PERANCANGAN DAN IMPLEMENTASI LAYANAN *WEBSERVICES BIG DATA* PADA PENGUMPULAN DATA PRIBADI MENGGUNAKAN METODE *ROUND ROBIN* SEBAGAI *LOAD BALANCE*

SKRIPSI

Disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik (S.T)

Disusun oleh:

HANIF ANGGIT WICAKSONO

3332150017

JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNIK

UNIVERSITAS SULTAN AGENG TIRTAYASA

2022

LEMBAR PERNYATAAN KEASLIAN SKRIPSI

Dengan ini saya sebagai penulis skripsi berikut:

Judul	: Perancangan Dan Implementasi Layanan *Webservices Big Data* Pada Pengumpulan Data Pribadi Menggunakan Metode *Round Robin* Sebagai *Load Balance*
Nama Mahasiswa	: Hanif Anggit Wicaksono
NPM	: 3332150017
Fakultas/Jurusan	: Teknik/Teknik Elektro

Menyatakan dengan seseungguhnya bahwa skripsi tersebut di atas adalah benar-benar hasil karya asli saya dan tidak memuat hasil karya orang lain, kecuali dinyatakan melalui rujukan yang benar dan dapat dipertanggunjawabkan. Apabila dikemudian hari ditemukan hal-hal yang menunjukkan bahwa sebagian atau seluruh karya ini bukan karya saya, maka saya bersedia dituntut melalui hukum yang berlaku. Saya juga bersedia menanggung segala akibat hukum yang timbul dari pernyataan yang secara sadar dan sengaja saya nyatakan melalui lembar ini.

Cilegon, 17 Juni 2022

Hanif Anggit Wicaksono

NPM.3332150017

LEMBAR PENGESAHAN

Dengan ini ditetapkan bahwa Skripsi berikut:

Judul	: Perancangan Dan Implementasi Layanan *Webservices Big Data* Pada Pengumpulan Data Pribadi Menggunakan Metode *Round Robin* Sebagai *Load Balance*
Nama Mahasiswa	: Hanif Anggit Wicaksono
NPM	: 3332150017
Fakultas/Jurusan	: Teknik/Teknik Elektro

Telah di uji dan di pertahankan pada tanggal 08 September 2022 melalui Sidang Skripsi di Fakultas Teknik Universitas Sultan Ageng Tirtayasa Cilegon dan dinyatakan LULUS.

Dewan Penguji

		Tanda Tangan
Pembimbing I	: Prof. Dr. Ir. Supriyanto, M.Sc., IPM.	.....
Pembimbing II	: Cakra Adipura Wicaksana, S.T., M.T.	.....
Penguji I	: Dr. Eng. Teguh Firmansyah, S.T., M.T., IPM.	.....
Penguji II	: Masjudin, S.T., M.Eng.	.....

Mengtahui,
Ketua Jurusan

Dr. Romi Wiryadinata, S.T., M.Eng
NIP. 198307032009121006

PRAKATA

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Puji dan syukur kehadirat kepada Allah SWT karena atas Rahmat dan Karunia-Nya akhirnya penulis dapat menyelesaikan laporan skripsi ini. Skripsi ini disusun sebagai salah satu syarat dalam memperoleh gelar Sarjana Teknik Elektro di Universitas Sultan Ageng Tirtayasa. Ucapan terima kasih yang tak terhingga penulis ucapkan untuk kedua orang tua tercinta, kepada Oleh karena itu, dalam kesempatan ini penulis menyampaikan rasa terima kasih yang sebesar-besarnya kepada semua pihak yang telah banyak membantu dalam pelaksanaan, penulisan dan penyelesaian skripsi ini, yaitu:

(1) Orang tua yang selalu menyelipkan nama penulis di setiap do'a, memberikan cinta kasih dan memberikan dukungan baik moril maupun materil yang begitu besar selama proses pengerjaan laporan skripsi ini.
(2) Bapak Dr. Romi Wiryadinata, S.T., M. Eng., selaku Ketua Jurusan Teknik Elektro sekaligus Dosen Pembimbing Akademik selama masa perkuliahan.
(3) Bapak Prof. Dr. Ir. Supriyanto, M.Sc., IPM., selaku Dosen Pembimbing I Skripsi yang dengan sabar telah banyak memberikan arahan, bimbingan dan motivasi kepada penulis dalam menyelesaikan skripsi
(4) Bapak Cakra Adipura Wicaksana, S.T., M.T., selaku Dosen Pembimbing II Skripsi yang telah menyediakan waktu, tenaga dan pikiran untuk selalu mengarahkan penulis dalam penyusunan skripsi ini.

Akhir kata, saya berharap Allah SWT berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga skripsi ini membawa manfaat bagi pengembangan ilmu.

Cilegon, 17 Juni 2022

Penulis

ABSTRAK

Hanif Anggit Wicaksono

Teknik Elektro

Perancangan dan implementasi layanan *Webservices Big Data* pada pengumpulan data pribadi menggunakan metode *Round Robin* sebagai *Load balance*

Penelitian ini bertujuan untuk mengetahui kinerja metode *Round Robin* sebagai *Load balance*. Metode ini digunakan untuk membagi kerja *server Load balance* ke beberapa *server Web* yang menggunakan *Apache Web Server*. *Database server* yang digunakan adalah MySQL. *Index* merupakan objek struktur data tersendiri yang tidak bergantung kepada struktur tabel. Setiap *Index* terdiri dari nilai kolom dan penunjuk (atau *ROWID*) ke baris yang berisi nilai tersebut. Percobaan 100 *user*/30s dengan Indexes dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 13922ms, rata-rata *min. Respons time* 178ms, rata-rata *max. Respons time* adalah 25977ms dan *error* 6,02%. Dapat disimpulkan dengan menggunakan *Round Robin* dan *Indexing* akan mengurangi *Respons Time* dan mendapatkan nilai yang stabil.

Kata kunci: *Web API, Load balancing, Round Robin, Webservice*

ABSTRACT

Hanif Anggit Wicaksono

Electrical Engineering

Design and implementation of Big Data Webservices on personal data collection using the Round Robin method as Load balance

This study aims to determine the performance of the Round Robin method as a load balance. This method is used to divide the load balance server work among several Web servers that use Apache Web Server. The database server used is MySQL. An Index is a separate data structure object that does not depend on the table structure. Each Index consists of a column value and a pointer (or ROWID) to the row containing that value. Test of 100 users/30s with Indexes with Round Robin get an average Response time of 10 tries is 13922ms, average min. Response time 178ms, average max. Response time is 25977ms and error is 6,02%. It can be concluded that using Round Robin and Indexing will reduce Response Time and get a stable value.

Keywords: Web API, Load balancing, Round Robin, Webservice

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Tingkat aktivitas *server* dari pengguna internet yang sangat tinggi, tentu saja hal ini akan berdampak pada penyedia informasi. Kinerja *webserver* dan *database* sebagai media penyedia konten selalu diharapkan dapat memenuhi semua kebutuhan dari pengguna. Dampak ini tentu tidak diinginkan oleh beberapa instansi yang semua aktivitasnya sudah ketergantungan dengan jaringan komputer [1]. Oleh karena itu, instansi-instansi tersebut tidak ragu lagi untuk mengalokasikan dananya untuk membeli perangkat *server* khusus dengan kemampuan yang tinggi [2]. Jika tidak ditanggapi dengan serius, ini bisa saja berakibat pada *server-server* yang kelebihan beban permintaan (*request*) dari pengguna [3]. Hal ini disebabkan permintaan dari pengguna lebih besar daripada kemampuan *server* untuk memberikan layanan. Setiap hari permintaan layanan dari pengguna selalu meningkat [4]. Hal ini tentu saja berhubungan dengan semakin banyaknya perangkat-perangkat yang dapat menggunakan fasilitas internet seperti komputer, laptop, *netbook*, *smartphone*, tablet, dan perangkat lainnya.

Situs *web* dengan *traffic* data yang tinggi dapat menyebabkan beban kerja yang berat di sisi *server*, yang pada gilirannya akan mengakibatkan turunnya kinerja *server*, bahkan kegagalan sistem secara keseluruhan [5]. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan menerapkan teknik *Load balancing* [6]. *Load balancing* adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi [7]. *Load balancing* digunakan pada saat sebuah *server* telah memiliki jumlah *user* yang telah melebihi maksimal kapasitasnya [8]. *Load balancing* juga mendistribusikan beban kerja secara merata pada dua atau lebih komputer, link jaringan, CPU, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal. Salah satu *tools Load balancing* adalah Nginx [9].

Round robin merupakan salah satu algoritma penjadwalan proses dalam sistem operasi. *Round robin* dirancang untuk membagi waktu setiap proses pada porsi yang sama dan dalam urutan melingkar, menjalankan semua proses tanpa prioritas dikenal juga sebagai eksekutif siklik [10]. Penjadwalan *Round robin* mudah diterapkan, dan bebas *starvation* [11]. Penjadwalan *Round robin* juga dapat diterapkan untuk masalah penjadwalan lainnya, seperti penjadwalan paket data dalam jaringan komputer. *Round robin* dirancang untuk sistem *time sharing* [12].

Layanan *Microservice* termasuk dalam sistem perangkat lunak yang dapat dibangun dengan cepat dengan mengintegrasikan berbagai *web* yang ada dari API penyedia yang berbeda [13]. Beberapa dari bagian yang terbuka di dalam *platform* API *web* yang di Internet, seperti *web* yang dapat diprogram lainnya yang dikembangkan oleh pengembang internal [14]. Penyedia umumnya memberikan deskripsi tentang memanggil API *web* sesuai dengan format kustom mereka sendiri, karena tidak ada bahasa deskripsi yang diterima secara luas untuk *web* API [15].

Dengan peningkatan API *web*, sangat memakan waktu bagi pengembang sistem perangkat lunak untuk menemukan secara manual dan mengintegrasikan API yang sesuai. Untuk meringankan pengembangnya, beberapa pendekatan anotasi semantik untuk *web* API. Berdasarkan makna *Web* API itu sendiri, pengembang dapat dengan cepat menemukan API yang sesuai dan terintegrasi secara otomatis dengan bantuan teknologi komposisi layanan.

1.2 Rumusan Masalah

Berdasarkan penjalasan latar belakang pada penelitian ini maka, dapat di tentukan rumusan masalah yaitu sejauh mana kekuatan aksesibilitas, kecepatan *Response time* dan perbedaan sebuah *web server* yang menggunakan *Load balance* dengan *web server* yang hanya menggunakan *single server* serta menambahkan *Indexes*

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah di atas, tujuan penelitian ini adalah mengimplementasikan dan merancang arsitektur *Microservice* menggunakan *Round Robin* dan penambahan *Indexes* untuk membantu dalam pengumpulan data

dan mengetahui kinerja algoritma-algoritma *Round Robin Load balance* yang diimplementasikan pada *Web server* Nginx dengan *database* MySQL.

1.4 Manfaat Penelitian

Manfaat dari penelitian ini adalah:

a. Bagi instansi atau lembaga, penelitian ini dapat menjadi referensi dalam mengembangkan *server Load balance* yang menggunakan metode *Round robin*, serta dalam pemilihan *Engine Web server* dan *database* pada instansi masing-masing.
b. Bagi peneliti, penelitian ini dapat meningkatkan pengetahuan penulis dalam bidang *server Load balance* yang menggunakan metode *Round robin*, *Engine Web server* dan *database* MySQL.

1.5 Batasan Masalah

Dengan melihat tujuan penelitian dan situasi saat penelitian, maka dapat diambil batasan masalah pada penelitian ini yaitu sebagai berikut:

a. Perangkat lunak *server Web* yang akan digunakan adalah Apache dan Nginx. Kedua perangkat lunak ini dipilih karena keduanya merupakan perangkat lunak *server Web* yang paling banyak digunakan untuk melayani berbagai macam aplikasi, dari yang sederhana hingga yang *enterprise*.
b. Pengambilan data melalui sosial media untuk di implementasikan ke API *Webservices*.

1.6 Sistematika Penulisan

Skripsi ini terdiri dari lima bab. Bab I, yaitu pendahuluan yang mendeskripsikan tentang hal yang berkaitan dan mempunyai latar belakang perencanaan dan penelitian yang dilakukan dalam penelitian, meliputi latar belakang, perumusan masalah, tujuan penelitian, manfaat penelitian dan batasan masalah.

Bab II menjelaskan mengenai teori dasar yang digunakan dalam perencanaan dan penelitian diantaranya mengenai *Round robin*, Bahasa GOLANG, JWT(*JSON*

Web Token), RESTful *WebService*, MySQL, *Postman*, dan aplikasi yang dibutuhkan lainnya.

Bab III, yaitu metodologi penelitian yang berisikan tentang diagram alir penelitian, rancang *database api webservices*, perancangan sistem serta struktur table dan relasinya.

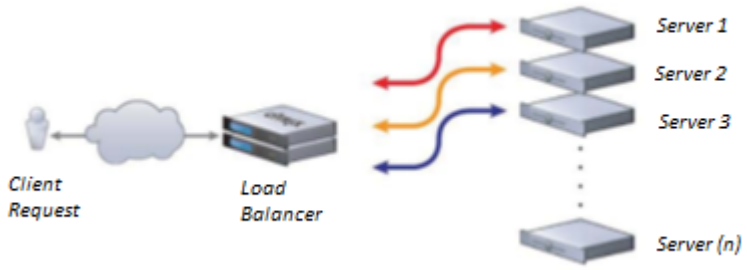
Bab IV, yaitu hasil dan pembahasan mengenai penelitian tentang data-data hasil dari uji coba yang di lakukan setiap pengambilan kumpulan data yang bertujuan untuk di analisis sesuai dengan sistemasi yang telah di rancang sebelumnya.

Bab V yaitu penutup yang berisikan mengenai hasil data dari penelitian yang berlangsung, lalu di tulis berupa rangkuman agar mempermudah untuk memahami dari penelitian tersebut, serta saran yang dapat digunakan untuk pengembangan penelitian yang akan di lakukan selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 *Round Robin*

Round robin merupakan salah satu algoritma penjadwalan proses paling sederhana pada sistem operasi. Algoritma penjadwalan *Round robin* mengirimkan setiap permintaan yang masuk ke *server* setelahnya di dalam daftar tanpa prioritas (dikenal dengan istilah *cyclic executive*). Algoritma *Round robin* bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lainnya. Jadi dalam tiga cluster *server* (*server* A, B dan C) permintaan 1 akan diberikan ke *server* A, permintaan 2 akan diberikan ke *server* B, permintaan 3 akan diberikan ke *server* C, dan permintaan 4 akan diberikan ke *server* A lagi. Konsep dasar dari algoritma *Round robin* ini adalah dengan menggunakan *time* sharing, pada intinya algoritma ini memproses antrian secara bergiliran [12]. Berikut proses Algoritma *Round robin* pada Gambar 2.1.



Gambar 2.1 Proses Algoritma *Round robin* [12]

Pada Gambar 2.1 *Round robin* mencoba mendistribusikan beban ke VM dalam urutan rotasi yang adil. Inti dari *Round robin* sendiri adalah seluruh VM yang berada di pusat data menerima beban yang sama dalam urutan melingkar tanpa memperhatikan kekuatan pemrosesan saat alokasi tugas. Ini efektif untuk pusat data yang memiliki semua VM yang mempunyai kekuatan pemrosesan yang sama. Adapun pusat data yang besar yang berada di VM memberi daya yang tidak efektif [16].

Namun, ada beberapa kelebihan dan kekurangan yang dimiliki oleh algoritma *Round robin* ini seperti yang akan peneliti uraikan di bawah ini:

1. Kelebihan algoritma *Round robin*:
    a. *Round robin* merupakan algoritma yang paling praktis, dengan bagan proses yang mana CPU diminta untuk mendapatkan prioritas.
    b. Untuk proses-proses yang kecil biasanya *Response time* nya berjalan lancar dan cepat.
    c. Mencegah terjadinya kondisi *deadlock* atau lebih dikenal dengan *starvision*.
    d. Memiliki *overhead* yang kecil jika ukuran proses yang rata-rata lebih kecil di bandingkan slot waktunya.
    e. Menghindari kesenjangan layanan atau ketidakadilan layanan terhadap proses-proses kecil seperti yang biasa terjadi pada FCFS.
2. Kekurangan algoritma *Round robin*:
    a. Waktu tunggu biasanya sangat lama untuk proses besar.
    b. Sering terjadi *convoy effect*
    c. Jika tempatnya terlalu kecil, maka sebagian proses tidak bisa diselesaikan oleh satu waktu saja.
    d. Memiliki performa yang buruk jika *quantum time* nya lebih besar dari pada prosesnya di banding FCFS.
    e. Jika waktunya kecil bisa menyebabkan *overhead*.
    f. Proses masukan dan keluaran mebutuhkan waktu yang sedikit lebih lama.

2.2 Bahasa GOLANG

Bahasa pemrograman GOLANG adalah bahasa program yang diciptakan oleh Google LLC. Tujuan dari pengembangannya adalah untuk membangun bahasa yang mempunyai keunggulan dari sisi kecepatan, keandalan, skalabilitas, dan kesederhanaan. GOLANG juga termasuk dalam bahasa yang dapat diketik secara statis serta menghasilkan kode biner pada mesin yang dapat dikompilasi. Selain itu, GOLANG juga dihimpun dari bahasa pemrograman C di abad ke –21. Bahasa Go juga dapat digunakan untuk kepentingan pembuatan aplikasi, *Web*site, dan *software* yang lainnya [17]. GOLANG mempunyai 3 komponen utama yaitu *Package Name*, *Imported Package* dan *Entrypoint* yang struktur dan contoh implementasinya dapat dilihat pada Gambar 2.2.

```
package main

import (
  "fmt"
)

//entrypoint
func main() {
  fmt.Println("Hello World!")
}
```

Gambar 2.2 Struktur dasar GOLANG [18]

Pada Gambar 2.2 terdapat 3 komponen utama dari Bahasa Golang yang terdiri dari *package name*, *imported package*, dan *entrypoint* yang di jelaskan pada berikut:

a. *Package name*

Setiap program yang ditulis menggunakan Go pasti memiliki *keyword package*. Seluruh kode di atas adalah bagian dari *package main*. Peneliti menggunakan *package* spesial bernama *main*, sebuah *package* yang dibutuhkan agar program bisa berjalan ketika dieksekusi.

b. *Imported Package*

Selanjutnya terdapat sebuah *statement* yang di dalamnya terdapat *keyword import*, *keyword* tersebut digunakan ketika ingin menggunakan sebuah *package built-in* yang telah disediakan Go. Pada kode sumber di atas peneliti menggunakan *package* fmt. Kegunaan dari *package* `fmt` adalah dapat berinteraksi dengan masukan dan keluaran, pada kode sumber di atas menggunakan fungsi `Println` dari *package* `fmt` untuk menampilkan teks pada standar keluaran.

c. *Entrypoint*

Pada kode Selanjutnya di atas memiliki fungsi `main()` yang bekerja sebagai *entrypoint* tempat pertama kali kode akan di baca saat program dieksekusi.

