ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

# IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS


AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Dosen Pembimbing I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

TUGAS AKHIR - KI141502

# IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS

AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Dosen Pembimbing I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

UNDERGRADUATE THESIS - KI141502

# IMPLEMENTATION OF PUBLISH/SUBSCRIBE ON DESIGN OF NETWORKING DEVICE MONITORING SYSTEM IN ITS

AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Supervisor I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

# LEMBAR PENGESAHAN

## IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS

**TUGAS AKHIR**

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
**AFIF RIDHO KAMAL PUTRA**
**NRP: 05111440000173**

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
NIP: 197708242006041001 ..... (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D
NIP: 198611252018031001 ..... (Pembimbing 2)


**SURABAYA**
**Juni 2018**

*(Halaman ini sengaja dikosongkan)*

**IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS**


| | | |
|---|---|---|
| **Nama** | : | **AFIF RIDHO KAMAL PUTRA** |
| **NRP** | : | **05111440000173** |
| **Jurusan** | : | **Informatika FTIK** |
| **Pembimbing I** | : | **Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D** |
| **Pembimbing II** | : | **Bagus Jati Santoso, S.Kom., Ph.D** |


## Abstrak


*Semakin berkembangnya teknologi informasi menuntut semakin banyaknya penggunaan perangkat berupa komputer atau perangkat jaringan. Setiap perangkat jaringan memiliki fungsi masing-masing. Sebagai contoh, salah satu fungsi yang dimiliki oleh router adalah untuk medistribusikan alamat kepada tiap host agar tiap host dapat berkomunikasi satu sama lain, lalu ada pula switch yang memiliki fungsi utama yaitu menerima informasi dari berbagai sumber yang tersambung dengannya, kemudian menyalurkan informasi tersebut kepada pihak yang membutuhkannya saja.*

*Pada suatu organisasi yang besar, kubutuhan akan perangkat jaringan sangatlah besar, terutama dalam hal jumlah perangkat yang digunakan. Banyaknya jumlah perangkat jaringan yang digunakan otomatis membuat jumlah perangkat jaringan yang dipantau juga banyak jumlahnya. Dikarenakan banyaknya jumlah perangkat jaringan yang harus dipantau, seringkali para teknisi mengalami kesulitan dalam memantau kinerja dari tiap perangkat jaringan. Oleh karena itu, dibutuhkan sebuah sistem yang dapat memantau kinerja dari setiap perangkat jaringan yang terpasang.*

*Aplikasi yang dirancang pada tugas akhir ini, menghadirkan sebuah sistem berbasis web yang dapat memantau seluruh perangkat jaringan yang terhubung dalam jaringan dengan metode publish/subscribe, sehingga setiap user nantinya dapat memilih perangkat jaringan apa saja yang ingin dipantau, dan dapat memilih informasi apa saja yang ingin didapat kan dari tiap-tiap perangkat jaringan yang telah dipilih.*

***Kata-Kunci**: aplikasi web, publish/subscribe*

**IMPLEMENTATION OF PUBLISH/SUBSCRIBE ON DESIGN OF NETWORKING DEVICE MONITORING SYSTEM IN ITS**


| | | |
|---|---|---|
| **Name** | : | **AFIF RIDHO KAMAL PUTRA** |
| **NRP** | : | **05111440000173** |
| **Major** | : | **Informatics FTIK** |
| **Supervisor I** | : | **Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D** |
| **Supervisor II** | : | **Bagus Jati Santoso, S.Kom., Ph.D** |


## Abstract


*The development of information technology requires the increasing of devices used in the form of computers or network devices. Every network device has its own function. For example, one of the functions owned by a router is to distribute the address to each host so that the host can communicate with each other, then there is also a switch that has the main function to receiving information from various sources connected to it, then channeling the information to the party who need it only.*

*In a large organization, the need for network devices is enormous, especially in the number of devices used. The large number of network devices used, automatically keeps the number of network devices being monitored also in increased. Due to the large number of network devices to monitor, it brings technicians into a trouble to monitor the performance of each network device frequently. Therefore, a system that can monitor the performance of each installed network device is required.*

*The application designed in this final project, presents a web-based system that can monitor all network devices connected in the network with the method of publish/subscribe, so that each user will be able to choose any network device that*

*they want to be monitored, and can choose what information that they want to get from each network device that has been selected.*

***Keywords**: web application, publish/subscribe*

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Implementasi *Publish/Subscribe* pada Rancang Bangun Sistem Monitoring Perangkat Jaringan di ITS**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Bapak Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
3. Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.

