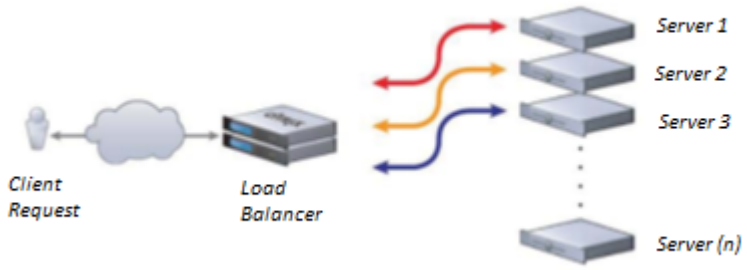
BAB II
TINJAUAN PUSTAKA

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.

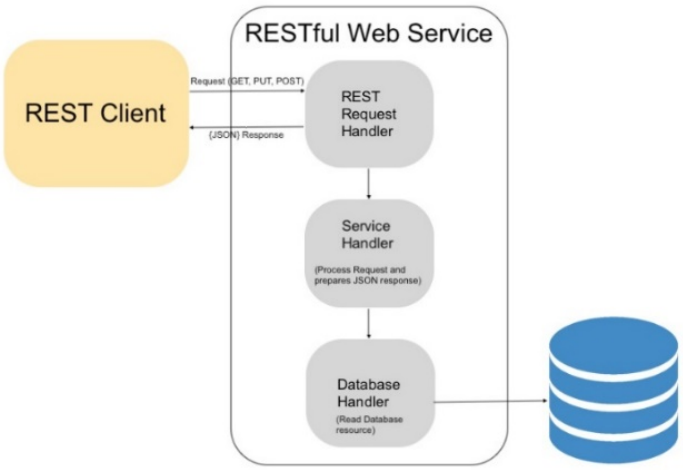
Hingga saat ini, Golang mulai banyak digunakan pada perusahaan-perusahaan besar maupun startup yang bergerak di bidang teknologi [13]. Hal ini dikarenakan pemrograman dengan Golang ini memiliki beberapa kelebihan sebagai berikut:

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.