Secara singkat, fungsi dari Golang itu sendiri memberi kemudahan kepada pengembang aplikasi atau *Website* secara efektif, efisien, dan sederhana. Skalabilitas pada Golang adalah keunggulan tersendiri bahkan bahasa pemrograman Go ini cocok untuk pembangunan situs *e-commerce*. Selain itu,

fungsi Golang juga mengatasi *Website* saat *traffic* sedang terlalu tinggi. Peningkatan aktivitas baik pada aplikasi atau *Website* tanpa ada penanganan khusus tentu akan mempengaruhi performa itu sendiri. Golang dilengkapi fungsi *footprint* dan *concurrency*. Fungsi tersebut hanya membutuhkan kapasitas memori kecil sehingga membantu sebuah aplikasi atau *Website* tetap berjalan normal meski aktivitas naik. Berikut beberapa hal lain yang menjadi fungsi Golang:

a. Membantu pengembangan kode *server* jaringan, khususnya untuk *Web server* dan layanan mikro.
b. Perancangan aplikasi berbasis *Web*site yang lebih aman.
c. Skalabilitas pada Golang dapat membantu pengembangan teknologi *cloud computing*.
d. Membantu dalam membangun pengembang yang lebih *scalable* untuk kepentingan bisnis.
e. Membangun sistem yang kompleks dan membutuhkan kinerja tinggi.

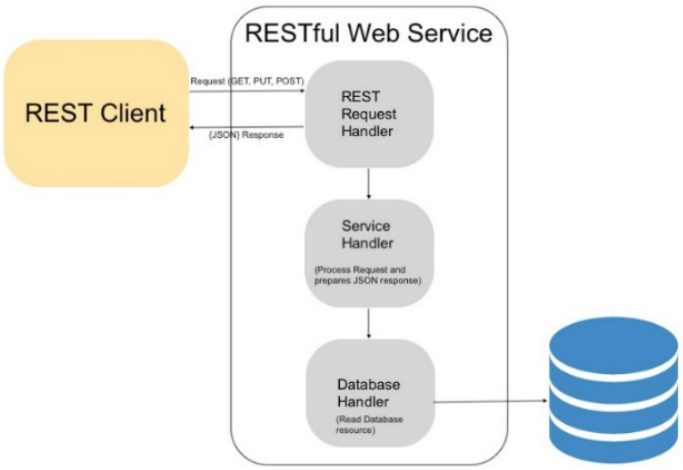
Hingga saat ini, Golang mulai banyak digunakan pada perusahaan-perusahaan besar maupun startup yang bergerak di bidang teknologi [13]. Hal ini dikarenakan pemrograman dengan Golang ini memiliki beberapa kelebihan sebagai berikut:

a. Mendukung konkurensi dalam sistem pemrograman dengan sangat baik dengan pengaplikasiannya sendiri yang cukup mudah.
b. Merupakan bahasa pemrograman yang bersifat *open source*.
c. Memiliki sistem *garbage collection* yang baik dengan memanfaatkan bantuan *built-in garbage collector process* (*Goroutines*).
d. Memiliki sintaks yang bersifat bersih, tidak mengotori sistem terlalu berlebihan.
e. Golang mampu membuat aplikasi dengan menggunakan waktu yang singkat dan biaya yang paling rendah dibandingkan lainnya.
f. *Developer* tidak perlu khawatir aplikasi akan mengalami *crash*.

2.3 *RESTful Web Service*

Arsitektur REST didasarkan pada prinsip-prinsip yang mendukung *World Wide Web*. Singkatnya, menurut prinsip-prinsip REST, antarmuka REST

bergantung secara eksklusif pada *Uniform Resource Identifier* (URI) untuk deteksi interaksi, dan biasanya pada *hypertext transfer ptrotocol* untuk kirim Pesan [14]. REST adalah salah satu gaya arsitektur yang dapat diadaptasikan ketika membangun *Web service* dapat dilihat pada Gambar 2.3.



Gambar 2.3 *RESTful Web Service* [19]

Pada Gambar 2.3 di jelaskan bahwa sumber daya biasanya diidentifikasi oleh *Uniform Resource Identifier* (URI), yang membuatnya dapat dialamatkan dan dapat dimanipulasi menggunakan protokol aplikasi, biasanya *HTTP* [20]. Titik akhir API adalah *URI* unik yang mengidentifikasi satu atau beberapa sumber daya. Sebagian besar *RESTful Web* API mengikuti serangkaian pedoman desain yang terkenal, yang mencakup penerapan metode *HTTP* standar sebagai berikut:

a. *GET*, digunakan untuk mengambil atau memanggil *resource* dari *server*.
b. *POST*, digunakan untuk membuat *resource* baru.
c. *PUT*, digunakan untuk mengubah *resource* yang ada.
d. *DELETE*, digunakan untuk menghapus *resource*.

Arsitektur ini sangat populer digunakan karena pengembangannya yang relatif mudah. Menggunakan pola *Request-Responsse* dalam berinteraksi, artinya ia memanfaatkan protokol *HTTP*. Dalam implementasinya arsitektur REST benar-benar memisahkan peran *client* dan *server*, bahkan keduanya tidak harus saling mengetahui. Artinya ketika terjadi perubahan besar di sisi *client*, tidak akan berdampak pada sisi *server*, begitu juga sebaliknya [21]. Ada beberapa kelebihan REST API yang dimiliki oleh REST API, diantaranya:

a. Dapat digunakan oleh beragam jenis bahasa pemrograman yang berbeda termasuk beragam platform yang digunakannya.
b. Lebih sederhana dan juga simpel, utamanya jika dibandingkan dengan penggunaan *SOAP*.
c. Lebih mudah untuk dipelajari.
d. Seperti *Web* yang mana selalu menggunakan *HTTP* di setiap bagian yang dimilikinya.
e. Aplikasi android yang menggunakan REST API jauh lebih cepat, dari aplikasi android berbasis *Web view*.

Kekurangan REST API, Selain kelebihannya REST API juga dikenal dengan beberapa kekurangan, seperti berikut:

a. Keamanan kurang baik, karena REST API melewati bagian protokol *HTTP* dalam proses penggunaannya.
b. Waktu akses yang biasanya lebih lama dibandingkan dengan *native library.*

2.4 JWT (*JSON Web Token*)

JWT merupakan salah satu standar *JSON* (RFC 7519) untuk keperluan akses *token*. *Token* dibentuk dari kombinasi beberapa informasi yang di-*encode* dan di-enkripsi. Informasi yang dimaksud adalah *header, payload*, dan *signature*. JWT dalam metode *Microservices* juga digunakan untuk menggantikan fungsi *Session* yang selama ini diketahui bahwa *Session* biasanya menyimpan informasi yang akan terus dipakai pada saat berhasil *login*. Cara kerja JWT untuk fitur *authentication* yaitu pada saat *user* telah berhasil *login* seperti *password* jadi ketika *users* berhasil melakukan *Login* maka server akan memberikan sebuah *Token*. *Token* tersebut akan disimpan oleh *users* pada *Local Storage* atau *Cookies Browser* dan bila *users* ingin mengakses halaman halaman tertentu maka harus menyertakan *token* tersebut. Untuk itu *users* akan mengirim balik *token* yang dikasih diawal tadi sebagai bukti bila *user* ini, sudah melakukan *login*. *Token* tersebut mempunyai struktur dasar dimana terdiri dari tiga bagian yaitu yang pertama *header* lalu kedua bagian *payload* atau datanya dan yang ketiga adalah bagian *verify signature*. skenarionya

yaitu dengan memanfaatkan *local storage* atau *cookies* sebagai media penyimpanan informasi *user* [22]. JWT terdiri dari 3 bagian, yaitu:

a. *Header*

Bagian pertama ini disebut dengan *header*, berisi algoritma yang digunakan untuk membuat *signature*. *Header* ini berisi informasi tentang algoritma dan jenis *token* yang digunakan. Adapun contoh dari *Header* dengan terjemahan pada *coding* dibawah ini.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Bagian ini merupakan data *string* yang di-*encode* menggunakan *encode base*64 agar bisa mendapatkan nilai asli dari teks tersebut dengan men-*decode*nya. *Token* tersebut ini dapat diverifikasi dan dipercaya karena sudah di-*sign* secara digital. *Token* JWT bisa di-*sign* dengan menggunakan *secret* (algoritma HMAC) atau pasangan *public* dan *private key* (algoritma RSA).

b. *Payload*

Bagian kedua disebut dengan *payload*. *Payload* berisi data yang ingin dikirim melalui *token* yang meliputi isi dari data/atribut yang akan di klaim seperti terjemahan pada *coding* dibawah ini.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Dalam penerapannya di otentikasi atau pun otorisasi, biasanya data ini berupa data yang sifatnya unik bagi *user*, *email*, *id/uuid*, dan juga data yang berkaitan dengan otorisasi seperti *role*, karena data tersebut akan digunakan sebagai tanda pengenal yang mengirimkan *token*.

c. *Signature*

Bagian ketiga adalah *signature*. *Signature* adalah *hash* gabungan dari *header*, *payload* dan sebuah *secret key* (berupa string *random* panjang biasanya). Bagian ini adalah hasil fungsi enkripsi (sesuai algoritma yang sudah disebutkan di bagian