5. Serta semua pihak yang telah turut membantu penulis dalam bentuk apapun selama proses penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

Afif Ridho Kamal Putra

# DAFTAR ISI

*(Halaman ini sengaja dikosongkan)*

# DAFTAR TABEL

*(Halaman ini sengaja dikosongkan)*

# DAFTAR GAMBAR

*(Halaman ini sengaja dikosongkan)*

# DAFTAR KODE SUMBER

*(Halaman ini sengaja dikosongkan)*

# BAB I

# PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

## 1.1 Latar Belakang

Semakin berkembangnya teknologi informasi menuntut semakin banyaknya penggunaan perangkat berupa komputer atau perangkat jaringan. Setiap perangkat jaringan memiliki fungsi masing-masing. Sebagai contoh, salah satu fungsi yang dimiliki oleh *router* adalah untuk medistribusikan alamat kepada tiap *host* agar tiap *host* dapat berkomunikasi satu sama lain, lalu ada pula *switch* yang memiliki fungsi utama yaitu menerima informasi dari berbagai sumber yang tersambung dengannya, kemudian menyalurkan informasi tersebut kepada pihak yang membutuhkannya saja.

Pada suatu organisasi yang besar, kubutuhan akan perangkat jaringan sangatlah besar, terutama dalam hal jumlah perangkat yang digunakan. Banyaknya jumlah perangkat jaringan yang digunakan otomatis membuat jumlah perangkat jaringan yang dipantau juga banyak jumlahnya. Dikarenakan banyaknya jumlah perangkat jaringan yang harus dipantau, seringkali para teknisi mengalami kesulitan dalam memantau kinerja dari tiap perangkat jaringan. Oleh karena itu, dibutuhkan sebuah sistem yang dapat memantau kinerja dari setiap perangkat jaringan yang terpasang.

Aplikasi yang dirancang pada tugas akhir ini, menghadirkan sebuah sistem yang dapat memantau seluruh perangkat jaringan yang terhubung dalam jaringan dengan metode *publish/subscribe*, sehingga setiap pengguna nantinya dapat memilih perangkat jaringan apa saja yang ingin dipantau, dan

dapat memilih informasi apa saja yang ingin didapat kan dari tiap-tiap perangkat jaringan yang telah dipilih.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara membuat agen *polling* untuk mengambil data pada suatu perangkat jaringan?
2. Bagimana cara mengimplementasikan *publish/subscribe* sebagai *middleware*?
3. Bagaimana cara menyaring informasi perangkat yang dipilih oleh pengguna (*user*)?

## 1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Pengguna (*user*) hanya dapat memantau perangkat jaringan, diantaranya adalah: *router* dan *switch*.
2. Parameter untuk memantau perangkat jaringan adalah ketersediaan dan beban yang ditampung.
3. Performa yang diukur adalah waktu respon sistem.

## 1.4 Tujuan

Tugas akhir dibuat dengan beberapa tujuan. Berikut beberapa tujuan dari pembuatan tugas akhir:

1. Membuat sistem *monitoring* perangkat jaringan berbasis *web service*.
2. Mengimplementasikan metode *publish/subscribe* pada sistem *monitoring* perangkat jaringan.

## 1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini antara lain adalah:

1. Memonitor ketersediaan dan beban pada sebuah perangkat jaringan.
2. Memudahkan pengguna (*user*) untuk memantau perangkat jaringan yang diinginkan.

*(Halaman ini sengaja dikosongkan)*

# BAB II

# TINJAUAN PUSTAKA

## 2.1 *Publish/subscribe*

*Publish/subscribe* muncul sebagai paradigma komunikasi yang populer untuk sistem terdistribusi dalam skala yang besar. dalam *publish/subscribe consumer* akan berlangganan ke suatu *event* yang diinginkan. terlepas dari kegiatan *consumer*, ada *event producer* yang akan menerbitkan suatu *event*. jika event yang diterbitkan oleh produser sesuai dengan *event* yang dilanggani oleh *consumer*, *event* tersebut akan dikirim kepada *consumer* secara *asynchronus*. Interaksi ini difasilitasi oleh *middleware publish/subscribe*. *middleware publish/subscribe* dapat dipusatkan menjadi sebuah *node* tunggal yang berperan sebagai *broker* dari sebuah *event* atau dipisahkan menjadi kumpulan beberapa *node broker* dari sebuah *event*.

pada dasarnya, *publish/subscribe* dibagi dari dua jenis yaitu: *topic-based* dan *content-based*. Pada *topic-based publish-subscribe*, *event* diterbitkan melalui sebuah topik dan *consumer event* akan berlangganan topik tersebut untuk mendapatkan data dari suatu *event*. berlangganan pada kasus *topic-based* tidak didukung pemilahan data dari suatu *event*. contohnya, *consumer* akan menerima semua data dari suatu *event* yang diterbikan pada suatu topik. pada *content-based publish-subscribe*, berlangganan pada kasus ini didukung oleh fitur pemilahan yang diterapkan pada suatu *event* yang diterbitkan. data yang dipilah oleh *consumer* pada suatu *event* yang berlangganan akan dikirimkan ke *consumer*.[1]

