlukan satu *server* untuk menampung beberapa *database* yang diperlukan oleh masing-masing servis. Selanjutnya dilakukan *refactoring database* dengan menggunakan metode yang diusulkan pada penelitian [11]. Tahapan proses mendeskripsikan skema *database* telah dijelaskan pada gambar 6. Tahapan selanjutnya untuk melakukan *refactoring database*, yaitu mengubah skema *database* dan memindahkan data. Proses pengubahan skema *database* pada penelitian ini menggunakan tabel yang telah diperoleh dari proses pemecahan

arsitektur *monolithic* berdasarkan kepemilikan data. Gambar 8 merupakan struktur dan pengelompokkan tabel yang telah dilakukan proses *refactoring*.



Gambar 8. Struktur dan pengelompokan tabel yang telah dilakukan *refactoring*

Setelah melakukan proses mengubah skema *database*, tahapan selanjutnya memindahkan data dari skema *database* yang lama ke skema *database* yang baru. Proses pemindahan data tersebut menggunakan perintah SQL.

3.4. Penambahan Konfigurasi Continuous Integration

Sebelum melakukan penambahan konfigurasi *continuous integration*, terlebih dahulu dilakukan proses pembuatan tempat penyimpanan kode aplikasi untuk masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik. Proses penambahan konfigurasi *continuous integration* dilakukan pada masing-masing servis aplikasi absensi PT. Graha Usaha Teknik untuk kebutuhan otomatisasi proses pembuatan artifak dan pengujian aplikasi tersebut. Konfigurasi *continuous integration* yang digunakan pada penelitian ini menggunakan *gitlab runner* sehingga hanya dibutuhkan konfigurasi secara kode program.

3.5. Perubahan Pemanggilan Kode Aplikasi Dengan Pemanggilan Servis

Perubahan pemanggilan kode aplikasi dengan pemanggilan servis dilakukan jika terdapat ketergantungan data antar servis pada aplikasi absensi PT. Graha Usaha Teknik. Pada aplikasi absensi PT. Graha Usaha Teknik yang telah dipecah terdapat empat proses bisnis yang memerlukan pemanggilan servis, yaitu proses bisnis absensi, karyawan, manajemen APD dan laporan harian. Proses pemanggilan servis melalui keempat proses bisnis tersebut dilakukan melalui API (*application programming interface*).

3.6. Penambahan Konfigurasi Service Discovery Server

Proses penambahan konfigurasi *service discovery server* berfungsi untuk mendaftarkan seluruh servis-servis pada aplikasi absensi PT. Graha Usaha Teknik yang telah dipecah. Penambahan konfigurasi *service discovery server* dilakukan dengan mengembangkan sebuah aplikasi *service discovery server*. Pada penelitian ini, *spring cloud eureka server* digunakan sebagai *framework* untuk mengembangkan aplikasi *service discovery server.*

3.7. Penambahan Konfigurasi Service Discovery Client

Untuk mengetahui status dari masing-masing servis, maka dilakukan penambahan konfigurasi *service discovery client* untuk masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik. Proses penambahan konfigurasi *service discovery client* bertujuan agar masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik dapat mengirimkan status servis ke *service discovery server* sehingga mempermudah pemantauan servis. Pada penelitian ini, *spring cloud eureka client* digunakan sebagai *framework* untuk menambahkan konfigurasi *service discovery client* pada masing-masing servis.

 https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

3.8. Penambahan Konfigurasi Internal Load Balancer

Penambahan konfigurasi internal *load balancer* yang dilakukan untuk masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik berfungsi jika sewaktu-waktu terjadi penambahan lebih dari satu instance servis. Penambahan konfigurasi hanya dilakukan pada tiga servis, yaitu servis *man hour*, apd dan *report daily* dikarenakan ketiga servis tersebut membutuhkan pemanggilan servis lainnya. Pada penelitian ini, teknologi ribbon digunakan sebagai konfigurasi internal *load balancer*.

3.9. Penambahan Konfigurasi Circuit Breaker

Adanya proses penambahan konfigurasi *circuit breaker* untuk beberapa servis pada aplikasi absensi PT. Graha Usaha Teknik disebabkan terdapat beberapa proses pemanggilan servis. Untuk mengurangi kegagalan sistem jika servis yang akan diakses dalam keadaan mati, maka dilakukan penambahan konfigurasi *circuit breaker* pada servis-servis yang melakukan pemanggilan servis lainnya. Penambahan konfigurasi *circuit breaker* hanya dilakukan pada empat servis, yaitu servis *man hour, man power*, apd dan *report daily*.