header) dengan masukan *header*, *payload*, dan *secret* seperti terjemahan pada *coding* dibawah ini.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
  )
```

Pada *Signature* ini berguna untuk memverifikasi bahwa *header* maupun *payload* yang ada dalam *token* tidak berubah dari nilai aslinya (karena untuk membuat *payload* dan *header* palsu itu cukup mudah). *Signature*-nya sendiri tidak mungkin dapat diakali, karena sudah dalam berbentuk `hash;` yang mana adalah fungsi satu arah (tidak dapat dikembalikan ke nilai semula), dan algoritma *hashing*-nya juga memerlukan *secret key* yang mana hanya pembuat aplikasi yang tahu.

2.5 MySQL

MySQL atau dibaca *My Sequel* merupakan sebuah *database management system* atau sering disingkat DBMS yang dijalankan menggunakan perintah SQL (*Structured Query Language*) yang populer digunakan untuk pembuatan aplikasi berbasis *Website*. Selain itu SQL dipuji karena kesederhanaan sintaks yang pendek dan mudah dipahami. Sehingga SQL cocok dipilih sebagai bahasa terbaik untuk memulai untuk belajar data *science* bagi pemula sebelum belajar bahasa pemrograman. MySQL juga termasuk ke dalam RDBMS atau *Relational Database Management System*, dimana di dalam struktur *database*nya sehingga ketika proses pengambilan data menggunakan metode *relational database*. Yang juga menjadi penghubung antara perangkat lunak dan *database server* [23].

MySQL memiliki fungsi untuk mengelola dan membuat *database* dari sisi *server* yang memuat berbagai informasi dengan menggunakan bahasa SQL. Dalam data *science* fungsi lain dari MySQL digunakan untuk melakukan *query dataset* dalam jumlah besar agar dapat memudahkan pengguna dalam mengakses *dataset* tersebut dalam bentuk *string* atau teks. Sehingga akan mendapat pemahaman yang kuat tentang basis data relasional dan karenanya memungkinkan untuk menguasai dasar-dasar data *science*, seperti mengetahui *missing value*, selain itu juga dapat mengidentifikasi NULLS dan format *dataset*. Melalui filter, agregasi, dan

penggabungan, dengan bahasa SQL sangat memungkinkan untuk bermain-main dengan *dataset*, mengenalnya secara menyeluruh dan mengetahui bagaimana nilai-nilai tersebut didistribusikan dan bagaimana dataset disusun.

Ada beberapa alasan sehingga MySQL banyak digunakan oleh para *Web developer* [24]. Antara lain alasannya ialah:

a. MySQL merupakan *database* yang memiliki kecepatan yang tinggi dalam melakukan pemrosesan data, dapat diandalkan dan mudah digunakan serta mudah dipelajari. MySQL mudah digunakan karena MySQL telah banyak digunakan sehingga jika mempunyai masalah dengan *database* tersebut, dapat bertanya kepada orang melalui internet maupun orang sekitar yang alam menyelesaikan masalah tersebut serta banyaknya dukungan manual maupun refrensi di internet.
b. MySQL mendukung banyak bahasa pemrograman seperti C, C++, Perl, Phyton, Java dan PHP. Bahasa tersebut dapat digunakan sebagai bahasa pemrograman untuk berinteraksi maupun berkomunikasi dengan MySQL *server*, atau dapat juga digunakan sebagai komponen pembentuk antar muka dari suatu *database* MySQL.
c. Koneksi, kecepatan dan keamanan membuat MySQL sangat cocok diterapkan pengaksesan data melalui internet, dengan menggunakan bahasa pemrograman atau PHP sebagai antarmukanya.
d. MySQL dapat melakukan koneksi dengan *client* menggunakan TCP/IP.
e. MySQL dapat menangani *database* dengan skala yang sangat besar dengan jumlah *record* mencapai lebih dari 50 juta, dapat menampung 60 ribu tabel dan juga bisa menampung 5 milyar baris data. Selain itu, batas indeks pada tiap tabel data menampung mencapai 32 indeks.
f. Dalam hal relasi antar tabel pada suatu *database*, MySQL menerapkan metode yang sangat cepat, yaitu dengan menggunakan metode *one-sweep multijoin*. MySQL sangat efisien dalam mengelola informasi yang diminta yang berasal dari banyak tabel sekaligus.
g. *Multi-user*, yaitu dalam satu *database server* pada MySQL dapat diakses oleh beberapa *user* dalam waktu yang sama tanpa mengalami konflik atau *crash*.

h. Dalam segi keamanan, MySQL memiliki keamanan yang luar biasa. Akses *user* bisa diproteksi menggunakan *user validation* dalam bentuk terenkripsi.
i. MySQL merupakan *software* aplikasi yang bersifat gratis.

2.6 *Postman*

Postman adalah sebuah aplikasi yang berfungsi sebagai REST *Client* untuk uji coba REST API. Aplikasi ini memungkinkan pengembang dengan mudah membuat, berbagi, menguji, dan mendokumentasikan API. Sebenarnya itu sangat berguna, karena pengembang dapat membuat dan menyimpan permintaan *HTTP*, serta membaca tanggapan mereka [5].

Postman biasa digunakan oleh *developer* pembuat API sebagai *tools* untuk menguji API yang telah di buat. *Postman* merupakan *tool* untuk melakukan proses *development* API [25]. Untuk saat ini sudah banyak fitur-fitur yang sangat membantu dalam proses *development* API, diantaranya:

a. *Collection*

Pengelompokan *request* API yang bisa disimpan atau diatur dalam bentuk folder. Memudahkan untuk pengelompokan *request* sesuai dengan proyek yang di kerjakan.

b. *Environment*

Semacam *config* untuk menyimpan atribut dan atribut tersebut dapat digunakan ataupun dimanipulasi dalam proses *request* API.

c. *Response*

Developer dapat membuat *Mockup* API sebelum benar-benar mengimplementasikan ke dalam proyek.

d. *Mock Server*

Dengan fitur ini, *Mockup* API yang dibuat menggunakan fitur *Example Response* dapat diakses dari internet layaknya *Mockup* API tersebut sudah di implementasikan dan di *deploy* ke *server*.

e. *Script Test*

Fitur untuk melakukan validasi *Respons*, termasuk di dalamnya menuliskan *test* sesuai dengan kebutuhan dari API tempat mengirim permintaan. Tambahkan berapa banyak tes yang dibutuhkan untuk setiap permintaan. Saat menambahkan

tes ke folder atau koleksi, tes akan dijalankan setelah setiap permintaan di dalamnya.

f. *Automated Test (Runner)*

Menjalakan *request* dalam satu *collection* secara otomatis, dengan menggunakan *script test*. *Postman* adalah salah satu alat paling populer yang digunakan dalam pengujian API. Aplikasi ini memungkinkan pengembang dengan mudah membuat, berbagi, menguji, dan mendokumentasikan API. Sebenarnya ini sangat berguna, karena pengembang dapat membuat dan menyimpan permintaan *HTTP*, serta membaca tanggapan, juga dapat membuat skenario pengujian yang berbeda, yang memungkinkan kami untuk meningkatkan cakupan kode sumber aplikasi.

2.7 *Big Data*

Big data adalah kumpulan proses yang terdiri *volume* data dalam jumlah besar yang terstruktur maupun tidak terstruktur dan digunakan untuk membantu kegiatan bisnis. *Big data* sendiri merupakan pengembangan dari sistem *database* pada umumnya. Perbedaan disini adalah proses kecepatan, *volume*, dan jenis data yang tersedia lebih banyak dan bervariatif daripada DBMS (*Database Management System*) pada umumnya. Definisi dari *big data* juga dapat dibagi menjadi 3 bagian, yang biasa disebut dengan 3V:

a. Volume

Ukuran data yang dimiliki oleh *Big Data* memiliki kapasitas yang besar. Peneliti dapat mencoba melakukan proses data dengan ukuran yang besar untuk dijalankan.

b. Velocity

Kecepatan transfer data juga sangat berpengaruh dalam proses pengiriman data dengan efektif dan stabil. *Big data* memiliki kecepatan yang memungkinkan untuk dapat diterima secara langsung (*real-time*). Salah satu buktinya antara lain, adanya sistem operasi *online* berbasis *Microsoft Silverlight*, aplikasi perkantoran (*office*) berbasis *web* seperti *Office365*, *cloud storage* seperti *Dropbox* dan *GDrive*. Kecepatan tertinggi yang bisa didapatkan langsung melalui aliran data ke memori apabila dibandingkan dengan yang ditulis pada sebuah *disk*.

c. Variety

Jenis variasi data yang dimiliki oleh *Big Data* lebih banyak daripada menggunakan sistem *database* SQL. Jenis data yang masih bersifat tradisional, lebih terstruktur daripada data yang belum terstruktur. Contohnya adalah *text*, *audio*, dan *video* merupakan data yang belum terdefinisikan secara langsung dan harus melalui beberapa tahap untuk dapat diproses dalam sebuah *database*.

2.8 *Indexing*

Indexing adalah struktur data yang digunakan secara internal oleh DBMS untuk mempercepat pencarian [26]. Dalam proses *indexing* bisa dilakukan secara manual maupun otomatis. Pada *database*, *index* merupakan sebuah struktur data yang berisi kumpulan *keys* beserta referensinya ke aktual data di *table*. Tujuannya untuk mempercepat proses penentuan lokasi data tanpa melakukan pencarian secara penuh ke seluruh data (*full-scan*). Fitur *index* sangat membantu dalam mengoptimalkan proses *query*, terutama dengan data yang besar. Satu-satunya *additional cost* pada penggunaan fitur *index* adalah bertambahnya ukuran data secara total, karena *index* memiliki ukuran data tersendiri, meskipun relatif jauh lebih kecil daripada data asli.

2.9 Nginx

Nginx adalah *web server* berbasis *open source* yang memiliki keunggulan untuk membuat performa *website* terlihat lebih canggih dan *powerful*. Salah satu kelebihan Nginx adalah mudah terkonfigurasi [3]. Mulanya, fungsi Nginx adalah sebagai *HTTP Web serving*. Nginx semakin berkembang dan telah dimanfaatkan juga untuk *HTTP cache*, *server* proxy (IMAP, POP3, SMTP), dan *load balancer* (TCP, HTTP, UDP). Nginx adalah *Web server* yang bisa dipakai di berbagai sistem operasi, seperti Linux, Mac OS X, HP-UX, BSD Varian, dan Solaris [4]. Ketika seseorang mengirimkan permintaan untuk membuka halaman *web*, *browser* akan menghubungi *server website* tersebut. *Server* lalu mencari *file* halaman yang diminta oleh *user* dan mengirimkannya ke *browser*.

Proses ini menunjukkan cara kerja *server* untuk permintaan atau *request* sederhana. Contoh tersebut juga bisa disebut sebagai *single thread*. *Web server*

biasa membuat *single thread* untuk setiap permintaan, tapi tidak demikian dengan Nginx. Seperti yang telah disebutkan sebelumnya, Nginx menjalankan arsitektur yang *event-driven* dan asinkron. Ini menunjukkan bahwa *thread* yang sama atau serupa dikelola di bawah satu *worker process*, dan setiap *worker process* terdiri atas unit yang lebih kecil, disebut *worker connection*. Keseluruhan unit ini bertugas untuk menangani *request thread*. *Worker connection* mengirimkan permintaan ke *worker process*, yang juga dikirimkannya *ke master process. Master process* kemudian menampilkan hasil dari permintaan atau *request* tersebut.

2.10 Kajian Pustaka

Penelitian tentang sistem *Load balancing* yang digunakan untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi. *Load balancing* digunakan pada saat sebuah *server* telah memiliki jumlah *user* yang telah melebihi maksimal kapasitasnya [3].

Penelitian tentang menganalisis literatur tentang JSON *Web Token*. JSON *Web Token* (JWT) adalah objek JSON yang didefinisikan dalam RFC 7519 sebagai metode aman untuk mewakili sekumpulan informasi antara dua pihak. Aplikasi lingkungan rumah pintar, aplikasi *cloud* saas, aplikasi Manajemen *smartphone* menggunakan mekanisme JWT untuk otentikasi klien untuk memanfaatkan sumber daya *server* atau mengakses perangkat pada *platform* IoT. Pendekatan ini memiliki beberapa kerentanan otentikasi yang dapat disalahgunakan oleh penyerang untuk mengakses kembali sumber daya *server*. Kerentanan umum yang ada dalam pendekatan ini adalah penggunaan token yang sama hingga JWT berakhir atau pengguna melakukan operasi *logout* [22].

Penelitian tentang aplikasi *E-Agri* kegiatan pertanian yang memanfaatkan keunggulan dari teknologi. Metode pengembangan yang digunakan dalam perancangan dan pembuatan perangkat lunak ini adalah *rapid application development* (RAD). Dalam pembuatan perangkat lunak ini penulis menggunakan perangkat lunak PHP dan *Adobe Dreamweaver CS6* dan untuk *database* menggunakan MySQL [24].

Penelitian tentang teknik pengindeksan yang tepat yang diterapkan pada *Database* MySQL untuk sistem perawatan kesehatan dan masalah kinerja terkait menggunakan mesin vektor dukungan multikelas (SVM). *Database* pasien umumnya sangat besar dan berisi banyak variasi. Untuk pencarian cepat atau pengambilan cepat informasi yang diinginkan dari *database*, menjadi penting untuk memilih dan menerapkan teknik pengindeksan yang sesuai. *Multiclass* SVM diusulkan untuk menjadi solusi optimal karena SVM dapat dilatih dengan *dataset* pasien untuk memilih metode pengindeksan yang sesuai dalam *database* MySQL [27].

Penelitian tentang *Load balance cluster* bekerja dengan mengirimkan layanan-layanan di dalam sebuah jaringan ke *node-node* yang telah di-*cluster* untuk menyeimbangkan beban permintaan layanan di antara *node cluster*. Prinsip kerja *load balance cluster*, ketika node pada *cluster load balance* tidak bekerja maka aplikasi *load balance* akan mendeteksi kegagalan dan meneruskan permintaan ke *node cluster* lainnya [11].

Maka, berdasarkan kajian tersebut, memutuskan bahwa penelitian ini menggunakan metode *Round Robin* sebagai *Load Balancing* yang ada pada Nginx untuk menyalurkan lalu lintas yang masuk secara berurutan dari satu *server* ke *server* lainnya, dan *Indexing* sebagai proses penyimpanan dan pengaturan konten serta menggunakan JWT (JSON *Web Token*) sebagai sekuriti data.

BAB III
METODE PENELITIAN

3.1 Metode Penelitian

Tahapan penelitian yang di ajukan dalam proses penelitian mencangkup analisa kebutuhan *hardware* dan *software* yang digunakan, hingga pengujian keseluruhan sistem. Penelitian diawali dengan melakukan analisa terhadap kebutuhan akan *hardware*, *software*, dan komponen pendukung yang dibutuhkan. Kemudian studi literatur, yaitu melakukan pengumpulan data refrensi untuk dipelajari dari jurnal penelitian, baik jurnal penelitian nasional maupun internasional, serta buku artikel dan juga *Web*site untuk digunakan menjadi refrensi dalam penelitian ini. Lalu identifikasi masalah, mengidentifikasi masalah yang sebelumnya belum terpenuhi didalam suatu perancangan penelitian.

Kemudian perancangan *Database* RESTful API, JWT (JSON *Web Token*) dan *Indexing*, merancang *database* menggunakan MySQL kemudian merancang sekuriti menggunakan JWT sebagai autentikasi dan otorisasi dan penambahan metode *Indexing*. Perancangan metode *Round robin*, merancang sistem *Load balancing* menggunakan metode *Round robin* dengan Nginx. Kemudian pengujian sistem, menguji sistem secara keseluruhan agar sistem dapat berjalan dengan yang diharapkan, dan jika pengujian sistem ini tidak sesuai yang di rencanakan, maka dilakukan perbaikan pada perancangan sistem. Terakhir penulisan laporan akhir, menulis hasil penelitian dan kesimpulan dalam bentuk laporan akhir.

3.2 Identifikasi Masalah

Salah satu teknologi yang belum termanfaatkan adalah penggunaan sistem database yang dapat memastikan integritas data yang dikirim dan dapat digunakan untuk otentikasi/otorisasi dua aplikasi yang berbeda. Menggunakan otentikasi dapat membantu meningkatkan sekuritas dari data tersebut. Kemudian mengoptimasi *server virtual* yang sebelumnya menggunakan *single server* lalu di optimalkan menggunakan *multi-server*. Menggunakan *multi-server* berguna membantu *server-server* tersebut mentransfer data secara efisien, dan mengoptimalkan penggunaan aplikasi pengiriman sumber daya sehingga terhindar dari *server* yang *overload*.

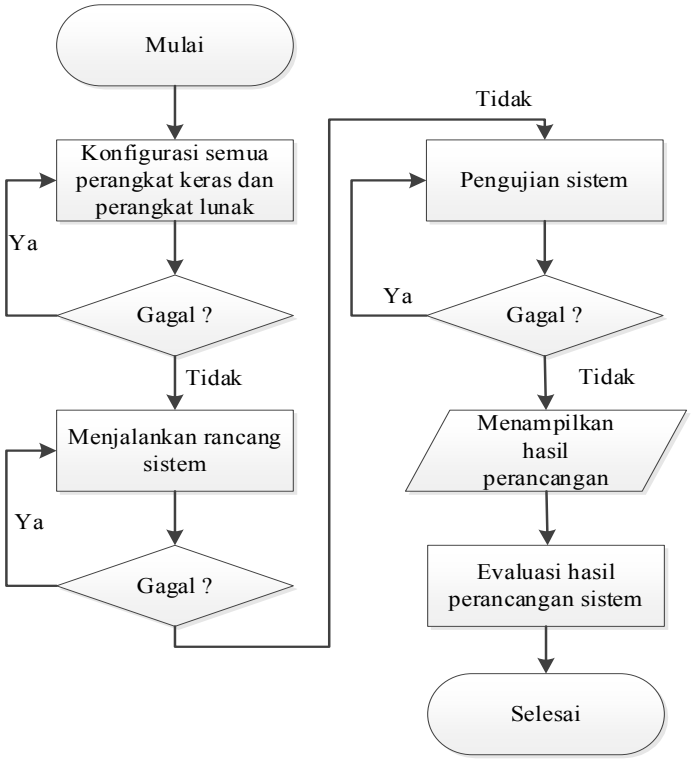
Algoritma yang dipakai dalam penelitian ini menggunakan Algoritma *Round robin*. Metode ini dapat merotasi *server* dengan mengarahkan *traffic* ke *server* yang pertama tersedia, lalu memindahkan *server* tersebut ke bawah dari antrean. Paling berguna saat *server-server* dengan spesifikasi serupa dan tidak banyak terdapat koneksi yang persisten.

3.3 Perancangan Sistem

Perancangan sistem adalah proses perancangan untuk merancang sistem atau memperbaiki sistem yang telah ada sehingga sistem menjadi lebih baik serta dapat mengerjakan pekerjaan secara efektif dan efisien, proses rancangan bisa berupa rancangan masukan, rancangan keluaran, dan rancangan *file*.

3.3.1 Alur Perancangan Sistem

Di bagian sub-bab ini menjelaskan tentang alur dari perancangan sistem yang terdiri dari konfigurasi perangkat lunak, perancangan sistem, pengujian sistem, dan hasil perancangan. Alur perancangan sistem tersebut digambarkan melalui *flowchart* yang dapat dilihat pada Gambar 3.1.

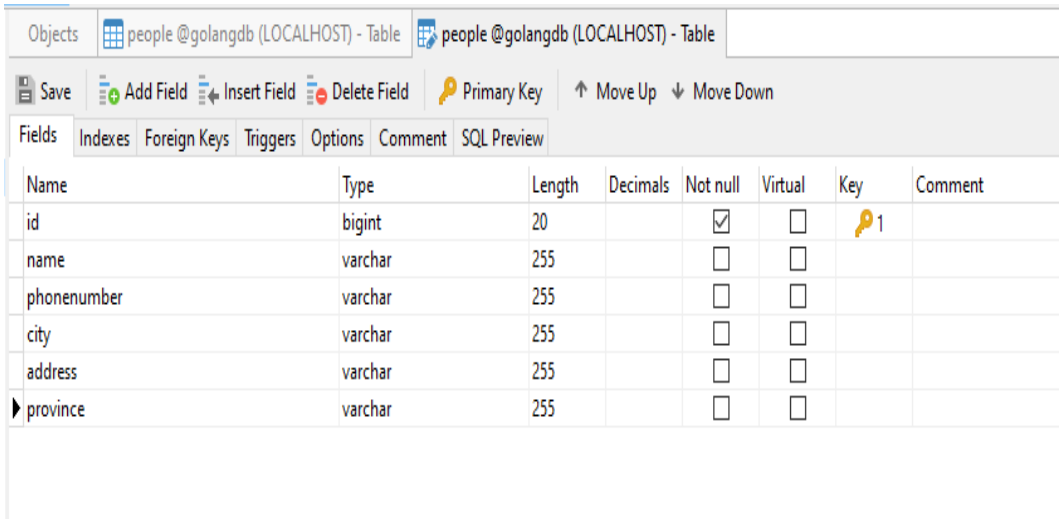


Gambar 3.1 Diagram Alur Perancangan Sistem

Memulai dari konfigurasi perangkat lunak yang digunakan yang disebutkan sebelumnya di sub-bab identifikasi masalah, lalu menjalankan rancangan sistem suatu infrastruktur, kemudian melakukan pengujian sistem untuk menguji sistem yang telah dirancang sebelumnya, dan yang terakhir menampilkan hasil perancangan sistem sekaligus melakukan evaluasi dari hasil perancangan tersebut.

3.3.2 Sistem Basis Data

Dalam perancangan sistem, dibutuhkan suatu tempat untuk menyimpan data-data yang dibutuhkan sistem, data-data tersebut dapat berupa *file* atau tabel. Perancangan basis data yang digunakan dalam penelitian ini dengan menggunakan *Database* MySQL terdiri dari satu tabel yang dilabeli *person,* memiliki 6 *field* di antaranya adalah *id*, *name*, *phonenumber*, *city*, *address*, dan *province*. Adapun struktur dari tabel *people* seperti yang ditunjukkan pada Gambar 3.2.



Gambar 3.2 Struktur Tabel Basis Data *people*

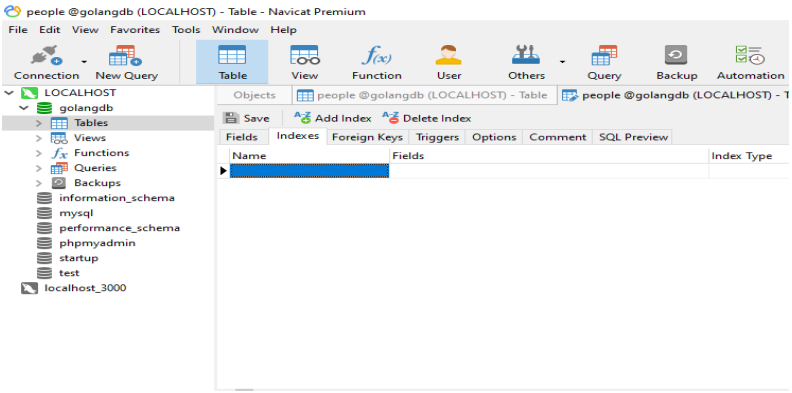
Pada Gambar 3.2 menunjukkan *Fields* dengan *row id* sebagai *primary keys* mempunyai format tipe *bigint* dengan *total length* 20. Kemudian untuk *name, phonenumber, city, address,* dan *province* mempunyai total *length* yang sama yaitu 255 dan mempunyai format tipe *varchar.*

Basis Data yang dirancang mempunyai *dummy* data yang diimpor dari *Microsoft Excel* berformat .csv berjumlah 59.283 yang bertujuan menjadi tolak

ukur perhitungan seberapa cepat *Response time, min. Response time, max. Response time,* dan *error.*

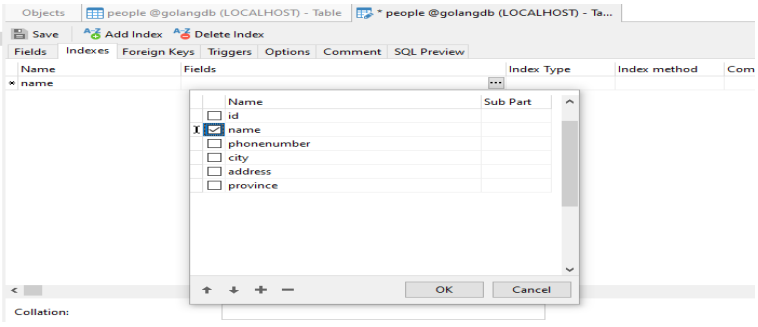
3.3.3 Sistem Penambahan *Indexes*

Index adalah objek pada MySQL yang berisi data yang terurut dari nilai-nilai pada satu atau lebih *field* dalam suatu *table*. Sama seperti daftar isi pada sebuah buku, *index* terutama digunakan untuk mempercepat pencarian terhadap suatu set data dengan kondisi tertentu yang melibatkan kombinasi *field* yang sudah didefinisikan dalam suatu *index*. Tanpa *index*, pencarian data biasanya akan memakan waktu lama, terutama jika data sudah dalam skala jumlah yang sangat besar. *Index* pada tabel dapat dikonfigurasi pada menu *Indexes* di *Software Navicat* yang dapat dilihat pada Gambar 3.3.



Gambar 3.3 Tab *Indexes* di *Navicat*

Pada Gambar 3.3 Tab *Indexes* digunakan untuk fitur *index* di *database* melalui *software Navicat* dapat mengkonfigurasikannya pada tab *Indexes*, dengan memilih *field* yang akan diimplementasikan suatu *index* dan mengubahnya hanya dengan mencentang *field* yang akan dipilih sebagai *index*, seperti tampak pada Gambar 3.4.

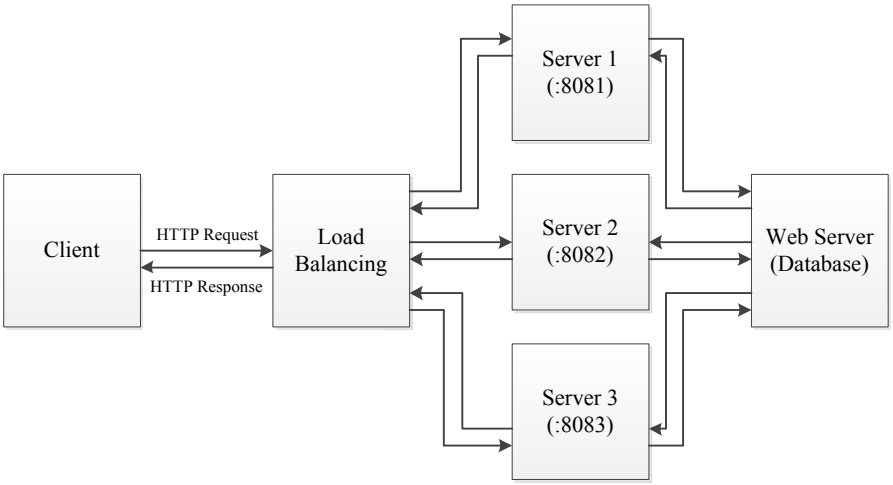


Gambar 3.4 Kotak edit *name* untuk mengatur indeks

Pada Gambar 3.4 Kotak edit *name* digunakan untuk mengatur *index* di *database* melalui *software Navicat* dapat mengkonfigurasikannya pada tab *Indexes*, dengan memilih *field* yang akan diimplementasikan suatu *index* dan mengubahnya hanya dengan mencentang *field* yang akan dipilih sebagai *index*.

3.3.4 Sistem *Load balancing*

Pada penelitian ini menggunakan perancangan sistem jaringan dengan menerapkan algoritma *Load balancing* sebagai mekanisme dalam pemilihan *server*. Perancangan sistem dapat dilihat dengan blok diagram sistem *Load balancing* pada Gambar 3.5.



Gambar 3.5 Blok diagram sistem *Load balancing*

Pada gambar 3.5 merupakan skenario percobaan dengan menggunakan 3 buah *virtual server* yang dihubungkan dengan *Load balancing*. Masing-masing *virtual server* di mulai melalui *Command Prompt*. Proses *Load balancing* secara umum, yaitu pembagian beban pada *server* dalam membagi *traffic* secara otomatis. Pada awal proses, *client* melakukan *request* kepada *server*. *Request* dari *client* akan masuk ke dalam *controller Load balancing* dan dilakukan pengecekan *server*. Kemudian *controller* akan memilih *server* sesuai algoritma *Load balancing*. *Request client* akan diteruskan ke dalam antrian *server*. *Server* akan merespon dan mengirimkan data kepada *client*. Sistem *Load balancing* akan terus berjalan hingga *request* dari *client* telah berhenti.

3.4 Konfigurasi Sistem

Sebelum menjalankan program ada beberapa hal yang diperhatikan, yaitu kebutuhan sistem dan konfigurasi dari sistem. Tujuan pokok dari sistem komputer adalah mengolah data untuk menghasilkan informasi. Dalam melaksanakan tujuan pokok tersebut diperlukan adanya elemen-elemen yang mendukung. Elemen-elemen dari sistem tersebut antara lain kebutuhan perangkat keras (*Hardware*) dan kebutuhan perangkat lunak (*Software*).

3.4.1 Konfigurasi Algoritma *Round Robin*

Sebelumnya telah dibahas mengenai perancangan sistem. Dimana dalam perancangan sistem tersebut terdapat langkah dalam konfigurasi perangkat. Konfigurasi perangkat merupakan langkah awal dalam melakukan analisa terhadap algoritma *Load balancing*. Pengaturan algoritma *Load balancing* dilakukan pada *server controller Load balancing*. Konfigurasi dilakukan pada *file default* pada direktori `etc/nginx/sites-available`. *Source code* konfigurasi dari *Load balancing Round robin* pada aplikasi *web server nginx* dapat dilihat pada Gambar 3.6.

localhost.conf

C: > Program Files > nginx-1.23.0 > conf > sites-available > localhost.conf

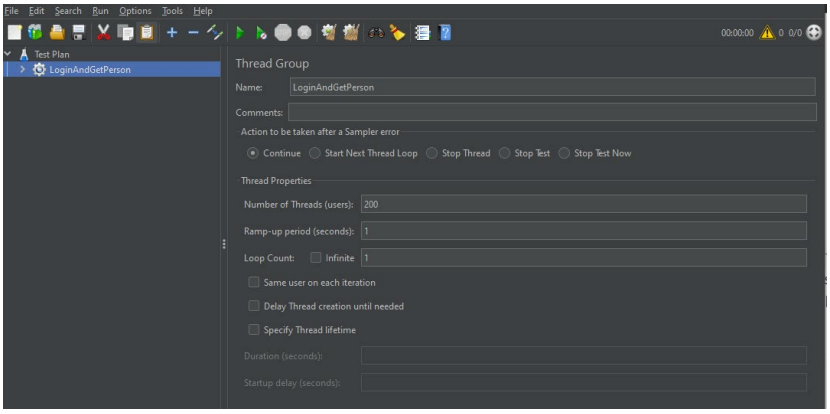
```
upstream localhost {
  server localhost:8081;
  server localhost:8082;
  server localhost:8083;
}