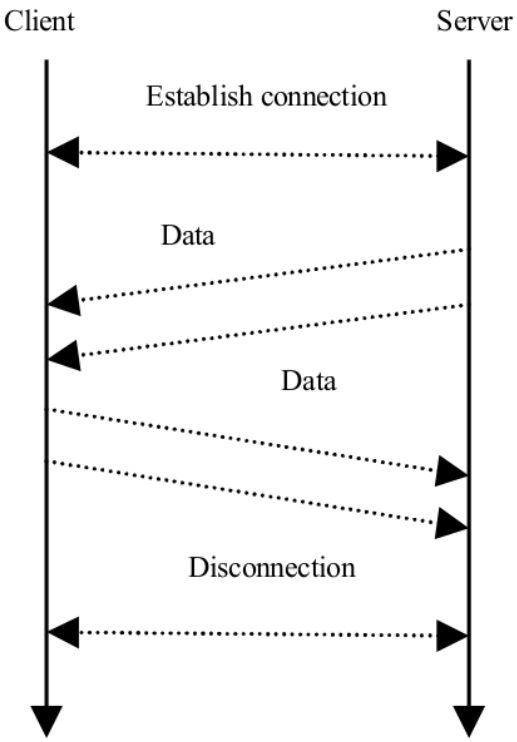
## 2.2 *Websocket*

Websocket adalah protokol berbasis TCP yang menyediakan *channel* komunikasi *full-duplex* antara *server* dan *client* melalui koneksi TCP tunggal. dibandingkan dengan skema komunikasi *web real-time* tradisional, protokol Websocket menghemat

banyak sumber daya *bandwidth* pada jaringan, sumber daya server, dan performa *real-time* yang sangat jauh lebih baik dibanding websocket tradisional. Websocket adalah protocol berbasis TCP yang independen. Websocket hanya berhubungan dengan HTTP yang memiliki *handshake* yang diterjemahkan oleh HTTP *server* sebagai pengembangan dari sebuah *request*. Websocket terdiri dari dua bagian yaitu: *handshake* dan *data transfer*.

Untuk membuat koneksi Websocket, *client* harus mengirimkan *request* HTTP kepada *server*. Setelah itu protokol akan diupgrade menjadi protokol Websocket, lalu server akan mengenali tipe request berdasarkan header pada HTTP. Protokol akan diupgrade menjadi Websocket apalbila diminta oleh Websocket, dan kedua kubu (*client* dan *server*) akan memulai komunikasi *full-duplex*, yang berarti *client* dan *server* dapat bertukar data kapanpun sampai salah satu dari *client* atau *server* menutup koneksi tersebut. Model komunikasi Websocket dapat dilihat pada gambar 2.1.

Websocket memiliki kemampuan yang lebih baik dalam berkomunikasi dibandingkan dengan skema komunikasi tradisional, dimana komunikasi terjadi secara *realtime*. sekali koneksi sudah berhasil dibuat, *server* dan *client* melakukan aliran data dua arah, dimana aktivitas tersebut meningkatkan mempuan *server* untuk mengirim data. Bandingkan dengan protokol HTTP, dimana informasi yang dikirimkan lebih ringkas dan mengurangi transmisi dari data yang redundan. Dengan skala user yang besar dan kebutuhan komunikasi *realtime* yang tinggi, menurunkan beban pada jaringan akan menjadi keuntungan dibanding komunikasi *realtime* secara tradisional.[2]




**Gambar 2.1:** Model Komunikasi *Websocket*

## 2.3 SNMP

*Simple Network Management Protocol* (SNMP) adalah aplikasi pada lapisan (*layer*) protokol yang digunakan untuk mengatur data pada jaringan. hampir semua *vendor* jaringan mendukung protokol SNMP. beberapa vendor peralatan telekomunikasi juga mulai didukung oleh protokol SNMP untuk mencapai pengaturan (manajemen) yang terintegrasi. Banyak dari aktivitas manajemen jaringan pada jaringan *enterprise* yang menggunakan SNMP dalam persentasi yang sangat besar.

SNMP yang berdasarkan paradigma *server - client* termasuk kedalam manajemen stasiun, agen dan *Management Information Bases* (MIB). tujuan dari manajemen stasiun adalah untuk mengirimkan *request* kepada *agent* dan mengendalikan mereka, manajemen stasiun juga menyediakan antarmuka antara manajer jaringan manusia dan sistem manajemen jaringan. setiap perangkat jaringan memungkinkan untuk mempunyai agen yang dapat mengendalikan basis data dan ketika stasiun manajemen mulai melakukan polling, agen-agen tersebut akan mengirimkan laporan kepada stasiun manajemen.[3]

Dalam pendekatan pemantauan dengan menggunakan SNMP, setiap agen akan mengirimkan stasiun manajemen sebuah informasi melalui *polling* laporan kejadian. *Polling* adalah aktivitas untuk melakukan interaksi antara agen dan stasiun manajemen menggunakan metode *request* dan *response*. Namun, stasiun manajemen hanya mendengarkan kepada informasi masuk pada pendekatan pelaporan kejadian. Agen akan mengirim informasi kepada stasiun manajemen setiap informasi tersebut dibutuhkan berdasarkan sebuah keputusan.