3.10. Penambahan Konfigurasi Server

Masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik memiliki konfigurasi yang berbeda sehingga dibutuhkan sebuah aplikasi yang berfungsi untuk menyimpan seluruh konfigurasi servis. Untuk menyimpan seluruh konfigurasi aplikasi, maka dilakukan proses penambahan konfigurasi *server* dengan mengembangkan sebuah aplikasi yang berfungsi sebagai konfigurasi *server*. Pada penelitian ini, pengembangan aplikasi yang berfungsi sebagai konfigurasi *server* menggunakan *framework spring cloud config server*.

3.11. Penambahan Konfigurasi Edge Server

Untuk mempermudah dalam memantau status dan jumlah penggunaan dari masing-masing servis, maka dilakukan penambahan konfigurasi *edge server* pada aplikasi absensi PT. Graha Usaha Teknik. Penambahan konfigurasi *edge server* pada penelitian ini dilakukan dengan mengembangkan aplikasi yang berfungsi sebagai *edge server*. Proses pengembangan aplikasi *edge server* pada penelitian ini menggunakan *spring cloud zuul*.

3.12. Pengaturan Servis Dengan Docker Container

Proses penganturan servis dengan *docker container* dilakukan untuk masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik. Penggunaan *docker container* pada masing-masing servis pada aplikasi absensi PT. Graha Usaha Teknik berfungsi untuk menghemat penggunaan *resource* secara berlebihan pada *server*. Pada penelitian ini, dilakukan tiga tahapan proses pengaturan servis dengan menggunakan *docker container*, yaitu :

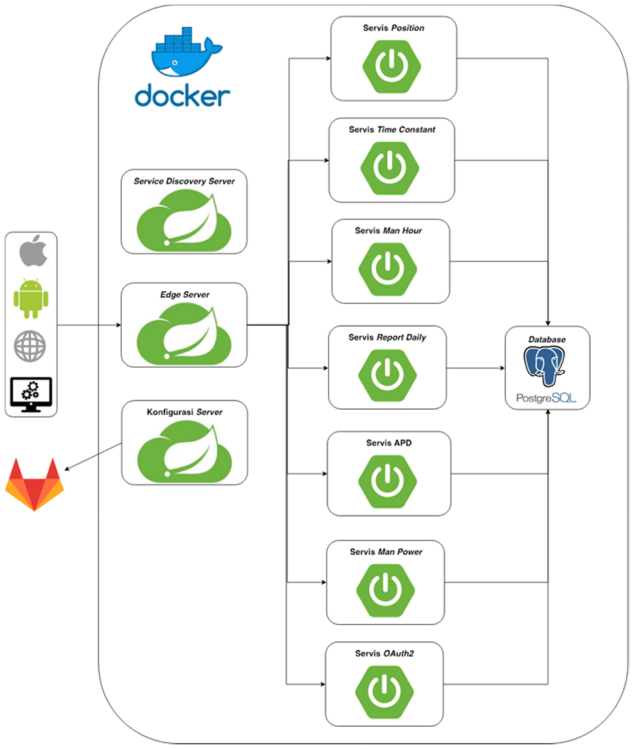
1. Pembuatan artifak untuk masing-masing servis. Proses pembuatan artifak servis pada penelitian ini menggunakan gradle. Masing-masing servis dibuatkan konfigurasi gradle yang berfungsi untuk membuat artifak dalam bentuk artifak jar.
2. Membuat konfigurasi *docker image*. Proses pembuatan konfigurasi *docker image* dilakukan dengan menambahkan konfigurasi *docker image* berbentuk kode aplikasi pada *file* Dockerfile untuk masing-masing servis.
3. Membuat dan menggugah docker image. Proses membuat *docker image* dilakukan dengan menggunakan perintah *docker*. Setelah dilakukan pembuatan *docker image*, lalu dilakukan proses mengunggah *docker image* ke sebuah tempat penyimpanan *docker image*, yaitu *docker hub*.

3.13. Implementasi Servis Pada Server Cluster

Implementasi servis pada *server cluster* dilakukan dengan menjalan servis-servis aplikasi absensi PT. Graha Usaha Teknik melalui *docker container* di dua *server* yang berbeda. Proses implementasi aplikasi absensi PT. Graha Usaha Teknik pada *server cluster* berfungsi untuk mengurangi kegagalan sistem aplikasi jika salah satu *server* dalam keadaan mati. Pada penelitian ini, implementasi servis pada *server cluster* dilakukan dengan menggunakan teknologi *docker swarm*. Proses implementasi servis pada *docker swarm* dilakukan dengan dua tahapan, yaitu :

1. Melakukan konfigurasi *docker swarm* pada masing-masing servis. Proses pembuatan konfigurasi *docker swarm* dilakukan dengan menambahkan kode konfigurasi *docker swarm* pada file docker-compose-swarm.yml.
2. Tahap terakhir dilakukan dengan menjalankan konfigurasi *docker swarm* dengan perintah *docker swarm*.