server {
  listen          8080;
  server_name     localhost;
  access_log      "C:/Program Files/nginx-1.23.0/logs/localhost.access.log" combined;

  location / {
    proxy_pass      http://localhost;
  }
}
```

Gambar 3.6 *Source Code Algoritma Round robin*

Round robin merupakan metode *default* apabila parameter algoritma tidak ditentukan. Maka otomatis metode yang digunakan menggunakan algoritma *Round robin* dan beban akan dibagi secara rata ke seluruh *server*.

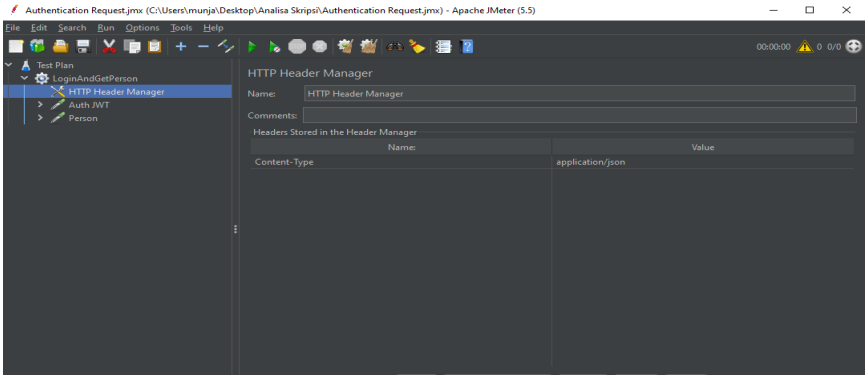
3.4.2 Konfigurasi Apache JMeter

Apache JMeter untuk pengujian kinerja membantu untuk menguji sumber daya statis dan dinamis, membantu menemukan pengguna bersamaan di situs *Web* dan berbagai analisis grafis untuk pengujian kinerja. Pengujian kinerja JMeter meliputi uji beban dan uji tegangan aplikasi *Web*. Ini adalah urutan konfigurasi dari perangkat lunak Apache JMeter pada Gambar 3.7.



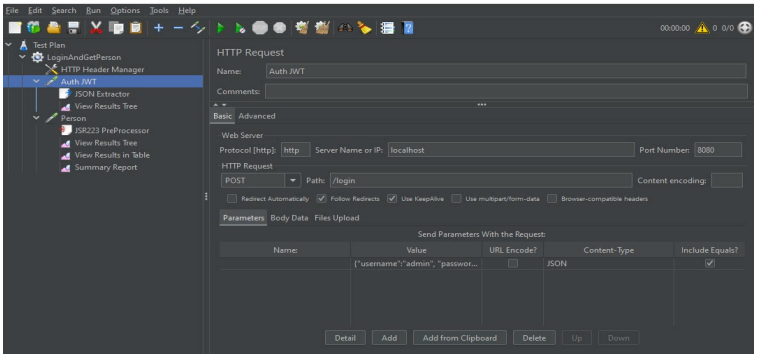
Gambar 3.7 Menambahkan *Thread Group*

Pada Gambar 3.7 menjelaskan tentang *Thread* Group yang digunakan untuk mengelompokan *service* yang akan di tes. Contoh mempunyai *service* Mahasiswa dan Dosen, kedua *service* ini sebaiknya dibuatkan *Thread Group* masing-masing pada Gambar 3.8.



Gambar 3.8 HTTP *Header Manager*

Pada Gambar 3.8 HTTP *Header Manager* digunakan untuk menambahkan *name* dan *value*, dengan kolom name diisi dengan *Content-Type* dan *Value application-json* yang di jelaskan pada Gambar 3.9.



Gambar 3.9 Konfigurasi HTTP *Request*

Pada Gambar 3.9 HTTP *Request* berfungsi untuk menambahkan node HTTP *Request*. Di node inilah peneliti akan menentukan *Web service* yang akan di tes. Misal *Web service* mempunyai layanan *Auth* JWT yaitu untuk menkonfigurasi otentikasi ke *database* menggunakan JSON *Extractor* dan *Person* untuk menerima otentikasi dari *Auth* JWT menggunakan *JSR223 PreProcessor*. Layanan ini akan ditambahkan sebagi bagian dari HTTP *Request*.

3.5 Perangkat Penelitian

Instrumen yang di gunakan pada penelitian kali ini meliputi perangkat keras (*Hardware*) dan perangkat lunak (*Software*). Berikut spesifikasi lebih rinci terkait *Hardware* dan *Software* yang digunakan penelitian dalam aplikasi sebagai berikut:

1. Perangkat keras

Satu unit laptop dengan *processor* intel core i3-7020. RAM dengan tipe DDR4 berkapasitas 4GB, penyimpanan *internal* MidasForce SSD berkapasitas 256GB, dan *Graphics card* menggunakan Intel UHD.

2. Perangkat lunak

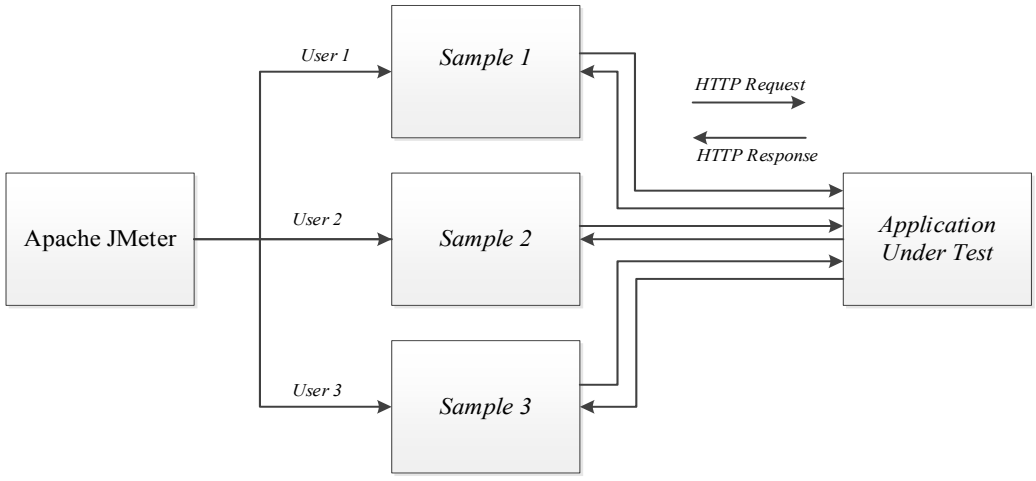
Software yang di gunakan adalah sebagai berikut:

a. Sistem Operasi yang di gunakan adalah *Windows 10 Home Single Language* 64bit (*build* 19043)
b. Perangkat Lunak yang digunakan sebagai *testing* API adalah *POSTMAN* (v9.25.2)
c. Bahasa Pemrograman yang digunakan adalah GOLANG (GO *Language*) (v1.18.4)

d. *Web server* yang digunkan adalah *Apache* (v2.5.53)
e. *Database Server* yang digunakan adalah MySQL dengan jenis *server* MariaDB(v10.4.24) dan perangkat lunak tambahan menggunakan *Navicat* (v15.0.22)
f. Perangkat lunak yang digunakan sebagai *code editor* adalah *Microsoft Visual Studio Code* (v1.69.2)
g. Perangkat lunak yang digunakan sebagai *load balancer* adalah *Nginx* (v.1.23.0)
h. Perangkat lunak yang digunakan sebagai *test performance* menggunakan *Apache JMeter* (v.5.5)

3.6 Desain Pengujian

Uji coba kali ini menggunakan perangkat lunak Apache JMeter, Aplikasi Apache JMeter adalah perangkat lunak *open source*, 100% aplikasi Java murni dirancang untuk memuat tes perilaku fungsional dan mengukur kinerja. Pada pengujian ini akan menggunakan dua *Confing Element* yaitu, *HTTP Cookie Manager* dan *HTTP Request Default*. *HTTP Request Default* merupakan salah satu fitur yang ada pada *Config Element* agar pengujian lebih optimal sehingga *Server Name* atau *IP* and *Port Number* tidak di jalankan berulang-ulang. *HTTP Cookie Manager* merupakan salah satu fitur yang ada pada *Config Element* agar dapat menyimpan *cookie* yang dapat digunakan oleh setiap *request* selanjutnya. Alur kinerja dari Apache JMeter pada Gambar 3.10.



Gambar 3.10 Alur Kinerja *Apache JMeter*

Pada Gambar 3.10 terlihat bahwa tujuan dari pengujian yaitu untuk mengetahui perbandingan performa dari algoritma *Round robin* pada *Web server*. Pengujian juga dilakukan untuk mengetahui nilai rata-rata pada setiap parameter yang diberikan. Pemberian *request* digunakan untuk memberi beban kepada *server* dalam satu waktu. Hasil dari setiap percobaan akan dibandingkan dan dilakukan analisa. Sehingga akan didapat perbandingan perfoma antara algoritma *Round robin* dan tidak memakai algoritma *Round robin*, serta memakai *Index*es atau tidak memakai *Index*es yang dapat disimpukan pada akhir penelitian. Berikut tahapan uji coba menggunakan perangkat lunak Apache JMeter pada Tabel 3.1.

Tabel 3.1 Tahapan pengujian penelitian

Urutan percobaan	**Deskripsi Percobaan**
1	Mengaktifkan *server* tanpa *Load balancing*, *client* melakukan *Request* GET *Person* all.
2	Mengaktifkan 3 *server* dengan *Load balancing*, *client* melakukan *Request* GET *Person* all.
3	Mengaktifkan *server* tanpa *Load balancing* menggunakan metode *Indexes*, *client* melakukan *Request* GET *Person* all
4	Mengaktifkan *server* dengan *Load balancing* menggunakan metode *Indexes*, *client* melakukan *Request* GET *Person* all.
5	Mengaktifkan *server* tanpa *Load balancing* dan tidak menggunakan metode *Indexes*, *client* melakukan *Request* GET *Person* all.
6	Mengaktifkan *server* dengan *Load balancing* tetapi tidak menggunakan metode *Indexes*, *client* melakukan *Request* GET *Person* all.

Pengujian pada parameter *Response time* di atas yang dilakukan sebanyak 6 kali, mempunyai 5 kategori sebagai berikut.

a. 40 req/30s (*rate*) dengan banyak koneksi 40.
b. 60 req/30s (*rate*) dengan banyak koneksi 60.
c. 80 req/30s (*rate*) dengan banyak koneksi 80.
d. 90 req/30s (*rate*) dengan banyak koneksi 90.

e. 100 req/30s (*rate*) dengan banyak koneksi 100.

Hasil dari pengujian dengan mengirimkan *request* akan didapat nilai dari *Response time* yang kemudian diambil nilai rata-ratanya dari hasil 10 kali percobaan. Perekaman hasil pengujian pada Apache JMeter dilakukan secara otomatis menjadi tabel hasil pengukuran disetiap percobannya, dan disimpan hasilnya secara manual dengan format *.csv*. Hal tersebut dilakukan karena pada Apache JMeter tidak ada fitur untuk menyimpan hasil pengukuran secara otomatis.

Pengujian yang dilakukan menggunakan parameter *Samples* (*User*/s), *Response time* Average (ms), *Min Respons*e *time* (ms), *Max Response time* (ms), dan *Error*. Berikut penjelasan dari masing-masing parameter:

a. Samples

Samples menangkap jumlah total sampel yang dikirim ke *server*. Contoh menempatkan Pengontrol *Loop* untuk menjalankannya 5 kali permintaan khusus ini dan kemudian 2 iterasi (Disebut Hitungan *Loop* di Grup Utas) diatur dan uji beban dijalankan untuk 100 pengguna.

b. *Response time Average*

Ini adalah waktu *Response* rata-rata untuk permintaan *HTTP* tertentu. Waktu *Respons*e ini dalam milidetik, dan rata-rata untuk 5 *loop* dalam dua iterasi untuk 100 pengguna.

c. Min Response time

Waktu minimum yang dihabiskan oleh permintaan sampel yang dikirim untuk label ini. Total sama dengan waktu minimum di semua sampel.

d. Max Response time

Pengeluaran ikatan maksimum berdasarkan permintaan sampel yang dikirim untuk label ini. Totalnya sama dengan waktu maksimum di semua sampel.

e. Error

Persentase total kesalahan yang ditemukan untuk permintaan sampel tertentu. 0,0% menunjukkan bahwa semua permintaan berhasil diselesaikan. Total sama dengan persentase kesalahan sampel di semua sampel (Total Sampel).

BAB IV
HASIL DAN PEMBAHASAN

4.1 Analisis Hasil Perancangan

Bab ini menguji hasil implementasi *Load balancing* pada sistem pengumpulan data. Dari hasil pengujian tersebut, akan dilakukan pembahasan untuk membandingkan antara memakai metode *Round robin* atau tidak memakai metode *Round robin* dan penambahan *Index*es dengan tidak memakai *Index*es.

Data pengujian akan dikumpulkan dari hasil *summary report* yang di ekspor ke *Microsoft Excel* pada *tool* JMeter untuk mendapatkan hasil performa dari metode *Round robin*. Pengujian ini dilakukan untuk mengetahui performa dari *Web server*, parameter yang digunakan yaitu *Respons time*. *Respons time* adalah waktu menentukan hasil rata-rata dari permintaan *HTTP*. *Source code* bisa di akses di halaman URL https://github.com/hanifanggit/golang-jwt-roundrobin .

4.1.1 Pengujian tanpa *Round Robin*

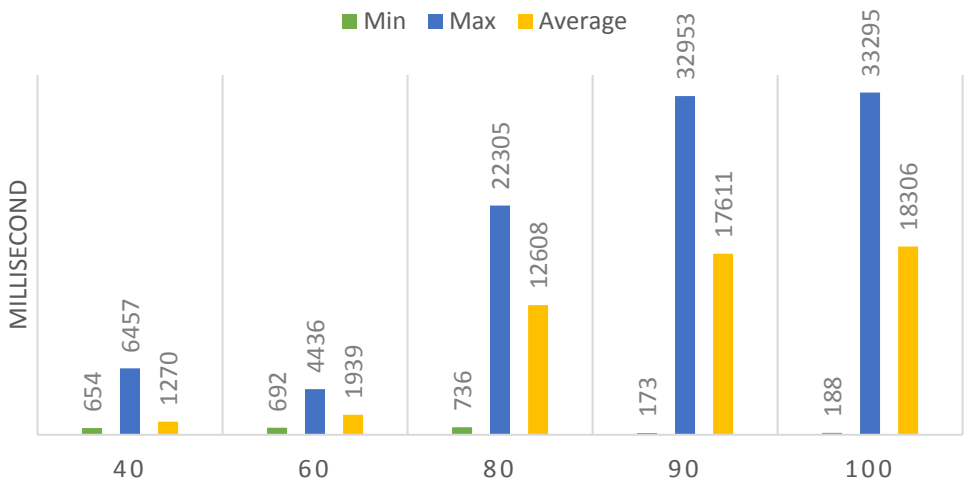
Berikut pengujian tanpa *Round robin* dengan sampel 40, 60, 80, 90, dan 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (*Average, Min, Max*) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian tanpa *Round robin* dapat dilihat pada Tabel 4.1.

Tabel 4.1 Pengujian rata-rata percobaan tanpa *Round Robin*

Percobaan	*Min* (ms)	*Max* (ms)	*Average* (ms)	*Error* (%)
40	654	6.457	1.270	0,00%
60	692	4.436	1.939	0,00%
80	736	22.305	12.608	0,50%
90	173	32.953	17.611	10,98%
100	188	33.295	18.306	15,80%

Pada Tabel 4.1 percobaan 40 *user*/30s tanpa *Round robin* mendapatkan rata-rata *Response time* dari 10 kali percobaan adalah 1270ms, rata-rata *min. Response time* 654ms, rata-rata *max. Response time* adalah 6457ms dan *error* 0.00%, lalu percobaan 60 *user*/30s tanpa *Round robin* mendapatkan rata-rata *Response time* dari 10 kali percobaan adalah 1939ms, rata-rata *min. Response time* 692ms, rata-rata

max. Response time adalah 4436ms dan *error* 0,00%, percobaan 80 *user*/30s tanpa *Round robin* mendapatkan rata-rata *Response time* dari 10 kali percobaan adalah 12608ms, rata-rata *min. Responss time* 736ms, rata-rata *max. Response time* adalah 22305 ms dan *error* 0,50%, percobaan 90 *user*/30s tanpa *Round robin* mendapatkan rata-rata *Response time* dari 10 kali percobaan adalah 17611ms, rata-rata *min. Response time* 173ms, rata-rata *max. Response time* adalah 32953ms dan *error* 10,98%, percobaan 100 *user*/30s tanpa *Round robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 18306ms, rata-rata *min. Respons time* 188ms, rata-rata *max. Respons time* adalah 33295ms dan *error* 15,80%, bentuk grafik dapat dilihat pada Gambar 4.1.



Gambar 4.1 Grafik rata-rata percobaan tanpa *Round robin*

Pada Gambar 4.1 pengujian tanpa *Round robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90 *user*, dan 100 *user*. Rata-rata dari 40 *user* terlihat dari grafik diatas lebih stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil walaupun di percobaan ke-1 mempunyai nilai yang cukup tinggi, rata-rata dari 80 *user* terlihat grafik yang tidak stabil dan mempunya nilai *error* 0,50%, dikarenakan semakin banyaknya *request* daripada 40 *user* dan 60 *user*, rata-rata dari 90 *user* dan 100 *user* terlihat grafik rata-rata *Respons time* yang hampir sama karena nilai parameter yang tidak jauh jaraknya dan mempunyai nilai *error* untuk 90 *user* 10,98% dan 100 *user* 15,80%, dibandingkan dengan 40 dan 60 *user* yang tidak mempunyai *error*. 90 dan

100 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single server*.

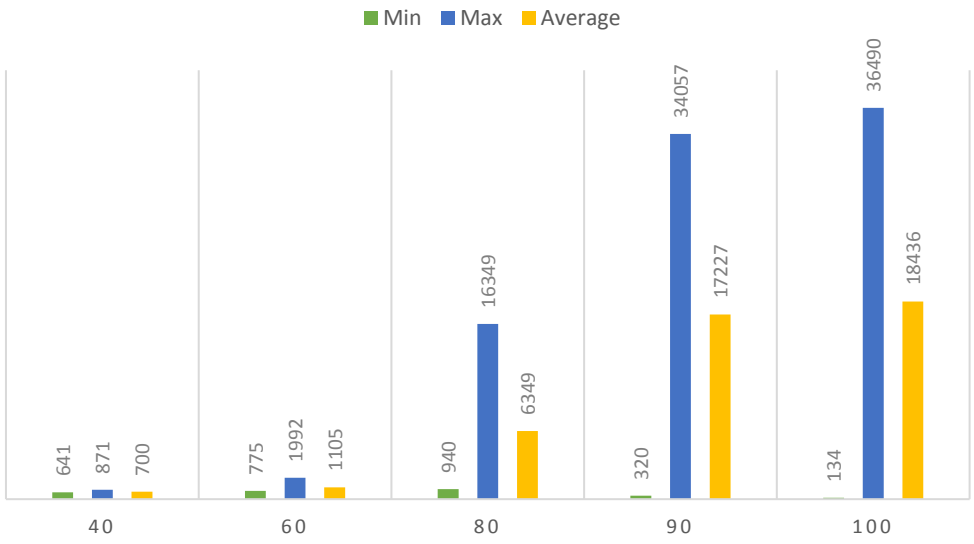
4.1.2 Pengujian dengan *Round Robin*

Berikut pengujian dengan *Round robin* dengan sampel 40, 60, 80, 90, dan 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (*Average, Min, Max*) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian dengan *Round robin* dapat dilihat pada Tabel 4.2.