Pendekatan pemantauan secara *realtime* akan didefinisikan sebagai persetujuan antara agen dan stasiun manajemen, dimana pada persetujuan semacam ini, agen harus mengirimkan informasi kepada manajemennya secara berkala tanpa permintaan dari stasiun.[4]

Dalam sebuah kelompok, status dan sifat dari sistem akan dipertimbangkan sebagai pekerjaan memantau dari informasi MIB. tipe data yang akan digunakan untuk memantau lebih penting dibandingkan dengan perancangan jaringan. berikut ini adalah ingormasi yang harus digunakan dalah pemantauan:

- Static: Struktur dan elemen didalam konfigurasi dikategorikan seperti id dari *port* pada sebuah *router* atau *host*. Informasi ini akan jarang berubah.
- Dynamic: Informasi kejadian pada jaringan seperti paket

dan elemen jaringan
- Statistical: Informasi harus berasal dari informasi dinamis seperti rata-rata dari paket yang dikirimkan pada tiap unit.

### 2.3.1 OID

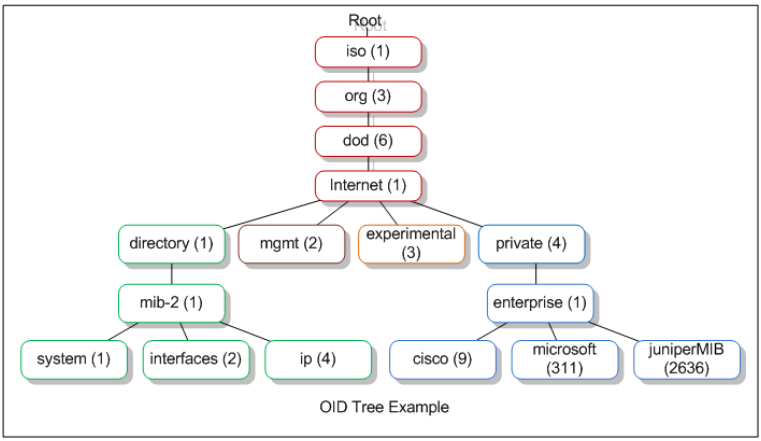
*Object Identifier* adalah sesuatu untuk mengidentifikasi sebuah objek. Objek ini dapat berupa daerah atau *disk drive* tunggal. Yang paling umum, didalam IEEE-RAC, adalah OUI (Organizationally Unique Identifier), dan diturunkan secara terorganisasir, dan terdaftar diluar OUI. pengidentifikasi yang paling umum selanjutnya, termasuk pengidentifikasi alamat ethernet adalah pengidentifikasi *Extended Unique Identifiers* (EUI) atau the *World Wide Name* (WWN). uniknya, untuk sistem yang sesuai, merupakan properti berharga dalam dua kasus ini. keunikan ini diasumsikan oleh struktur dari nomor unik yang dimulai dengan OUI.

IEEE-RAC menetapkan OUI sebagai *Object Identifier* untuk sebuah organisasi. *Object Identifier* ini merupakan lapisan didalam konteks yang lebih luas dari pengidentifikasi yang diturunkan secara unik dari titik awal dari sebuah OID, *International Telecommunication Union Telecommunication Standardization Sector* (ITU-T) dan dideskripsikan didalam standar ASN.1. jalur tersebut dilacak menuju ITU-T disebut sebagai ”arc” dari sebuah OID. Arc ini berkembang menjadi OUI dan RAC lain menetapkan perancang dan melalui penempatan yang dibuat oleh organisasi hingga titik akhir dari sebuah *Object Identifier*.[5].

Pada implementasinya, OID digambarkan sebagai sebuah hirarki yang tiap entitasinya diidentifikasi dengan nomor di tiap oktet pada OID. Contoh hirarki pada OID dapat dilihat pada gambar 2.2.

Pada tugas akhir ini OID diimplementasikan pada sebuah perangkat jaringan seperti *router* atau *switch*, yang nantinya

router atau switch tersebut menyimpan OID tersebut sebagai pengidentifikasi untuk diambil datanya lewat protokol SNMP.




**Gambar 2.2:** Contoh *Object Identifier* (OID)

## 2.4 Nagios

Nagios adalah perangkat lunak *opensource* yang aktif dikembangkan, memiliki banyak user juga komunitas yang luas, memiliki banyak *plugin* tambahan yang dikembangkan oleh user maupun yang terdapat langsung pada awal pengaturan, dan banyak buku tentang Nagios yang diterbitkan. Nagios juga merupakan sistem pemantauan yang paling populer yang cocok dengan hampir semua distribusi linux. Dukungan komersial tersedia dari perusahaan yang didirikan oleh penciptanya dan pengembang utama sebagai penyedia solusi resmi. Peralatan pemantauan yang berbasis Nagios juga tersedia, seperti sensor yang dirancang untuk beroprasi bersama nagios. Karena fleksibilitas dari rancangan perangkat lunak yang menggunakan arsitektur *plug-in*, layanan pengecekan untuk aplikasi yang pustakanya sudah ditentukan dapat di gunakan. Didalam Nagios

terdapat beberapa *plugin* lain, seperti *script* tambahan yang dapat dikostumisasi dan dapat digunakan pada Nagios. Nagios adalah program yang ringan dan menyediakan alat pemantauan yang sempurna yang dapat membantu untuk memantau seluruh protokol yang aktif dan perakngkat jaringan yang terhubung dengan topologi. Nagios juga mampu untuk menyediakan grafik yang komperhensif dan bersifat *realtime* dan analisis tren.[6]