Berdasarkan hasil *refactoring* aplikasi absensi PT. Graha Usaha Teknik dari arsitektur *monolithic* menjadi arsitektur *microservice* terdapat komponen-komponen yang harus dikembangkan pada arsitektur *microservice*. Komponen-komponen yang dikembangkan untuk mendukung arsitektur *microservice* diantaranya adalah *service discovery server, load balancer, circuit breaker,* konfigurasi *server* dan *edge server*. Pada penelitian ini, telah berhasil dilakukan proses identifikasi sebanyak tujuh servis dari arsitektur *monolithic* pada aplikasi absensi PT. Graha Usaha Teknik dengan menggunakan 13 tahapan seperti pada penelitian [7] dan strategi *strangler pattern*. Berdasarkan hasil *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada aplikasi absensi PT. Graha Usaha Teknik, maka dihasilkan sebuah arsitektur *microservice* pada aplikasi absensi PT. Graha Usaha Teknik seperti gambar 9.



Gambar 9. Arsitektur *microservice* pada aplikasi absensi PT. Graha Usaha Teknik

3.14. Pengujian Perangkat Lunak

Pada penelitian ini, pengujian perangkat lunak dilakukan berdasarkan langkah-langkah yang diusulkan pada penelitian [17].

3.14.1. Perencanaan Pengujian Perangkat Lunak.

Perencanaan pengujian perangkat lunak dilakukan dengan mengidentifikasikan kebutuhan untuk melakukan pengujian perangkat lunak *load test*. Tabel merupakan informasi mengenai kebutuhan perangkat lunak *load test*. Tabel 4 merupakan informasi mengenai kebutuhan pengujian perangkat lunak dengan *load test*.

3.14.2. Analisis Dan Perancangan Pengujian Perangkat Lunak

Perancangan pengujian perangkat lunak *load test* dilakukan dengan menentukan beban yang sanggup ditangani oleh sebuah aplikasi. Beratnya beban yang sanggup ditangani oleh sebuah aplikasi direpresentasikan dengan banyaknya pengguna mengakses sebuah aplikasi dalam waktu tertentu. Pada penelitian ini, waktu yang digunakan untuk menentukan banyaknya pengguna, yaitu dalam satuan detik atau dalam istilah lain disebut sebagai tps (*transactions per second*). Tabel 5 merupakan skenario *load test* yang dilakukan pada aplikasi absensi PT. Graha Usaha Teknik yang berarsitektur *monolithic* dan *microservice*.

Tabel 4. Kebutuhan pengujian perangkat lunak dengan *load test*

Kebutuhan	Deskripsi
Aplikasi yang dilakukan pengujian	Aplikasi yang diuji adalah tujuh servis yang telah dipecah
Sistem Operasi	Ubuntu 18.04 LTS
Database	*Embedded* postgreSQL *server*
Framework test	*Gatling load and performance testing*
Spefisikasi komputer yang menjalankan aplikasi absensi	• Prosesor AMD Ryzen 7 2700X 3,70 GHz • RAM 8 GB • Disk 500 GB
Spesifikasi komputer yang menjalankan aplikasi pengujian	• Prosesor Intel Core i5 2,70 GHz • RAM 8 GB • Disk 250 GB

Tabel 5. Skenario pengujian perangkat lunak dengan *load test*

Nama Arsitektur	Jumlah Akses	Lama Pengujian
Monolithic	5 tps	10 menit
	10 tps	10 menit
	15 tps	10 menit
Microservice	5 tps	10 menit
	10 tps	10 menit
	15 tps	10 menit

3.14.3. Implementasi Pengujian Perangkat Lunak

Proses implementasi pengujian perangkat lunak dilakukan dengan membuat dan menjalankan *load test* pada tujuh servis yang telah dipecah. Proses pembuatan *load test* dilakukan dengan membuat kode pengujian aplikasi.