Tabel 4.2 Pengujian rata-rata percobaan dengan *Round Robin*

Percobaan	***Min* (ms)**	***Max* (ms)**	***Average* (ms)**	***Error* (%)**
40	641	871	700	0,00%
60	775	1.992	1.105	0,00%
80	940	16.349	6.349	0,00%
90	320	34.057	17.227	7,65%
100	134	36.490	18.436	6,80%

Pada Tabel 4.2 percobaan 40 *user*/30s dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 700ms, rata-rata *min. Respons time* 641ms, rata-rata *max. Respons time* adalah 871ms dan *error* 0,00%, lalu percobaan 60 *user*/30s dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1105ms, rata-rata *min. Respons time* 775ms, rata-rata *max. Respons time* adalah 1992ms dan *error* 0,00%, percobaan 80 *user*/30s tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 6394ms, rata-rata *min. Respons time* 940ms, rata-rata *max. Respons time* adalah 16349ms dan *error* 0,00%, percobaan 90 *user*/30s dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 17227ms, rata-rata *min. Respons time* 320ms, rata-rata *max. Respons time* adalah 34057ms dan *error* 7,65%, percobaan 100 *user*/30s tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 18436ms, rata-rata *min. Respons time* 134ms, rata-rata *max. Respons time* adalah 36490ms dan *error* 6,80%, bentuk grafik dapat dilihat pada Gambar 4.2.



Gambar 4.2 Grafik rata-rata dengan *Round Robin*

Pada Gambar 4.2 pengujian dengan *Round Robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90 *user*, dan 100 *user*. Rata-rata dari 40 *user* terlihat dari grafik diatas sangat stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil, dan rata-rata dari 80 *user* terlihat grafik yang tidak sedikit stabil, dikarenakan semakin banyaknya *request* dan naik turunnya performa dari perangkat kerasnya itu sendiri, dan rata-rata dari 90 *user* dan 100 *user* terlihat dari grafik cukup signifikan dan mempunyai perbandingan nilai dari *average Response time* yang tidak jauh, Nilainya itu sendiri juga cukup besar dibandingkan dengan pengujian 40, 60, dan 80 *user*. Nilai dari percobaan dengan *Round Robin* ini mempunyai nilai yang lebih sedikit dibanding tanpa *Round Robin*, karena *Round Robin* itu sendiri mempunyai 3 *server* yang sifatnya *Looping* atau berputar. Setiap *server* yang sudah penuh, *request* akan beralih ke *server* selanjutnya.

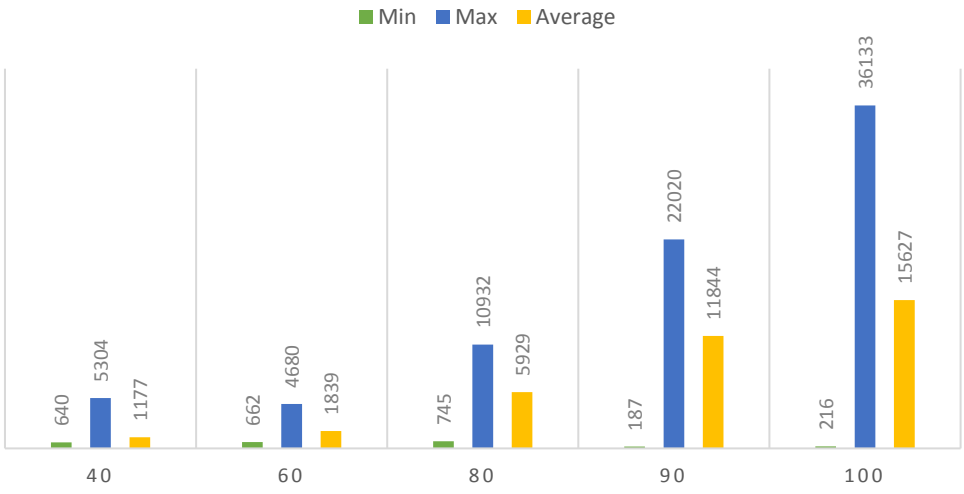
4.1.3 Pengujian *Indexes* tanpa *Round Robin*

Berikut pengujian dengan *Indexes* tanpa *Round Robin* dengan sample 40, 60, 80, 90, 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (*Average, Min, Max*) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian *Indexes* tanpa *Round robin* dapat dilihat pada Tabel 4.3.

Tabel 4.3 Pengujian rata-rata percobaan *Indexes* tanpa *Round Robin*

Percobaan	*Min* (ms)	*Max* (ms)	*Average* (ms)	*Error* (%)
40	640	5.304	1.177	0,00%
60	662	4.680	1.839	0,00%
80	745	10.932	5.929	0,00%
90	187	22.020	11.844	11,11%
100	216	36.133	15.627	9,61%

Pada Tabel 4.3 dengan percobaan 40 *user*/30s dengan *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1177ms, rata-rata *min. Respons time* 640ms, rata-rata *max. Respons time* adalah 5304ms dan *error* 0,00%, percobaan 60 *user*/30s dengan *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1839ms, rata-rata *min. Respons time* 662ms, rata-rata *max. Respons time* adalah 4680ms dan *error* 0,00%, percobaan 80 *user*/30s dengan *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 5929ms, rata-rata *min. Respons time* 745ms, rata-rata *max. Respons time* adalah 10932ms dan *error* 0,00%, percobaan 90 *user*/30s dengan *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 11844ms, rata-rata *min. Respons time* 187ms, rata-rata *max. Respons time* adalah 22020ms dan *error* 11,11%, percobaan 100 *user*/30s dengan *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 15627ms, rata-rata *min. Respons time* 216ms, rata-rata *max. Respons time* adalah 36133ms dan *error* 9,61%, bentuk grafik dapat dilihat pada Gambar 4.3.



Gambar 4.3 Grafik rata-rata dengan *Indexes* tanpa *Round Robin*

Pada Gambar 4.3 pengujian *Indexes* tanpa *Round Robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90*user*, dan 100 *user*. Rata-rata dari 40 *user* terlihat dari grafik diatas sangat stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil, rata-rata dari 80 *user* terlihat grafik yang cukup stabil, dan untuk 90 *user* dan 100 *user* mempunyai grafik rata-rata *Response time* yang nilainya mendekati dan tidak jauh berbeda, grafik menunjukkan sangat stabil. 90 dan 100 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single server*.

Nilai dari semua percobaan dengan *Indexes* tanpa *Round Robin* ini mempunyai nilai yang cukup tinggi namun memiliki grafik yang rata-rata cukup stabil karena menambahkan metode *Indexing*. Nilai *Min. Response Time* dan *Max. Response Time* juga cukup rendah, stabil dan tidak jauh dari rata-rata masing – masing parameter. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROWID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel. Di penelitian ini tabel "*name*" dipilih sebagai ROWID, sehingga menghindari terjadinya *full table-scan*.

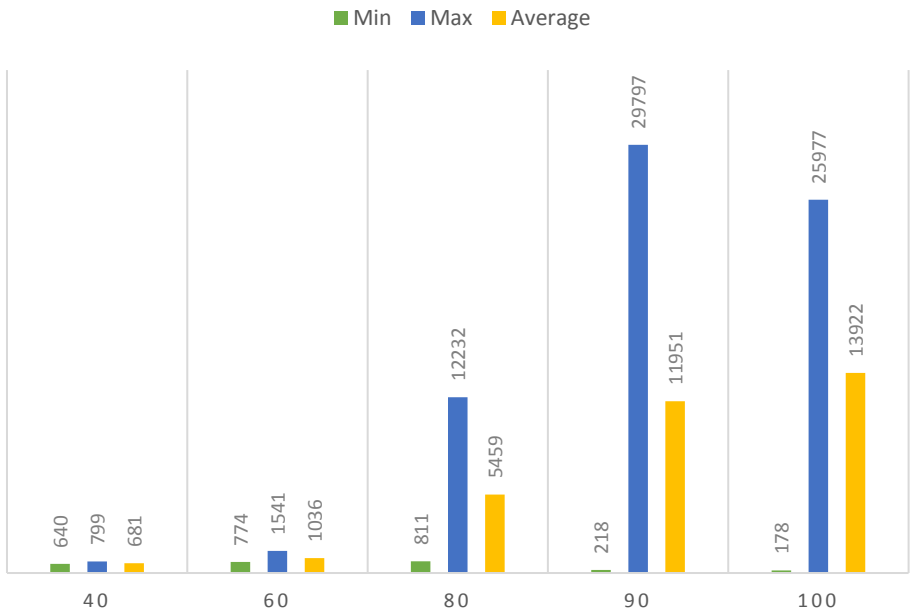
4.1.4 Pengujian *Indexes* dengan *Round Robin*

Berikut pengujian dengan *Indexes* dengan *Round Robin* dengan sample 40, 60, 80, 90, 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (*Average, Min, Max*) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian *Indexes* dengan *Round robin* dapat dilihat pada Tabel 4.4.

Tabel 4.4 Pengujian rata-rata percobaan *Indexes* dengan *Round Robin*

Percobaan	***Min* (ms)**	***Max* (ms)**	***Average* (ms)**	***Error* (%)**
40	640	799	681	0,00%
60	774	1.541	1.036	0,00%
80	811	12.232	5.459	0,00%
90	218	29.797	11.951	5,11%
100	178	25.977	13.922	6,02%

Pada Tabel 4.4 bahwa dengan percobaan 40 *user*/30s dengan *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 681ms, rata-rata *min. Respons time* 640ms, rata-rata *max. Respons time* adalah 799ms dan *error* 0,00%, percobaan 60 *user*/30s dengan *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1036ms, rata-rata *min. Respons time* 774ms, rata-rata *max. Respons time* adalah 1541ms dan *error* 0,00%, percobaan 80 *user*/30s dengan *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 5459ms, rata-rata *min. Respons time* 811ms, rata-rata *max. Respons time* adalah 12232ms dan *error* 0,00%, percobaan 90 *user*/30s dengan *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 11951ms, rata-rata *min. Respons time* 218ms, rata-rata *max. Respons time* adalah 29797ms dan *error* 5,11%, percobaan 100 *user*/30s dengan *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 13922ms, rata-rata *min. Respons time* 178ms, rata-rata *max. Respons time* adalah 25977ms dan *error* 6,02%, bentuk grafik dapat dilihat pada Gambar 4.4.



Gambar 4.4 Grafik rata-rata *Indexes* dengan *Round Robin*

Pada Gambar 4.4 pengujian *Indexes* dengan *Round Robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90 *user* dan 100 *user*. Rata-rata dari 40 *user* terlihat

dari grafik diatas sangat stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil, rata-rata dari 80 *user* terlihat grafik yang sangat stabil, dan untuk 90 *user* dan 100 *user* terlihat grafik sangat stabil dibandingkan dengan pengujian sebelumnya, nilai *error* dari 90 *user* dan 100 *user* ini cukup kecil yang menandakan *request* dan *Response* tersebut mempunyai tingkat keberhasilan yang tinggi. 90 dan 100 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single server*.

Nilai dari semua percobaan dengan *Indexes* dengan *Round Robin* ini mempunyai nilai yang sedikit lebih rendah dibandingkan dengan pengujian *Indexs* tanpa *Round Robin*, karena *request* distribusi data menggunakan *Load balancing Round Robin* yang membuat nilai tersebut rendah dan di pengujian ini juga menggunakan *Indexing.* Nilai *Min. Response Time* dan *Max. Response Time* juga cukup rendah, stabil dan tidak jauh rata-rata dari masing-masing parameter. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut.

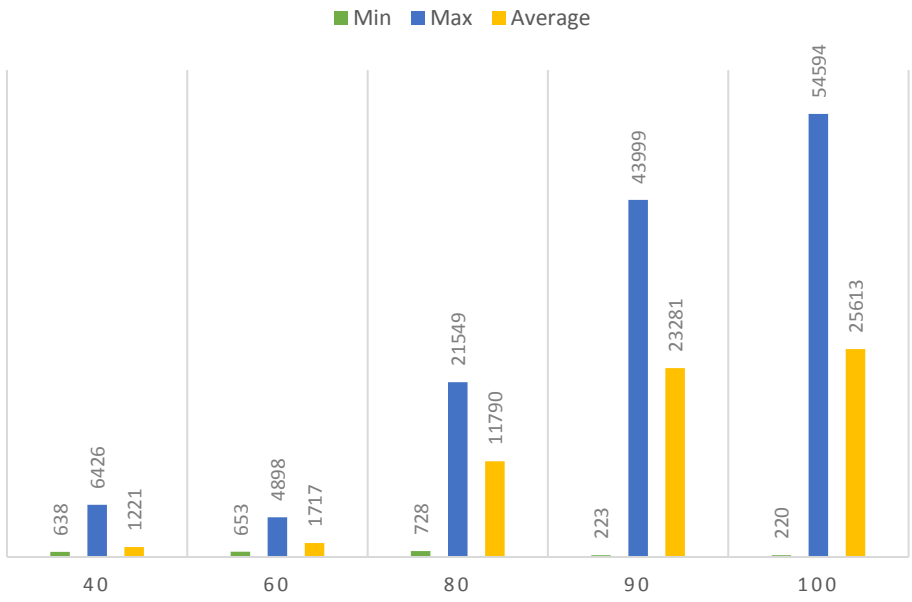
4.1.5 Pengujian tanpa *Indexes* tanpa *Round Robin*

Berikut pengujian tanpa *Indexes* tanpa *Round Robin* dengan sample 40, 60, 80, 90, 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (*Average, Min, Max*) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian tanpa *Indexes* tanpa *Round robin* dapat dilihat pada Tabel 4.5.

Tabel 4.5 Pengujian rata-rata percobaan tanpa *Indexes* tanpa *Round Robin*

Percobaan	*Min* (ms)	*Max* (ms)	*Average* (ms)	*Error* (%)
40	638	6.426	1.221	0,00%
60	653	4.898	1.717	0,00%
80	728	21.549	11.790	0,00%
90	223	43.999	23.281	18,25%
100	220	54.594	25.613	21,08%

Pada Tabel 4.5 dengan percobaan 40 *user*/30s tanpa *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1221ms, rata-rata *min. Respons time* 638ms, rata-rata *max. Respons time* adalah 6426ms dan *error* 0,00%, percobaan 60 *user*/30s tanpa *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1717ms, rata-rata *min. Responsse time* 653ms, rata-rata *max. Respons time* adalah 4898ms dan *error* 0,00%, percobaan 80 *user*/30s tanpa *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 11790ms, rata-rata *min. Responsse time* 728ms, rata-rata *max. Responsse time* adalah 21549ms dan *error* 0,00%, percobaan 90 *user*/30s tanpa *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 23281ms, rata-rata *min. Response time* 223ms, rata-rata *max. Response time* adalah 43999ms dan *error* 18,25%, percobaan 100 *user*/30s tanpa *Indexes* tanpa *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 25613ms, rata-rata *min. Response time* 220ms, rata-rata *max. Response time* adalah 54594ms dan *error* 21,08%, bentuk grafik dapat dilihat pada Gambar 4.5.



Gambar 4.5 Grafik rata-rata tanpa *Indexes* tanpa *Round Robin*

Pada Gambar 4.5 pengujian tanpa *Indexes* tanpa *Round Robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90 *user*, dan 100 *user*. Rata-rata dari 40

user terlihat dari grafik diatas stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil, rata-rata dari 80 *user* terlihat grafik yang kurang stabil, dan untuk 90 *user* dan 100 *user* menunjukkan grafik yang tidak signifikan dan nilai dari rata-rata *Respons time* sangat tinggi. 90 dan 100 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single server*.

Nilai dari semua percobaan tanpa *Indexes* tanpa *Round Robin* ini mempunyai nilai yang sangat tinggi karena tidak memakai *Indexes* dan hanya memakai single *server* saja. Nilai *Min. Response Time* dilihat dari grafik cukup stabil namun untuk *Max. Response Time* mempunyai nilai yang sangat tidak stabil dan di setiap percobaannya selalu mendapatkan nilai yang jauh dan berbeda. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel. Di penelitian ini tabel "*name*" dipilih sebagai ROW ID sehingga menghindari terjadinya *full table-scan*.

4.1.6 Pengujian tanpa *Indexes* dengan *Round Robin*

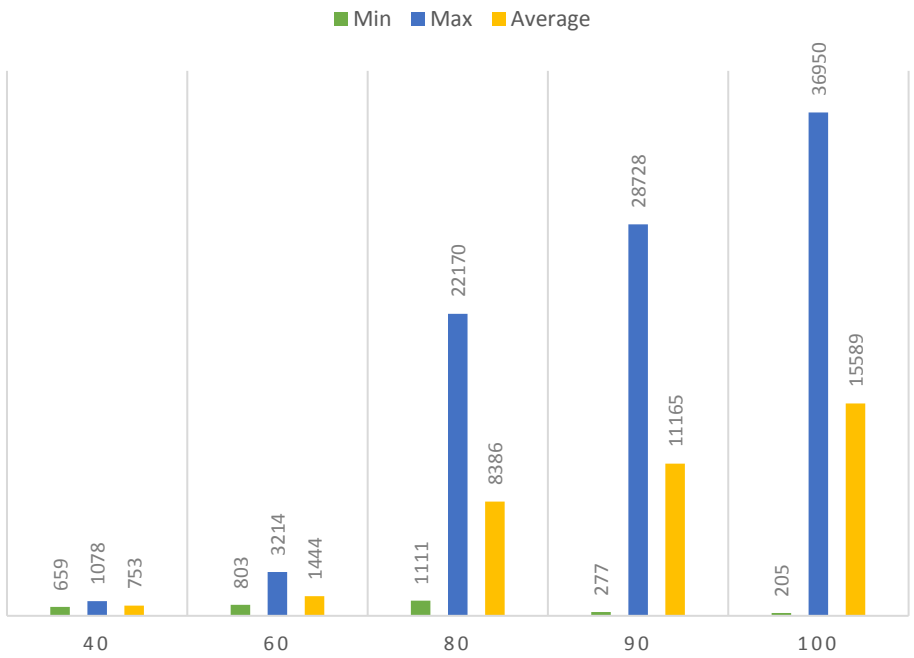
Berikut pengujian tanpa *Indexes* dengan *Round Robin* dengan sample 40, 60, 80, 90, 100 *user* dengan waktu per 30s dan melakukan 10 kali percobaan dengan parameter *Respons time* (Average, Min, Max) dan *error*. Tabel rata-rata dengan 10 kali percobaan pengujian tanpa *Indexes* dengan *Round robin* dapat dilihat pada Tabel 4.6.

Tabel 4.6 Pengujian rata-rata percobaan tanpa *Indexes* dengan *Round Robin*

Percobaan	*Min* (ms)	*Max* (ms)	*Average* (ms)	*Error* (%)