## 2.5 REST API

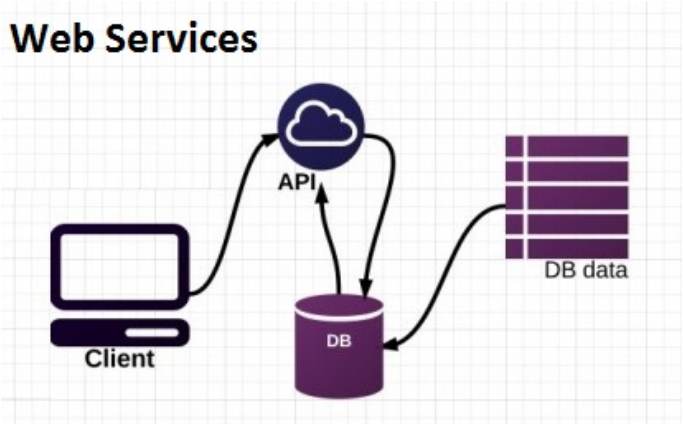
Istilah *representational state transfer* (REST) diperkenalkan oleh Roy Fielding. Gaya arsitektur REST adalah arsitektur *client-server* di mana *client* mengirim *request* ke *server*, kemudian server memproses request dan mengembalikan *response*. *Request* dan *response* ini membangun sekitar transfer dari representasi sumber daya. Sumber daya adalah sesuatu yang diidentifikasi oleh URI.

REST lebih sederhana daripada SOAP. Bahasa REST didasarkan pada penggunaan kata benda dan kata kerja. REST tidak perlu format pesan seperti *envelope* dan *header* yang diperlukan di SOAP. Jadi parsing XML juga tidak membutuhkan *bandwidth* yang banyak. Prinsip desain REST adalah sebagai berikut: *addressability, statelessness* dan *uniform interface*.

*Addressability*-REST memodelkan dataset untuk beroperasi sebagai sumber daya di mana sumber daya ditandai dengan URI. Sebuah antarmuka yang seragam dan standar digunakan untuk mengakses sumber daya yang tersedia yaitu menggunakan metode HTTP yang tetap. Setiap transaksi bersifat independen dan tidak terkait dengan transaksi sebelumnya, karena semua data yang diperlukan untuk memproses permintaan terkandung dalam permintaan itu, data sesi *client* tidak disimpan di sisi *server*. Oleh karena itu tanggapan *server* juga independen.

Prinsip membuat aplikasi REST sederhana dan ringan.

Aplikasi web yang mengikuti arsitektur REST dapat disebut sebagai layanan web RESTful. Penggunaan layanan web menggunakan metode http GET, PUT, POST dan DELETE untuk mengambil, membuat, memperbaharui, dan menghapus sumber daya.[7]. Arsitektur REST dapat dilihat pada gambar 2.3.

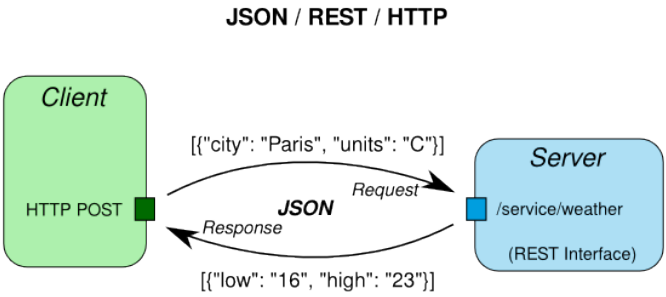



**Gambar 2.3:** Arsitektur REST

Pada gambar diatas, dapat dilihat bahwa arsitektur REST API memiliki prinsip kerja yang sangat sederhana, hanya membutuhkan server API dan database yang digunakan untuk mengambil atau menyimpan data, tanpa perlu sebuah tampilan antarmuka yang dapat dilihat oleh pengguna.

untuk menggunakan REST API pengguna harus mengirimkan *request* dengan sebuah metode yang sesuai dengan apa yang sudah ditentukan menuju ke sebuah endpoint yang tersedia pada REST API. Selain mendeklarasikan endpoint dan metode yang diperlukan untuk mengirim, dibutuhkan juga sebuah pesan pada badan (*body*) *request* dalam bentuk json. Pada gambar 2.4 dapat dilihat bahwa pengguna (*client*) mengirimkan sebuah *request* dalam bentuk json, setelah itu

server akan mengirimkan respon berupa data json yang sesuai dengan apa yang diinginkan oleh pengguna (*client*).