3.14.4. Menganalisis Dan Mengevaluasi Hasil Pengujian Perangkat Lunak

Hasil pengujian *load test* dapat berupa laporan jumlah sukses, gagal dan *response time*. Laporan pengujian tersebut ditulis ke sebuah tabel untuk membandingkan *load test* arsitektur *monolithic* dengan *microservice* pada aplikasi absensi PT. Graha Usaha Teknik. Tabel 6 merupakan hasil pengujian perangkat lunak dengan *load test*.

https://doi.org/10.25077/ TEKNOSI.v5i1.2019.57-68

Tabel 6. Hasil pengujian perangkat lunak dengan *load test*

Nama Arsitektur	Jumlah Akses	Jumlah *Request*	*Response Time*			
			t < 800 ms	800 ms < t < 1200 ms	t > 1200 ms	Gagal
Monolithic	5 tps	21000	20918	38	16	4
Microservice	5 tps	21000	20973	13	14	0
Monolithic	10 tps	42000	41921	41	10	4
Microservice	10 tps	42000	41973	12	15	0
Monolithic	15 tps	63000	62443	232	297	4
Microservice	15 tps	63000	62958	23	19	0

Berdasarkan tabel 6, pengujian perangkat lunak dengan *load test* dilakukan menggunakan tiga skenario, yaitu 5, 10 dan 15 tps untuk masing-masing arsitektur. Dari tabel 6, dapat disimpulkan bahwa pada saat pengujian untuk skenario 5 tps, arsitektur *microservice* mengembalikan *response time* dibawah 800 ms lebih banyak dari pada arsitektur *monolitich*, yaitu sebanyak 20973 *request*. Dari jumlah *request* yang ditangani oleh arsitektur *microservice*, hal ini membuktikan bahwa arsitektur *microservice* yang telah dibangun dapat menangani beban hingga 5 tps meskipun terdapat beberapa *response time* diatas satu detik. Pada skenario 10 tps, arsitektur *microservice* yang telah dibangun juga dapat menangani beban hingga 10 tps, hal ini dibuktikan dengan jumlah *request* yang diproses oleh arsitektur *microservice* lebih banyak dari pada arsitektur *monolitich*. Pada skenario yang terakhir, jumlah *request* yang diproses oleh arsitektur *monolitich* lebih rendah dari pada arsitektur *microservice*. Arsitektur *monolitich* dan *microservice* dapat menangani beban hingga 15 tps, akan tetapi arsitektur *monolitich* mengembalikan *response* time dibawah 800 ms dan diatas setengah detik lebih banyak dari pada arsitektur *microservice*. Hasil dari pengujian untuk skenario 15 tps, dapat disimpulkan bahwa semakin banyak *request* yang diproses oleh arsitektur *monolitich*, maka semakin sedikit jumlah *response time* yang dibawah 800 ms. Dengan skenario yang sama, arsitektur *microservice* dapat mengembalikan *response time* dibawah 800 ms lebih banyak dibandingkan arsitektur *monolitich*.

Berdasarkan hasil pengujian perangkat lunak menggunakan *load test* pada arsitektur *microservice* dan *monolithic* dapat disimpulkan bahwa arsitektur *microservice* yang telah dibangun dapat menangani beban yang telah ditentukan, yaitu sebesar 15 tps dan menggunakan spesifikasi komputer yang telah ditentukan. Pada pengujian perangkat lunak dengan menggunakan *load test*, arsitektur *microservice* dapat mengungguli arsitektur *monolithic* dalam mengembalikan *response time*. Dengan menggunakan spesifikasi komputer yang sama, arsitektur *microservice* dapat lebih optimal pada saat jumlah pengguna dinaikan menjadi 15 tps.

4. KESIMPULAN

Berdasarkan hasil penelitian dapat disimpulkan bahwa melakukan *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* pada aplikasi absensi PT. Graha Usaha Teknik dapat dilakukan dengan menggunakan strategi *strangler pattern* dan 13 tahapan *refactoring* seperti pada penelitian [7]. Proses *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* dilakukan sesuai dengan 13 tahapan seperti pada penelitian [7] dan strategi *strangler pattern* hanya digunakan pada 10 tahapan. Berdasarkan hasil *refactoring* dengan menggunakan 13 tahapan seperti pada penelitian [7] dan strategi *strangler pattern*, terdapat tujuh servis yang berhasil diidentifikasikan, yaitu servis *position, time constant, man hour, report daily*, APD, *man power* dan *OAuth2*. Pada pengujian dengan menggunakan *load test* pada arsitektur *microservice* dan *monolithic*, arsitektur *microservice* yang telah dibangun dapat menangani beban yang telah ditentukan, yaitu sebesar 15 tps dan menggunakan spesifikasi komputer yang telah ditentukan. Berdasarkan hasil pengujian perangkat lunak dengan menggunakan *load test*, arsitektur *microservice* dapat mengungguli arsitektur *monolithic* dalam mengembalikan *response time*. Dengan menggunakan spesifikasi komputer yang sama, arsitektur *microservice* dapat lebih optimal dibandingkan arsitektur *monolithic* pada saat jumlah pengguna dinaikan menjadi 15 tps.