40	659	1.078	753	0,00%
60	803	3.214	1.444	0,00%
80	1111	22.170	8.386	0,00%
90	277	28.728	11.165	8,79%
100	205	36.950	15.589	8,50%

Pada Tabel 4.6 dengan percobaan 40 *user*/30s tanpa *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 753ms,

rata-rata *min. Response time* 659ms, rata-rata *max. Response time* adalah 1078ms dan *error* 0,00%. Dapat dilihat dari Tabel 4.27 dan Gambar 4.27 bahwa dengan percobaan 60 *user*/30s tanpa *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 1444ms, rata-rata *min. Response time* 803ms, rata-rata *max. Response time* adalah 3214ms dan *error* 0,00%, percobaan 80 *user*/30s tanpa *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 8386ms, rata-rata *min. Response time* 1111ms, rata-rata *max. Response time* adalah 22170ms dan *error* 0,00%, percobaan 90 *user*/30s tanpa *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 11165ms, rata-rata *min. Response time* 277ms, rata-rata *max. Response time* adalah 28728ms dan *error* 8,79%, percobaan 100 *user*/30s tanpa *Indexes* dengan *Round Robin* mendapatkan rata-rata *Respons time* dari 10 kali percobaan adalah 15589ms, rata-rata *min. Response time* 205ms, rata-rata *max. Response time* adalah 36950ms dan *error* 8,50%, bentuk grafik dapat dilihat pada Gambar 4.6.



Gambar 4.6 Grafik rata-rata tanpa *Indexes* dengan *Round Robin*

Pada Gambar 4.6 pengujian tanpa *Indexes* dengan *Round Robin* dengan 5 kategori yaitu 40 *user*, 60 *user*, 80 *user*, 90 *user* dan 100 *user*. Rata-rata dari 40

user terlihat dari grafik diatas stabil ketika melakukan 10 kali percobaan secara berurutan, kemudian rata-rata dari 60 *user* terlihat dari grafik diatas cukup stabil, rata-rata dari 80 *user* terlihat grafik yang cukup stabil, untuk 90 *user* dan 100 *user* terlihat grafik yang cukup stabil, namun mempunyai nilai *error* sebesar 8,50% dan 8,79%. 90 dan 100 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single server*.

Nilai dari semua percobaan tanpa *Indexes* dengan *Round Robin* ini mempunyai nilai yang cukup rendah karena memakai *Round Robin* walaupun tidak memakai *Indexes*. Nilai rata-rata (*Average)* dari setiap percobaan sangat rendah dibandingkan dengan pengujian tanpa *Indexes* tanpa *Round Robin*. Nilai *Min. Response Time* dilihat dari grafik walaupun sedikit tidak stabil tetapi mempunyai nilai yang rendah, begitu juga untuk *Max. Response Time* jika dibandingkan dengan percobaan tanpa *Indexes* tanpa *Round Robin*. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut.

4.2 Analisis Grafik Parameter *Average Respons Time*

Dari hasil pengujian yang dilakukan oleh peneliti dengan pengujian *Average Respons Time* sebagai waktu rata-rata yang diambil untuk setiap permintaan masukan dan keluaran yang dihasilkan dan parameter dari 40 *user*, 60 *user*, 80 *user*, 90 *user*, dan 100 *user*. Nilai ini diambil dari masing-masing pengujian Tanpa *Round Robin*, dengan *Round Robin*, *Indexes* tanpa *Round Robin*, *Indexes* dengan *Round Robin*, tanpa *Indexes* tanpa *Round Robin*, dan tanpa *Indexes* dengan *Round Robin*, maka dari itu diperoleh data dalam bentuk grafik sebagai berikut:

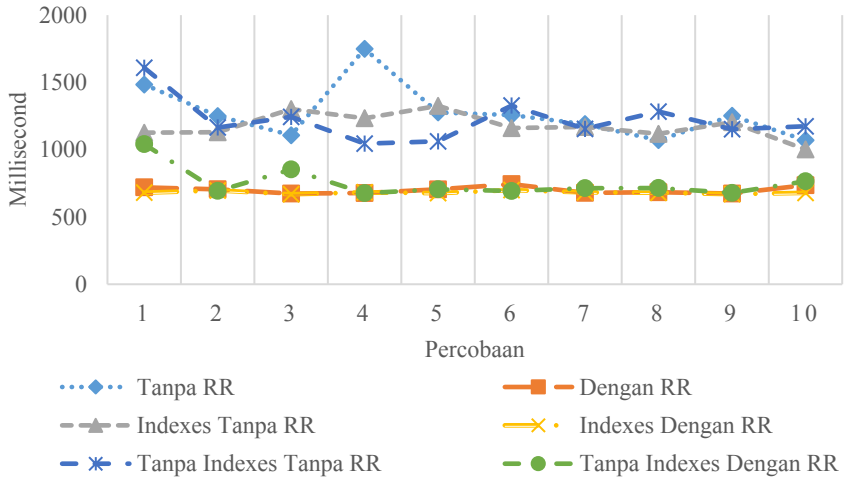
a. *Average Respons Time* 40 *user*/30s

Berikut tabel hasil dari total rata-rata 6 pengujian dengan *sample* 40 *user*/30s dengan parameter *Average Respons time* (*Average, Min, Max*) dan *error* yang terdapat pada Tabel 4.7.

Tabel 4.7 Pengujian *Average respons time* 40 *user*/30s

Percobaan	Tanpa RR (ms)	Dengan RR (ms)	Indexes Tanpa RR (ms)	Indexes Dengan RR (ms)	Tanpa Indexes Tanpa RR (ms)	Tanpa Indexes Dengan RR (ms)
1	1.484	720	1.126	681	1.609	1041
2	1.250	706	1.130	697	1.165	694
3	1.107	674	1.302	669	1.244	853
4	1.749	678	1.234	683	1.045	677
5	1.277	705	1.326	677	1.062	706
6	1.257	744	1.161	700	1.328	693
7	1.190	679	1.169	683	1.155	714
8	1.071	684	1.118	680	1.283	715
9	1.252	674	1.204	670	1.152	679
10	1.070	736	1.002	679	1.174	765

Pada Tabel 4.7 *Average Respons Time* dari parameter 40 *user*/30s dan 10 kali percobaan dengan pengujian *Indexes* dengan *Round Robin* mempunyai nilai yang sangat kecil yaitu 669ms yang artinya semakin kecil nilai itu, semakin baik. Untuk nilai yang paling besar dengan pengujian Tanpa *Round Robin* mempunyai nilai 1749ms yang berbeda sedikit dengan tanpa *Indexes* tanpa *Round Robin* karena sebelumnya menggunakan *Indexes* dan menggunakan *Round Robin*. Untuk grafik data dapat dilihat pada Gambar 4.7.



Gambar 4.7 Grafik rata-rata pengujian dari parameter 40 *user*/30s

Pada Gambar 4.7 Nilai dari semua percobaan dengan *Indexes* dengan *Round Robin* ini mempunyai nilai yang sedikit lebih rendah dibandingkan dengan

pengujian *Indexs* tanpa *Round Robin*, karena *request* distribusi data menggunakan *Load balancing Round Robin* yang membuat nilai tersebut rendah dan di pengujian ini juga menggunakan *Indexing.* Nilai *Min. Response Time* dan *Max. Response Time* juga cukup rendah, stabil dan tidak jauh rata-rata dari masing – masing parameter. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel. Di penelitian ini tabel "*name*" dipilih sebagai ROW ID sehingga menghindari terjadinya *full table-scan*.

b. *Average Respons Time* 60 *user*/30s

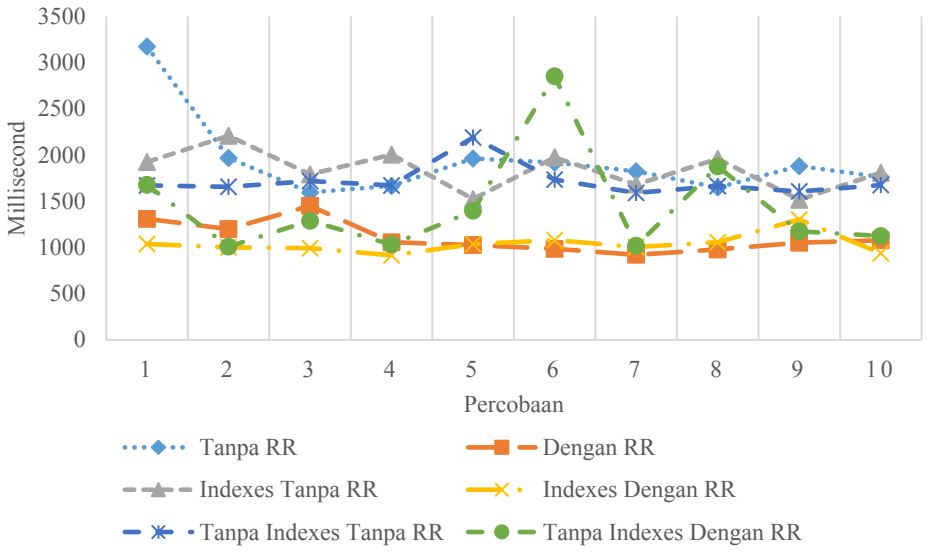
Berikut tabel hasil dari total rata-rata 6 pengujian dengan *sample* 60 *user*/30s dengan parameter *Average Respons time* (*Average, Min, Max*) dan *error* yang terdapat pada Tabel 4.8.

Tabel 4.8 Pengujian *Average respons time* 60 *user*/30s

Percobaan	Tanpa RR (ms)	Dengan RR (ms)	Indexes Tanpa RR (ms)	Indexes Dengan RR (ms)	Tanpa Indexes Tanpa RR (ms)	Tanpa Indexes Dengan RR (ms)
1	3.173	1.309	1.924	1.038	1.671	1.677
2	1.969	1.199	2.208	1.002	1.656	1.009
3	1.595	1.450	1.791	992	1.718	1.288
4	1.666	1.056	2.004	915	1.671	1.031
5	1.962	1.026	1.524	1.036	2.190	1.396
6	1.918	985	1.975	1.079	1.734	2.851
7	1.824	920	1.680	1.005	1.590	1.019
8	1.649	978	1.959	1.056	1.665	1.876
9	1.880	1.051	1.517	1.302	1.606	1.175
10	1.761	1.078	1.808	938	1.675	1.124

Pada Tabel 4.8 dapat dilihat lebih lengkapnya pada lampiran A, dianalisis bahwa *Average Respons Time* dari parameter 60 *user*/30s dan 10 kali percobaan dengan pengujian *Indexes* dengan *Round Robin* mempunyai nilai yang sangat kecil yaitu 669ms yang artinya semakin kecil nilai itu, semakin baik. Untuk nilai yang paling besar dengan pengujian Tanpa *Round Robin* mempunyai nilai 3173ms berbeda sedikit dengan tanpa *Indexes* dengan *Round Robin* yaitu 2851ms karena pengujian ini sebelumnya menggunakan *Indexes*. Nilai dari semua percobaan

dengan *Indexes* dengan *Round Robin* ini mempunyai nilai yang sedikit lebih rendah dibandingkan dengan pengujian *Indexs* tanpa *Round Robin*, karena *request* distribusi data menggunakan *Load balancing Round Robin* yang membuat nilai tersebut rendah dan di pengujian ini juga menggunakan *Indexing*. Nilai *Min. Response Time* dan *Max.* Untuk gambar grafiknya bisa dilihat pada Gambar 4.8.



Gambar 4.8 Grafik rata-rata pengujian dari parameter 60 *user*/30s

Pada Gambar 4.8 alur grafik *Response Time* juga cukup rendah, stabil dan tidak jauh rata-rata dari masing – masing parameter. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel. Di penelitian ini tabel "*name*" dipilih sebagai ROW ID sehingga menghindari terjadinya *full table-scan*.

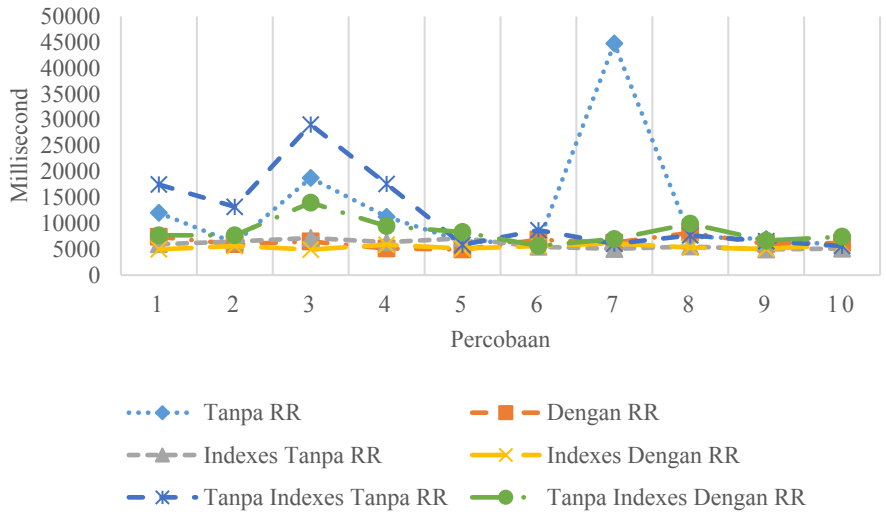
c. *Average Respons Time* 80 *user*/30s

Berikut tabel hasil dari total rata-rata 6 pengujian dengan *sample* 80 *user*/30s dengan parameter *Average Respons time* (*Average, Min, Max*) dan *error* yang terdapat pada Tabel 4.9.

Tabel 4.9 Pengujian *Average respons time* 80 *user*/30s

Percobaan	Tanpa RR (ms)	Dengan RR (ms)	Indexes Tanpa RR (ms)	Indexes Dengan RR (ms)	Tanpa Indexes Tanpa RR (ms)	Tanpa Indexes Dengan RR (ms)
1	12.034	7.403	5.970	4.977	17.519	7.697
2	6.204	5.971	6.501	5.621	13.195	7.697
3	18.783	6.499	7.174	4.939	29.098	14.002
4	11.275	5.143	6.366	5.867	17.622	9.465
5	6.493	4.953	7.113	5.123	5.866	8.322
6	6.344	6.842	5.482	5.633	8.675	5.652
7	44.794	6.263	5.062	6.139	6.066	6.981
8	7.424	8.175	5.539	5.325	7.677	9.942
9	6974	6023	4963	5076	6551	6674
10	5756	6218	5122	5892	5635	7429

Tabel 4.9 dapat dilihat lebih lengkapnya pada halaman lampiran A, dianalisis bahwa *Average Respons Time* dari parameter 80 *user*/30s dan 10 kali percobaan dengan pengujian *Indexes* dengan *Round Robin* mempunyai nilai yang sangat kecil yaitu 4939ms yang artinya semakin kecil nilai itu, semakin baik. Untuk nilai yang paling besar dengan pengujian Tanpa *Round Robin* mempunyai nilai 44794ms, nilai yang cukup mendekati adalah dengan pengujian Tanpa *Indexes* tanpa *Round Robin* yang mempunyai nilai 29098ms. Untuk gambar grafiknya bisa dilihat pada Gambar 4.9.



Gambar 4.9 Grafik rata-rata pengujian dari parameter 80 *user*/30s

Dari Gambar 4.9 dapat dilihat grafik dari nilai semua percobaan dengan *Indexes* dengan *Round Robin* ini mempunyai nilai yang sedikit lebih rendah dibandingkan dengan pengujian *Indexs* tanpa *Round Robin*, karena *request* distribusi data menggunakan *Load balancing Round Robin* yang membuat nilai tersebut rendah dan di pengujian ini juga menggunakan *Indexing.* Nilai *Min. Response Time* dan *Max. Response Time* juga cukup rendah, stabil dan tidak jauh rata-rata dari masing – masing parameter. *Indexing* sendiri merupakan objek struktur data yang tidak bergantung kepada struktur tabel *database*. Setiap indeks terdiri dari nilai kolom dan penunjuk (atau ROW ID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel. Di penelitian ini tabel "*name*" dipilih sebagai ROW ID sehingga menghindari terjadinya *full table-scan*.

d. *Average Respons Time* 90 *user*/30s

Berikut tabel hasil dari total rata-rata 6 pengujian dengan *sample* 90 *user*/30s dengan parameter *Average Respons time* (*Average, Min, Max*) dan *error* yang terdapat pada Tabel 4.10.

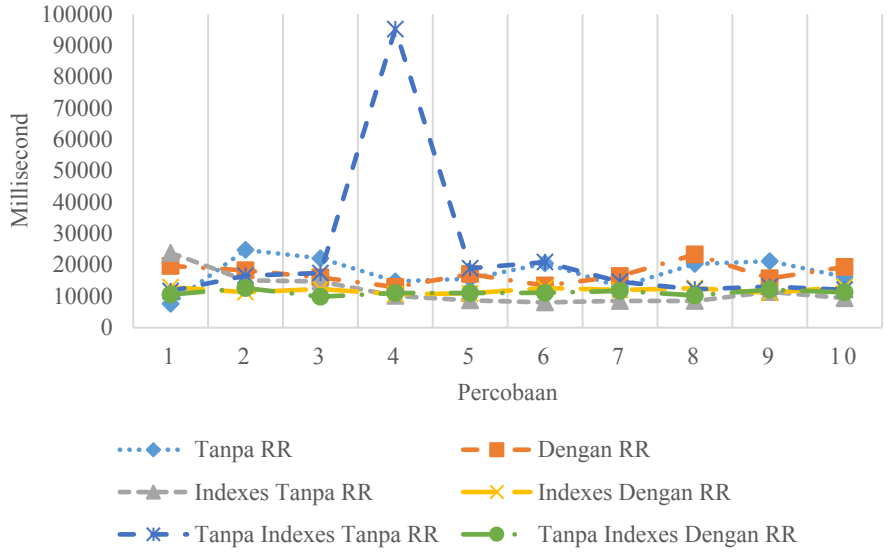
1

Tabel 4.10 Pengujian *Average respons time* 90 *user*/30s

Percobaan	Tanpa RR (ms)	Dengan RR (ms)	Indexes Tanpa RR (ms)	Indexes Dengan RR (ms)	Tanpa Indexes Tanpa RR (ms)	Tanpa Indexes Dengan RR (ms)
1	7.598	19.660	23.867	12.912	11.673	10.557
2	24.793	18.254	15.101	11.316	16.591	12.627
3	22.115	16.013	14.731	12.339	17.435	9.866
4	14.773	12.976	10.137	10.653	95.231	11.049
5	15.623	17.092	8.721	10.938	18.919	11.064
6	20.515	13.421	8.056	12.658	20.915	11.104
7	13.043	16.450	8.536	12.002	14.633	11.696
8	20.304	23.341	8.504	12.602	12.242	10.287
9	21.199	15.721	11.387	11.085	13.080	12.139
10	16.152	19.351	9.405	13.010	12.098	11.262

Pada Tabel 4.10 dapat dilihat lebih lengkapnya pada halaman lampiran A, dianalisis bahwa *Average Respons Time* dari parameter 90 user/30s dan 10 kali percobaan dengan pengujian tanpa *Round Robin* mempunyai nilai yang sangat kecil yaitu 7598ms, namun untuk pengujian selanjutnya mempunyai nilai yang

sangat tinggi. Berbeda dengan *Indexes* dengan *Round Robin*, walaupun mempunyai nilai yang sedikit lebih tinggi dari tanpa *Round Robin*, pengujian ini mempunyai nilai yang cukup stabil di setiap percobaannya. Untuk nilai yang paling besar dengan pengujian Tanpa *Indexes* tanpa *Round Robin* yang mempunyai nilai 95231ms. Untuk gambar grafiknya bisa dilihat pada Gambar 4.10.



Gambar 4.10 Grafik rata-rata pengujian dari parameter 90 *user*/30s