**Gambar 2.4:** Contoh Penggunaan REST

*(Halaman ini sengaja dikosongkan)*

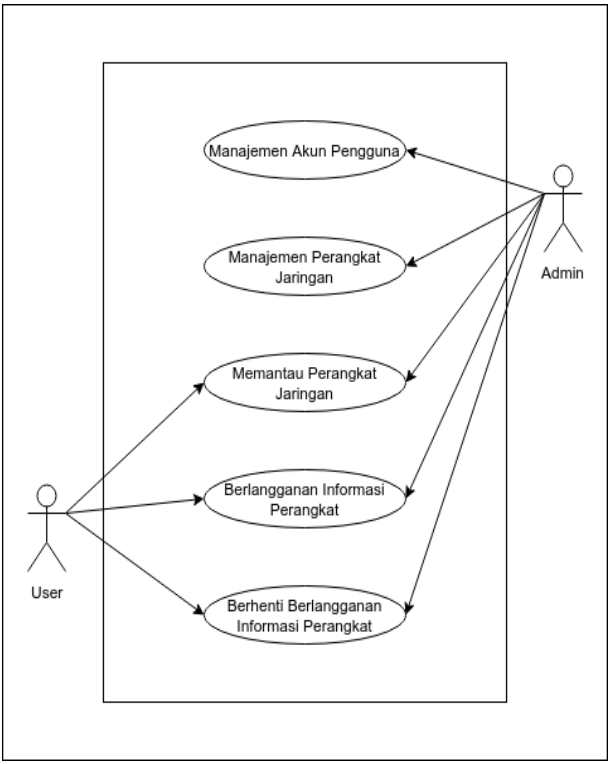
# BAB III

# DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

## 3.1 Kasus Penggunaan

Terdapat dua aktor dalam sistem ini, yaitu Pengembang (*Administrator*) dan *User* (Pengguna) dari aplikasi web yang dikelola oleh sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.




**Gambar 3.1:** Diagram Kasus Penggunaan

Diagram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

| **Kode Kasus Penggunaan** | **Nama Kasus Penggunaan** | **Keterangan** |
|---|---|---|
| UC-0001 | Manajemen Akun Pengguna. | Pengelola (*Admin*) dapat membuat, melihat, mengubah dan menghapus data akun pengguna. |
| UC-0002 | Manajemen Perangkat Jaringan. | Pengelola (*Admin*) dapat membuat, melihat, mengubah dan menghapus data perangkat jaringan. |
| UC-0003 | Memantau Perangkat Jaringan. | Pengelola (*Admin*) dan Pengguna (*User*) dapat memantau seluruh perangkat jaringan yang sudah ia langgani. |
| UC-0004 | Berlangganan Informasi Perangkat. | Pengelola (*Admin*) dan Pengguna (*User*) dapat berlangganan informasi perangkat jaringan yang diinginkan. |

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

| Kode Kasus Penggunaan | Nama Kasus Penggunaan | Keterangan |
|---|---|---|
| UC-0005 | Berhenti Berlangganan Informasi Perangkat. | Pengelola (*Admin*) dan Pengguna (*User*) dapat berhenti berlangganan informasi perangkat jaringan yang diinginkan. |

## 3.2 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun.

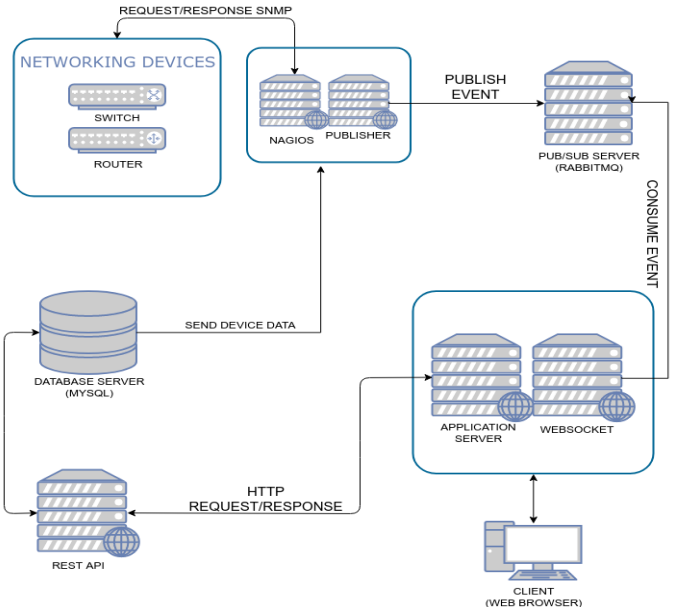
### 3.2.1 Desain Umum Sistem

Sistem yang akan dibuat yaitu sistem yang dapat melakukan pemantauan pada perangkat jaringan yang berbasis *web* dengan metode *pusblish/subscribe*, dimana pengguna (*user*) harus berlangganan kepada suatu informasi untuk mendapatkan informasi yang diinginkan.