Metode *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* yang diusulkan pada penelitian ini sebaiknya dimodifikasi dengan menggunakan berbagai bahasa pemrograman dalam mengembangkan arsitektur *microservice*. Pada penelitian selanjutnya, metode *refactoring* arsitektur *monolithic* menjadi arsitektur *microservice* yang diusulkan pada penelitian ini dapat digunakan sebagai acuan untuk melakukan proses *refactoring* pada sistem *monolithic* yang lebih besar seperti sistem ERP (*enterprise resource planning*), aplikasi *enterprise* berbasis *Java* EE *monolithic* dan sistem ESB (*enterprise service bus*).

DAFTAR PUSTAKA

[1] APJII, "Infografis penetrasi dan perilaku pengguna internet indonesia survey 2017" tech. rep., Asosiasi Penyelenggara Jasa Internet Indonesia, 2017. Available: https://web.kominfo.go.id/sites/default/files/Laporan%20Survei%20APJII_2017_v1.3.pdf [January 15, 2019].

[2] APJII, "Infografis penetrasi dan perilaku pengguna internet indonesia survey 2016" tech. rep., Asosiasi Penyelenggara Jasa Internet Indonesia, 2016. Available: https://apjii.or.id/downfile/downloadsurvei/infografis_apjii.pdf [January 15, 2019].

[3] S. Daya et al. (2015, August). Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach. (1st edition). [On-Line]. Available: http://www.redbooks.ibm.com/abstracts/sg248275.html [January 15, 2019].

[4] N. Dragoni et al. (2017, September 6). Microservices: Yesterday, Today, and Tomorrow. [On-Line]. Available: https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12 [January 15, 2019].

[5] K. Katuwal. "Microservices: A Flexible Architecture for the Digital Age Version 1.0". *American Journal of Computer Science and Engineering*, Volume 3, September 2016, Pages 20-24.

[6] A. Messina, R. Rizzo, P. Storniolo, and A. Urso. "A Simplified Database Pattern for the Microservice Architecture" In Conference: DBKDA, 2016.

[7] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn. "Microservices Migration Pattern". *Software: Practice and Experience*. Volume 48, Juli 2018, https://doi.org/10.1002/spe.2608.

[8] T. Ueda, T. Nakaike and M. Ohara. "Workload characterization for microservices" in 2016 IEEE International Symposium on Workload Characterization (IISWC), 2016, pp. 1-10.

[9] S. De Santis. L. Florez, D. V Nguyen and E. Rosa, (2016, December). Evolve the Monolith to Microservices with Java and Node. (1st edition). [On-Line]. Available: http://www.redbooks.ibm.com/abstracts/sg248358.html [January 15, 2019].

[10] K. Finnigan. (2018, October). Enterprise Java microservices. (1st edition). [On-Line]. Available: https://www.oreilly.com/library/view/enterprise-java-microservices/9781617294242 [January 15, 2019].

[11] A. D'Sousa and S. Bhatia. "Refactoring of a Database". arXiv preprint arXiv:0912.1016. Volume 6, December 2009.

[12] K. A. Carlos Buenosvinos Christian Soronellas. (2016). Domain-Driven Design in PHP. [On-Line]. Available: https://leanpub.com/ddd-in-php [January 15, 2019].

[13] N. Tune and S. Millett. (2015, May). Patterns, Principles, and Practices of Domain-Driven Design [On-Line]. Available: https://learning.oreilly.com/library/view/patterns-principles-and/9781118714706 [January 15, 2019].

BIODATA PENULIS

Rizki Mufrizal

Lahir di Puuk, 15 November 1993. Pendidikan sarjana di dapatkan dari Universitas Gunadarma pada Program Studi Teknik Informatika. Saat ini berkarir sebagai staf teknologi informasi di PT. Bank Danamon Indonesia.

Dina Indarti

Lahir di Bukittinggi, 20 April 1986. Pendidikan sarjana dan magister diperoleh dari Program Studi Matematika, Universitas Indonesia. Pendidikan doktor diselesaikan tahun 2014 di Program Studi Teknologi Informasi, Universitas Gunadarma. Saat ini adalah dosen tetap Program Studi Teknik Informatika, Universitas Gunadarma.