Dari Gambar 4.10 dapat dianalisis bahwa *Average Respons Time* dari parameter 90 *user*/30s dan 10 kali percobaan dengan pengujian tanpa *Round Robin* mempunyai nilai yang sangat kecil yaitu 7598ms, namun untuk pengujian selanjutnya mempunyai nilai yang sangat tinggi. Berbeda dengan *Indexes* dengan *Round Robin*, walaupun mempunyai nilai yang sedikit lebih tinggi dari tanpa *Round Robin,* pengujian ini mempunyai nilai yang cukup stabil di setiap percobaannya. Untuk nilai yang paling besar dengan pengujian Tanpa *Indexes* tanpa *Round Robin* yang mempunyai nilai 95231ms. Analisis 90 *user*/30s sedikit lebih berbeda dari pengujian 40 *user*, 60 *user*, dan 80 *user*, karena di pengujian 90 *user*/30s ini mempunyai *error* disetiap pengujian, maka nilai *Respons Time* tidak menentu, yang seharusnya *Indexes* dengan *Round Robin* mempunyai nilai yang paling kecil dibandingkan dengan pengujian yang lain. 90 *user* mempunyai nilai *error* karena *Hardware* yang digunakan kurang optimal. Nilai dari percobaan tanpa *Round robin* cukup tinggi karena hanya menggunakan satu *server* atau biasa di sebut *single*

server. Nilai dari semua percobaan dengan *Indexes* dengan *Round Robin* ini mempunyai nilai yang sedikit lebih rendah dibandingkan dengan pengujian *Indexs* tanpa *Round Robin*, karena *request* distribusi data menggunakan *Load balancing Round Robin* yang membuat nilai tersebut rendah dan di pengujian ini juga menggunakan *Indexing.* Nilai *Min. Response Time* dan *Max. Response Time* juga cukup rendah, stabil dan tidak jauh rata-rata dari masing – masing parameter.

e. *Average Respons Time* 100 *user*/30s

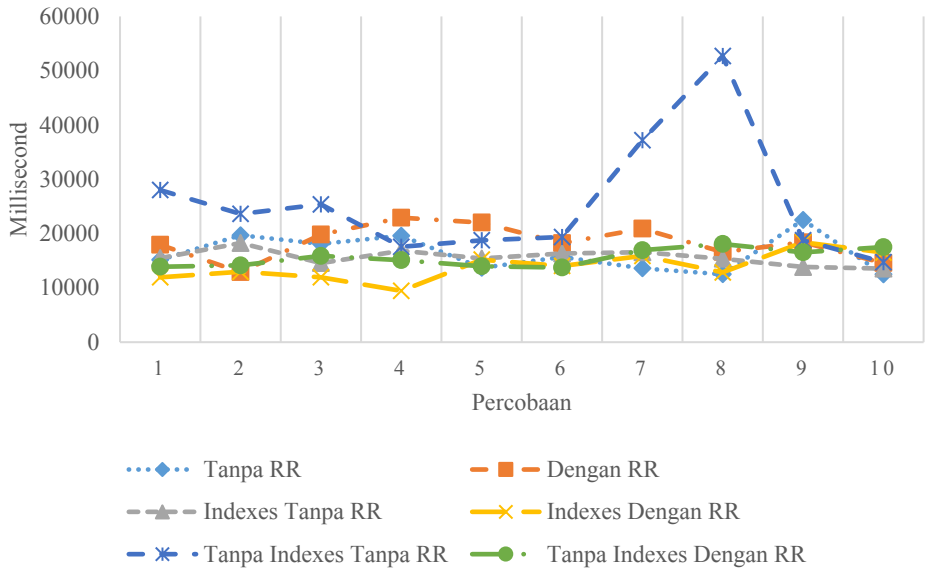
Berikut tabel hasil dari total rata-rata 6 pengujian dengan *sample* 100 *user*/30s dengan parameter *Average Respons time* (*Average, Min, Max*) dan *error* pada Tabel 4.11.

Tabel 4.11 Pengujian *Average respons time* 100 *user*/30s

Percobaan	Tanpa RR (ms)	Dengan RR (ms)	Indexes Tanpa RR (ms)	Indexes Dengan RR (ms)	Tanpa Indexes Tanpa RR (ms)	Tanpa Indexes Dengan RR (ms)
1	15.169	17.932	15.483	11.946	28.027	13.905
2	19.685	12.899	18.262	12.966	23.624	14.155
3	18.112	19.820	14.575	11.952	25.379	15.894
4	19.596	22.930	16.848	9.452	17.568	15.114
5	13.688	22.033	15.410	15.202	18.768	13.986
6	15.700	18.239	16.352	14.044	19.372	13.748
7	13.598	20.922	16.565	15.893	37.185	16.951
8	12.507	16.606	15.370	12.854	52.690	18.082
9	22.519	18.399	13.856	18.405	18.780	16.545
10	12.489	14.580	13.554	16.511	14.737	17.515

Dari Tabel 4.11 dapat dilihat lebih lengkapnya pada halaman lampiran A, dianalisis bahwa *Average Respons Time* dari parameter 100 *user*/30s dan percobaan 10 kali dengan pengujian *Indexes* dengan *Round Robin* mempunyai nilai yang sangat kecil yaitu 9452ms, sangat bebeda sedikit dengan tanpa *Round Robin* yang mempunyai nilai 12489ms. Untuk nilai yang paling besar dengan pengujian Tanpa *Indexes* tanpa *Round Robin* mempunyai nilai 52690ms. Analisis 100 *user*/30s sedikit lebih berbeda dari pengujian 40 *user*, 60 *user*, dan 80 *user*, karena di pengujian 100 *user*/30s ini mempunyai *error* disetiap pengujian, maka nilai *Respons Time* tidak menentu. *Indexes* tanpa *Round Robin* di pengujian ini justru

mempunyai nilai yang cukup tinggi dibandingkan dengan tanpa *Round Robin* yang seharusnya nilai *Indexes* tanpa *Round Robin* lebih kecil nilainya, karena di percobannya menggunakan *Indexes* walaupun tidak menggunakan *Round Robin*. Untuk gambar grafiknya bisa dilihat pada Gambar 4.11.



Gambar 4.11 Grafik rata-rata pengujian dari parameter 100 *user*/30s

Dari Gambar 4.11 dapat dianalisis bahwa *Average Respons Time* dari parameter 100 *user*/30s dan percobaan 10 kali dengan pengujian *Indexes* dengan *Round Robin* mempunyai nilai yang sangat kecil yaitu 9452ms, sangat bebeda sedikit dengan tanpa *Round Robin* yang mempunyai nilai 12489ms. Untuk nilai yang paling besar dengan pengujian Tanpa *Indexes* tanpa *Round Robin* mempunyai nilai 52690ms. Analisis 100 *user*/30s sedikit lebih berbeda dari pengujian 40 *user*, 60 *user*, dan 80 *user*, karena di pengujian 100 *user*/30s ini mempunyai *error* disetiap pengujian, maka nilai *Respons Time* tidak menentu.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan, diperoleh beberapa kesimpulan sebagai berikut:

1. Secara keseluruhan, kecepatan respon setiap *request* bergantung pada spesifikasi *server*, jaringan yang dilalui. Kedua faktor tersebut saling bergantung dan tidak bisa dipisahkan. Jika *client request* memliki jaringan yang cepat, namun *server* yang diakses memliki spesifikasi yang rendah maka respon yang dilakukan *server* ke *client* kurang responsif.
2. Kecepatan *Response time* dari *server* yang sudah diimplementasikan dengan *Load Balance* dan penambahan *Indexes* mempunyai nilai lebih rendah dan lebih stabil dibandingkan dengan *single server* dan tidak menambahkan *Indexes* yang mempunyai nilai lebih tinggi dan statistik yang tidak beraturan.

5.2 Saran

Penelitian yang telah dilakukan masih memiliki banyak kekurangan adapun saran yang dapat diberikan dari hasil penelitian ini, yaitu sebagai berikut:

1. Mengimplementasikan metode yang lebih akurat agar pembagian kinerja CPU di metode *Round Robin* sama rata menggunakan sistem *time sharing* atau pembagian waktu secara adil pada setiap proses yang akan dieksekusi atau dijalankan, yakni dengan menggunakan *static quantum time* .
2. Menerapkan struktur penyimpanan data yang tersimpan dan tersusun rapi pada ruang penyimpanan komputer secara sistematik sehingga mudah saat diakses oleh program untuk mencari keberadaan suatu data yang lebih rapi dan efisien.

DAFTAR PUSTAKA

[1] Ardianto F, B. Alfaresi, dan A. Darmadi. 2018. *Rancang Bangun Load Balancing Dua Internet Service Provider (Isp) Berbasis Mikrotik*. J. Surya Energy, Vol. 3, no. 1. p. 198. doi: 10.32502/jse.v3i1.1232.

[2] Verborgh R., D. Arndt, S. V. Hoecke. 2017. *Theory Pract. Log. Program.* Vol. 17, no. 1. pp. 1–48. doi: 10.1017/S1471068416000016.

[3] Komaruddin A. M, D. M. Sipitorini, dan P. Rispian. 2019. *Load Balancing dengan Metode Round Robin Untuk Pembagian Beban Kerja Web Server*. Siliwangi. Vol. 5, no. 2. pp. 47–50.

[4] Dani R., dan F. Suryawan. 2017. *Perancangan dan Pengujian Load Balancing dan Failover Menggunakan NginX*. Khazanah Inform. J. Ilmu Komput dan Inform. Vol. 3, no. 1. p. 43. doi: 10.23917/khif.v3i1.2939.

[5] Shershneu dan Oskin. 2020. *Postman Platform For Api Development In The Mobile Application Musicians Of Russia.* Mater. XII Jr. Res. Conf. Vol. 68, no. 1. pp. 128–131.

[6] Zongyu X. U. dan W. Xingxuan. 2018. *A Predictive Modified Round Robin Scheduling algorithm for web server clusters*. pp. 5804–5808.

[7] Vuluvala. 2018. *Proceedings of 2018 IEEE International Conference on Power, Instrumentation, Control and Computing (PICC) : 18th to 20th January 2018*. Int. Conf. Power, Instrumentation, Control Computer. pp. 1–5.

[8] Ghosh S. dan C. Banerjee. 2018. *Dynamic time quantum priority based round robin for load balancing in cloud environment*. Proc. - 2018 4th IEEE Int. Conf. Res. Comput. Intell. Commun. Networks. pp. 33–37. doi: 10.1109/ICRCICN.2018.8718694.

[9] Approach A. D. 2017. *Dynamic Load Balancing in Cloud*. Int. Conf. Networks Adv. Comput. Technology. pp. 162–166.

[10] Bari R. 2019. *Schedulers at different User Mobilty*. pp. 18–20.

[11] Arta Y. 2017. *Penerapan Metode Round Robin Pada Jaringan Multihoming Di Computer Cluster*. It J. Res. Development. Vol. 1, no. 2. pp. 26–35. doi: 10.25299/itjrd.2017.

[12] Nasser H. dan T. Witono. 2017. *Analisis Algoritma Round Robin, Least Connection, Dan Ratio Pada Load Balancing Menggunakan Opnet Modeler*. J. Information. Vol. 12, no. 1. pp. 25–32. doi: 10.21460/inf.2016.121.455.

[13] Yanti S. N. dan E. Rihyanti. 2021. *Penerapan Rest API untuk Sistem Informasi Film Secara Daring*. J. Inform. Univ. Pamulang. Vol. 6, no. 1. p. 195. doi: 10.32493/informatika.v6i1.10033.

[14] Neumann A., N. Laranjeiro, dan J. Bernardino. 2021. *An Analysis of Public REST Web Service APIs*. IEEE Trans. Serv. Computing. Vol. 14, no. 4. pp. 957–970. doi: 10.1109/TSC.2018.2847344.

[15] Wang X., Q. Sun, dan J. Liang. 2020. *JSON-LD based web API semantic annotation considering distributed knowledge*. IEEE Access. Vol. 8. pp. 197203–197221. doi: 10.1109/ACCESS.2020.3034937.

[16] Phi N. X., C. T. Tin, L. N. Ky Thu, dan T. C. Hung. 2018. *Proposed load balancing algorithm to reduce response time and processing time on cloud computing*. Int. J. Comput. Networks Communication. Vol. 10, no. 3. pp. 87–98. doi: 10.5121/IJCNC.2018.10307.

[17] Andani M. R. 2022. *Belajar Golang Beserta Kelebihan dan Framework yang Digunakan*. Sekawan Media. https://www.sekawanmedia.co.id/blog/belajar-golang/. (accessed Jul. 23, 2022).

[18] Anonim. 2021. *Golang Program*. Kotakkode, https://kotakode.com/blogs/7759/Golang-Fundamental--%3A-Basic-Structure-Go-Program.

[19] Anonim. 2022. *Restful Webservices*. Phppot.com. https://phppot.com/php/php-restful-web-service/. (accessed Jul. 23, 2022).

[20] Segura, J. A. Parejo, J. Troya, dan A. Ruiz-Cortes. 2018. *Metamorphic testing of RESTful Web APIs*. IEEE Trans. Softw. Eng. Vol. 44, no. 11. pp. 1083–1099, doi: 10.1109/TSE.2017.2764464.

[21] Tanaem P. F, D. Manongga, dan A. Iriani. 2016. *RESTFul Web Service Untuk Sistem Pencatatan Transaksi Studi Kasus PT . XYZ*. Vol. 2, no. April.

[22] Kumar P, A. Gurtov, J. Iinatti, M. Ylianttila, dan M. Sain. 2019. *An authentication based scheme for applications using JSON web token*. IEEE Sens. J. Vol. 16, no. 1. pp. 254–264. doi: 10.1109/JSEN.2015.2475298.

[23] Eyada M. M., W. Saber, M. M. El Genidy, dan F. Amer. 2020. *Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments*. IEEE Access. Vol. 8. pp. 110656–110668, doi: 10.1109/ACCESS.2020.3002164.

[24] Takalelumang M. F., Y. D. Y. Rindengan, dan A. Sambul. 2018. *Aplikasi E-Agri Kabupaten Minahasa Selatan*. J. Tek. Informasi. Vol. 13, no. 1. doi: 10.35793/jti.13.1.2018.20189.

[25] Anonim. 2020. *Postman*. Medium.com, https://medium.com/@novancimol12/postman-4f181d625fe1 (accessed Jul. 23, 2022).

[26] Pennino D., M. Pizzonia, dan A. Papi. 2019. *Overlay Indexes: Efficiently Supporting Aggregate Range Queries and Authenticated Data Structures in Off-the-Shelf Databases*. IEEE Access. Vol. 7. pp. 175642–175670. doi: 10.1109/ACCESS.2019.2957346.

[27] Kohli N. dan N. K. Verma. 2018. *MySQL based selection of appropriate indexing technique in hospital system using multiclass SVM*. Int. J. Eng. Sci. Technology. Vol. 2, no. 6. pp. 119–130. doi: 10.4314/ijest.v2i6.63703.

LAMPIRAN

Lampiran A Tabel Percobaan

Tabel Percobaan dengan 40 *user*/30s tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	40	0.00%	683	6109	1484
2	40	0.00%	658	6758	1250
3	40	0.00%	633	5697	1107
4	40	0.00%	670	7066	1749
5	40	0.00%	658	5653	1277
6	40	0.00%	640	4732	1257
7	40	0.00%	651	8794	1190
8	40	0.00%	657	5488	1071
9	40	0.00%	654	8812	1252
10	40	0.00%	640	5461	1070
Rata – rata		**0.00%**	**654ms**	**6457ms**	**1270ms**

Tabel Percobaan dengan 60 *user*/30s tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	60	0.00%	717	6094	3173
2	60	0.00%	772	4571	1969
3	60	0.00%	683	3519	1595
4	60	0.00%	673	4875	1666
5	60	0.00%	699	3685	1962
6	60	0.00%	719	4867	1918
7	60	0.00%	674	3581	1824
8	60	0.00%	670	3416	1649
9	60	0.00%	657	4844	1880
10	60	0.00%	665	4908	1761
Rata-rata		**0.00%**	**692ms**	**4436ms**	**1939ms**

Tabel Percobaan dengan 80 *user*/30s tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	80	0.00%	488	18972	12034
2	80	0.00%	627	11495	6204
3	80	0.00%	714	34411	18783
4	80	0.00%	890	20721	11275
5	80	0.00%	711	12872	6493