Sistem ini melibatkan 3 (Tiga) *server* yang berfungsi sebagai *Webserver* dan 1 (satu) *server* yang berfungsi sebagai *database server*. *Server* aplikasi dan *Websocket server* berada pada satu *server*, sehingga pada implementasinya *Webserver* aplikasi dan *Websocket* dijalankan pada *port* yang berbeda. Pada sistem ini *client* yaitu pengguna (*user*) dan pengelola (*admin*) akan mengakses aplikasi menggunakan *web browser*. yang nantinya jika mengakses fitur selain memantau perangkat jaringan, aplikasi akan mengirimkan *request* HTTP kepada REST API, dimana REST API tersebut melakukan transaksi data kepada

database *server*. setelah itu REST API akan mengirimkan *response* kepada aplikasi.

jika *client* mengakses fitur pemantauan perangkat jaringan, maka aplikasi akan terhubung dengan *websocket* yang tugasnya mengakses data yang berada pada pub/sub *server*, dimana pub/sub *server* menyimpan data yang diterbitkan oleh Nagios, data tersebut adalah hasil respon dari SNMP Nagios kepada tiap perangkat jaringan terkait. Penjelasan secara umum arsitektur sistem akan diuraikan pada Gambar 3.2.




**Gambar 3.2:** Desain Umum Sistem

### 3.2.2 Desain REST API

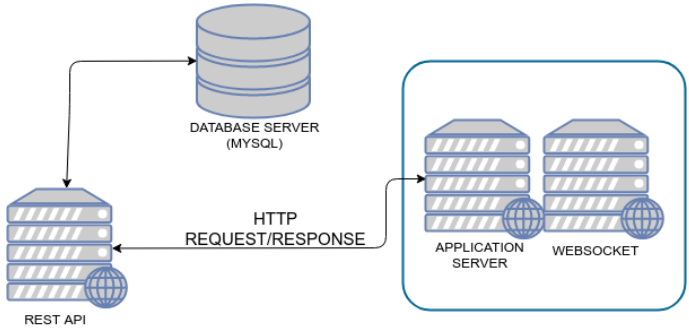
REST API bertujuan untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk di kembangkan

(*scalable*) terutama dalam pertukaran dan komunikasi data. REST API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti *developer*. URL API biasa disebut *endpoint* dalam pemanggilannya.

Pada sistem ini terdapat beberapa *endpoint*, beberapa *endpoint* dibagi menjadi beberapa *endpoint* sesuai dengan perintah yang diajalankannya. misal: *create*, *read*, *delete*, *update* dan lain-lain.

Server aplikasi mengirimkan HTTP *request* kepada REST API yang nantinya REST API akan melakukan trasaksi data pada database sesuai dengan *endpoint*nya masing-masing. setelah itu REST API akan mengirimkan HTTP *response* kepada server aplikasi.

Secara umum, arsitektur dari REST API dapat dilihat pada Gambar 3.3




**Gambar 3.3:** Desain REST API

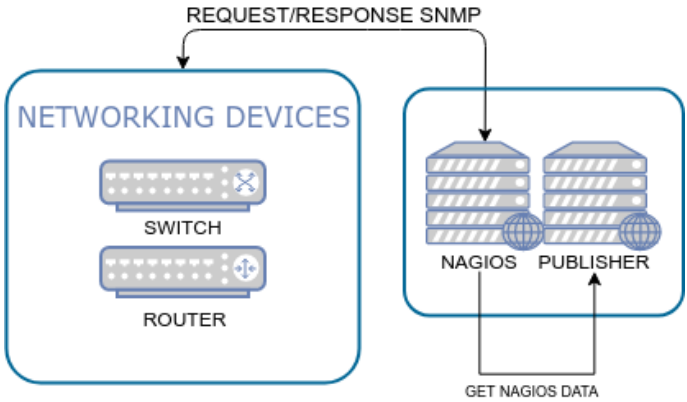
### 3.2.3 Desain Publisher Server

Pada *publisher server*, dipasang aplikasi untuk memantau kinerja jaringan, yaitu Nagios. Pada Nagios, perdapat plugin

untuk memantau kinerja jaringan dengan protokol SNMP yaitu `check-snmp`. plugin ini membutuhkan beberapa parameter, diantaranya: alamat perangkat yang ingin dipantau dan OID dari apa yang ingin dipantau.

Sebuah *script* dibuat untuk mengambil data dan mengirimkannya menuju pub/sub *server*. setiap perangkat yang dipantau dimasukkan ke sebuah *thread* baru agar dapat berjalan secara paralel. didalam *thread* tersebut, setiap perangkat terkait diperiksa kinerjanya dengan protokol SNMP dan hasilnya dikirimkan kepada pub/sub server melalui sebuah *exchange* yang telah diikat dengan sebuah *message queue* yang sebelumnya telah diinisiasi. Rancangan umum dari *Publisher Server* seperti yang digambarkan pada Gambar 3.4.