6	80	0.00%	946	11020	6344
7	80	5.00%	713	74433	44794
8	80	0.00%	997	15140	7424
9	80	0.00%	631	13113	6974
10	80	0.00%	643	10882	5756
Rata – rata		**0.50%**	**736ms**	**22305ms**	**12608ms**

Tabel Percobaan dengan 90 *user*/30s tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	90	24.44%	126	16134	7598
2	90	14.44%	443	54897	24793
3	90	13.13%	126	40656	22115
4	90	12.22%	168	26525	14773
5	90	10.00%	46	26995	15623
6	90	4.44%	255	36331	20515
7	90	14.44%	227	24874	13043
8	90	3.33%	5	37098	20304
9	90	4.44%	299	37609	21199
10	90	8.89%	40	28417	16152
Rata – rata		**10.98%**	**173ms**	**32953ms**	**17611ms**

Tabe Percobaan dengan 100 *user*/30s tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	100	16.00%	94	30422	15169
2	100	9.00%	94	33179	19685
3	100	11.00%	198	31465	18112
4	100	12.00%	14	37073	19596
5	100	19.00%	30	29180	13688
6	100	17.00%	122	31010	15700
7	100	20.00%	413	26438	13598
8	100	20.00%	463	22873	12507
9	100	12.00%	43	37629	22519
10	100	22.00%	196	23687	12489
Rata – rata		**15.80%**	**188ms**	**33295ms**	**18306ms**

Tabel Percobaan dengan 40 *user*/30s dengan *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	40	0.00%	642	913	720
2	40	0.00%	642	834	706
3	40	0.00%	639	765	674
4	40	0.00%	629	727	678
5	40	0.00%	641	937	705
6	40	0.00%	645	1050	744
7	40	0.00%	639	769	679
8	40	0.00%	637	795	684
9	40	0.00%	646	756	674
10	40	0.00%	649	1164	736
Rata - rata		**0.00%**	**641ms**	**871ms**	**700ms**

Tabel Percobaan dengan 60 *user*/30s dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	60	0.00%	909	2577	1309
2	60	0.00%	814	4154	1199
3	60	0.00%	767	2466	1450
4	60	0.00%	741	1997	1056
5	60	0.00%	740	1567	1026
6	60	0.00%	708	1288	985
7	60	0.00%	735	1355	920
8	60	0.00%	763	1290	978
9	60	0.00%	758	1560	1051
10	60	0.00%	814	1662	1078
Rata - rata		**0.00%**	**775ms**	**1992ms**	**1105ms**

Tabel Percobaan dengan 80 *user*/30s dengan *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	80	0.00%	1045	20940	7403
2	80	0.00%	826	14782	5971
3	80	0.00%	647	18183	6499
4	80	0.00%	823	13349	5143
5	80	0.00%	841	12194	4953
6	80	0.00%	875	17380	6842
7	80	0.00%	1417	17118	6263
8	80	0.00%	940	16973	8175
9	80	0.00%	913	15993	6023
10	80	0.00%	1076	16578	6218
Rata - rata		**0.00%**	**940ms**	**16349ms**	**6349ms**

Tabel Percobaan dengan 90 *user*/30s dengan *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	90	6.52%	333	35282	19660
2	90	6.67%	253	39747	18254
3	90	8.89%	43	30049	16013
4	90	12.22%	60	26154	12976
5	90	7.78%	420	33132	17092
6	90	14.44%	163	29516	13421
7	90	6.67%	394	33721	16450
8	90	0.00%	1259	40880	23341
9	90	7.78%	197	33803	15721
10	90	5.56%	87	38294	19351
Rata - rata		**7.65%**	**320ms**	**34057ms**	**17227ms**

Tabel Percobaan dengan 100 *user*/30s dengan *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	100	7.00%	29	43229	17932
2	100	6.00%	138	32249	12899
3	100	6.00%	147	42922	19820
4	100	8.00%	187	38939	22930
5	100	6.00%	255	32292	22033
6	100	4.00%	141	36029	18239
7	100	10.00%	108	35392	20922
8	100	3.00%	50	32313	16606
9	100	12.00%	24	33243	18399
10	100	6.00%	269	38299	14580
Rata - rata		**6.80%**	**134ms**	**36490ms**	**18436ms**

Tabel Percobaan dengan 40 *user*/30s dengan *Indexes* tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	40	0.00%	645	3080	1126
2	40	0.00%	633	3217	1130
3	40	0.00%	646	8928	1302
4	40	0.00%	644	5296	1234
5	40	0.00%	634	5719	1326
6	40	0.00%	650	5270	1161
7	40	0.00%	637	5179	1169
8	40	0.00%	649	5528	1118
9	40	0.00%	637	5364	1204
10	40	0.00%	632	5462	1002
Rata-rata		**0.00%**	**640ms**	**5304ms**	**1177ms**

Tabel Percobaan dengan 60 *user*/30s dengan *Indexes* tanpa *Round Robin*

Percobaan	Sampel	*Error* %	Min	Max	Average
1	60	0.00%	711	4977	1924
2	60	0.00%	696	5234	2208
3	60	0.00%	676	4687	1791
4	60	0.00%	656	4315	2004
5	60	0.00%	634	4923	1524
6	60	0.00%	627	4784	1975
7	60	0.00%	642	4741	1680
8	60	0.00%	657	4861	1959
9	60	0.00%	665	3440	1517
10	60	0.00%	664	4838	1808
Rata-rata		**0.00%**	**662ms**	**4680ms**	**1839ms**

Tabel Percobaan dengan 80 *user*/30s dengan *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	80	0.00%	640	12047	5970
2	80	0.00%	676	13269	6501
3	80	0.00%	1625	11994	7174
4	80	0.00%	625	11832	6366
5	80	0.00%	647	11661	7113
6	80	0.00%	606	9478	5482
7	80	0.00%	707	8643	5062
8	80	0.00%	665	11008	5539
9	80	0.00%	614	10712	4963
10	80	0.00%	645	8683	5122
Rata-rata		**0.00%**	**745ms**	**10932ms**	**5929ms**

Tabel Percobaan dengan 90 *user*/30s dengan *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	90	16.67%	8	45601	23867
2	90	7.78%	209	25395	15101
3	90	11.11%	462	24177	14731
4	90	12.22%	286	20896	10137
5	90	12.22%	160	17292	8721
6	90	12.22%	20	15596	8056
7	90	10.00%	126	16942	8536
8	90	11.11%	137	17042	8504
9	90	6.67%	324	20303	11387
10	90	11.11%	138	16956	9405
Rata-rata		**11.11%**	**187ms**	**22020ms**	**11844ms**

Tabel Percobaan dengan 100 *user*/30s dengan *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	100	12.00%	255	36676	15483
2	100	11.00%	64	44156	18262
3	100	10.00%	518	35061	14575
4	100	14.00%	390	35589	16848
5	100	7.00%	350	34080	15410
6	100	11.00%	160	37484	16352
7	100	7.00%	252	37909	16565
8	100	13.00%	110	34506	15370
9	100	5.05%	17	32403	13856
10	100	6.00%	44	33472	13554
Rata-rata		**9.61%**	**216ms**	**36133ms**	**15627ms**

Tabel Percobaan dengan 40 *user*/30s dengan *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	40	0.00%	643	740	681
2	40	0.00%	640	919	697
3	40	0.00%	637	764	669
4	40	0.00%	648	793	683
5	40	0.00%	648	758	677
6	40	0.00%	644	944	700
7	40	0.00%	632	853	683
8	40	0.00%	645	748	680
9	40	0.00%	644	716	670
10	40	0.00%	628	760	679
Rata - rata		**0.00%**	**640ms**	**799ms**	**681ms**

Tabel Percobaan dengan 60 *user*/30s dengan *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	60	0.00%	752	1482	1038
2	60	0.00%	760	1753	1002
3	60	0.00%	770	1556	992
4	60	0.00%	764	1095	915
5	60	0.00%	764	1437	1036
6	60	0.00%	773	1820	1079
7	60	0.00%	766	1456	1005
8	60	0.00%	805	1642	1056
9	60	0.00%	787	1984	1302
10	60	0.00%	808	1193	938
Rata - rata		**0.00%**	**774ms**	**1541ms**	**1036ms**

Tabel Percobaan dengan 80 *user*/30s dengan *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	80	0.00%	884	8866	4977
2	80	0.00%	784	12865	5621
3	80	0.00%	803	10324	4939
4	80	0.00%	841	13539	5867
5	80	0.00%	853	9568	5123
6	80	0.00%	720	11815	5633
7	80	0.00%	874	13988	6139
8	80	0.00%	767	14408	5325
9	80	0.00%	814	14841	5076
10	80	0.00%	779	12107	5892
Rata - rata		**0.00%**	**811ms**	**12232ms**	**5459ms**

Tabel Percobaan dengan 90 *user*/30s dengan *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	90	4.44%	307	32813	12912
2	90	6.67%	255	26685	11316
3	90	4.44%	232	31639	12339
4	90	5.56%	171	27905	10653
5	90	6.67%	131	27817	10938
6	90	4.44%	311	30197	12658
7	90	4.44%	235	28335	12002
8	90	3.33%	100	30409	12602
9	90	6.67%	228	28419	11085
10	90	4.44%	212	33756	13010
Rata - rata		**5.11%**	**218ms**	**29797ms**	**11951ms**

Tabel Percobaan dengan 100 *user*/30s dengan *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	100	7.00%	68	21375	11946
2	100	4.21%	60	25761	12966
3	100	6.00%	196	23488	11952
4	100	5.00%	277	21574	9452
5	100	7.00%	291	27990	15202
6	100	5.00%	242	23701	14044
7	100	5.00%	58	26043	15893
8	100	6.00%	226	26675	12854
9	100	7.00%	221	34622	18405
10	100	8.00%	150	28747	16511
Rata - rata		**6.02%**	**178ms**	**25977ms**	**13922ms**

Tabel Percobaan dengan 40 *user*/30s tanpa *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	40	0.00%	629	6777	1609
2	40	0.00%	621	6760	1165
3	40	0.00%	639	8827	1244
4	40	0.00%	646	5475	1045
5	40	0.00%	642	5456	1062
6	40	0.00%	640	6726	1328
7	40	0.00%	639	5459	1155
8	40	0.00%	636	6756	1283
9	40	0.00%	647	5275	1152
10	40	0.00%	641	6750	1174
Rata - rata		**0.00%**	**638ms**	**6426ms**	**1221ms**

Tabel Percobaan dengan 60 *user*/30s tanpa *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	60	0.00%	642	6151	1671
2	60	0.00%	653	6778	1656
3	60	0.00%	660	3291	1718
4	60	0.00%	659	3501	1671
5	60	0.00%	661	5015	2190
6	60	0.00%	638	4310	1734
7	60	0.00%	681	4958	1590
8	60	0.00%	653	6910	1665
9	60	0.00%	652	3304	1606
10	60	0.00%	640	4770	1675
Rata - rata		**0.00%**	**653ms**	**4898ms**	**1717ms**

Tabel Percobaan dengan 80 *user*/30s tanpa *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	80	0.00%	806	28030	17519
2	80	0.00%	767	23601	13195
3	80	0.00%	950	51135	29098
4	80	0.00%	937	33119	17622
5	80	0.00%	620	11274	5866
6	80	0.00%	605	17636	8675
7	80	0.00%	643	11886	6066
8	80	0.00%	689	15983	7677
9	80	0.00%	629	11015	6551
10	80	0.00%	640	11813	5635
Rata - rata		**0.00%**	**728ms**	**21549ms**	**11790ms**

Tabel Percobaan dengan 90 *user*/30s tanpa *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	90	16.67%	122	26430	11673
2	90	10.00%	24	32458	16591
3	90	9.20%	457	30107	17435
4	90	75.56%	35	182145	95231
5	90	12.22%	497	32284	18919
6	90	14.44%	125	43185	20915
7	90	8.89%	58	25599	14633
8	90	11.11%	377	22225	12242
9	90	11.11%	53	23083	13080
10	90	13.33%	484	22477	12098
Rata - rata		**18.25%**	**223ms**	**43999ms**	**23281ms**

Tabel Percobaan dengan 100 *user*/30s tanpa *Indexes* tanpa *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	100	9.00%	286	60447	28027
2	100	12.00%	12	43689	23624
3	100	13.75%	70	57338	25379
4	100	15.00%	182	33473	17568
5	100	11.00%	349	33037	18768
6	100	15.00%	89	38534	19372
7	100	52.00%	185	126148	37185
8	100	57.00%	440	93249	52690
9	100	11.00%	299	32305	18780
10	100	15.00%	291	27722	14737
Rata - rata		**21.08%**	**220ms**	**54594ms**	**25613ms**

Tabel Percobaan dengan 40 *user*/30s tanpa *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	40	0.00%	783	1417	1041
2	40	0.00%	634	844	694
3	40	0.00%	644	1532	853
4	40	0.00%	639	808	677
5	40	0.00%	651	899	706
6	40	0.00%	646	921	693
7	40	0.00%	659	996	714
8	40	0.00%	651	991	715
9	40	0.00%	648	745	679
10	40	0.00%	644	1635	765
Rata - rata		**0.00%**	**659ms**	**1078ms**	**753ms**

Tabel Percobaan dengan 60 *user*/30s tanpa *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	60	0.00%	811	4507	1677
2	60	0.00%	829	1525	1009
3	60	0.00%	829	2684	1288
4	60	0.00%	803	1769	1031
5	60	0.00%	760	3404	1396
6	60	0.00%	750	8532	2851
7	60	0.00%	798	1492	1019
8	60	0.00%	840	4112	1876
9	60	0.00%	811	2090	1175
10	60	0.00%	802	2026	1124
Rata - rata		**0.00%**	**803ms**	**3214ms**	**1444ms**

Tabel Percobaan dengan 80 *user*/30s tanpa *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	80	0.00%	1037	16635	7697
2	80	0.00%	1037	16635	7697
3	80	0.00%	859	32352	14002
4	80	0.00%	1636	20406	9465
5	80	0.00%	1448	28861	8322
6	80	0.00%	932	15933	5652
7	80	0.00%	886	18526	6981
8	80	0.00%	1742	20553	9942
9	80	0.00%	753	26461	6674
10	80	0.00%	783	25339	7429
Rata - rata		**0.00%**	**1111ms**	**22170ms**	**8386ms**

Tabel Percobaan dengan 90 *user*/30s tanpa *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	90	11.11%	403	25564	10557
2	90	5.56%	360	28561	12627
3	90	10.00%	241	25763	9866
4	90	8.89%	282	28350	11049
5	90	10.00%	360	28752	11064
6	90	8.89%	314	29734	11104
7	90	6.67%	183	29033	11696
8	90	8.89%	220	30036	10287
9	90	8.89%	73	29446	12139
10	90	8.89%	343	32044	11262
Rata - rata		**8.79%**	**277ms**	**28728ms**	**11165ms**

Tabel Percobaan dengan 100 *user*/30s tanpa *Indexes* dengan *Round Robin*

Percobaan	Samples	*Error* %	Min	Max	Average
1	100	11.00%	106	37852	13905
2	100	8.00%	304	36039	14155
3	100	8.00%	268	36664	15894
4	100	7.00%	16	37021	15114
5	100	10.00%	160	34269	13986
6	100	9.00%	320	33101	13748
7	100	7.00%	302	38280	16951
8	100	9.00%	80	41307	18082
9	100	10.00%	169	35734	16545
10	100	6.00%	334	39234	17515
Rata - rata		**8.50%**	**205ms**	**36950ms**	**15589ms**