*Exchange* yang dibuat oleh *script* tersebut namanya dibuat berdasarkan uuid versi 4 dari tiap perangkat yang diambil dari *database* dan nama *queue* dibuat berdasarkan UUID versi 4 yang dibuat baru.




**Gambar 3.4:** Desain Publisher Server

### 3.2.4 Desain Pub/Sub Server

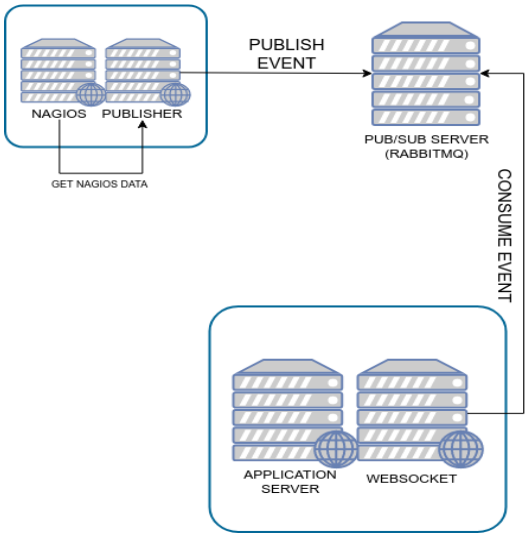
*Publish/subscribe server* atau bisa juga disebut pub/sub *server*. yaitu sebuah *server* untuk menampung seluruh pesan yang dikirimkan oleh *publisher*. didalamnya terpasang aplikasi *message broker* yaitu `RabbitMQ`. seluruh pesan yang dikirimkan oleh *publisher* dikirimkan ke pub/sub *server* melalui sebuah *exchange* yang diikat dengan sebuah *queue* setelah itu *server* akan menyimpan pesan *queue* tersebut hingga ada *subscriber* yang meminta data tersebut untuk dikirimkan. dalam kasus ini yang bertindak sebagai *subscriber* adalah *websocket server*.

Di sisi *websocket* dan *server* aplikasi, *websocket* menginisiasi sebuah *exchange* yang namanya dibuat berdasarkan uuid versi 4 dari tiap perangkat yang ingin dipantau dari *database server*, dengan syarat *exchange* dengan nama tersebut belum dibuat atau terdaftar sebelumnya. jika exchange dengan nama tersebut sudah dibuat atau terdaftar sebelumnya pada pub/sub *server* maka *websocket server* tidak perlu membuat *exchange* tersebut.

Begitu juga dengan *queue*-nya. *queue* dibuat dengan nama UUID yang telah dibuat acak oleh *client* dengan algoritma UUID versi 4, dengan syarat queue dengan nama tersebut belum dibuat atau terdaftar sebelumnya. Jika *queue* dengan nama tersebut sudah dibuat atau terdaftar sebelumnya pada pub/sub *server* maka *websocket server* tidak perlu membuat *queue* tersebut.

Setelah menghadapi masalah pembuatan *exchange* dan *queue*, *websocket* baru mengambil data perangkat pada pub/sub *server* sesuai dengan data apa saja yang dilanggani oleh *client*.

Secara umum, arsitektur rancangan dari *Pub/Sub Server* dapat dilihat pada Gambar 3.5.



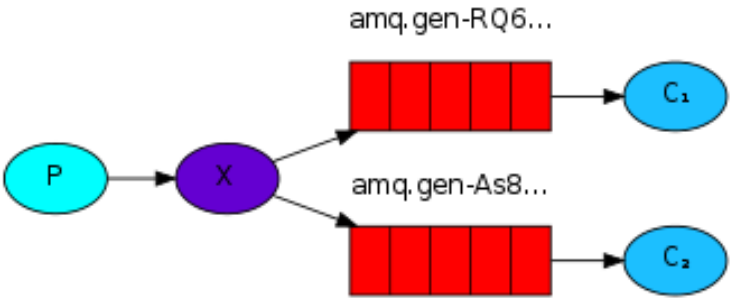

**Gambar 3.5:** Desain Pub/Sub Server

### 3.2.5 Desain subscriber pada Application Server dan Websocket

*Subscriber* berfungsi untuk mengambil data yang dibutuhkan oleh *client* dari pub/sub *server*. Pada sistem ini, *subscriber* didesain untuk diimplementasikan pada *websocket* agar data yang diterima oleh *client* adalah data yang paling terbaru (*realtime*). *websocket* ini nantinya akan disambungkan dengan sebuah *endpoint* (URL) pada aplikasi.

*subscriber* ini nantinya akan membuat sebuah *queue* dengan nama yang ditentukan oleh *client*. nama dari queue tersebut ditentukan dengan membuat *string* UUID versi 4 secara acak. Setelah berhasil membuat *queue*, *subscriber* membuat *exchange* yang banyaknya sejumlah perangkat yang terdaftar pada sistem dan *exchange* tersebut diberi nama sesuai dengan ID pada masing-masing perangkat yang dimana ID tersebut berformat

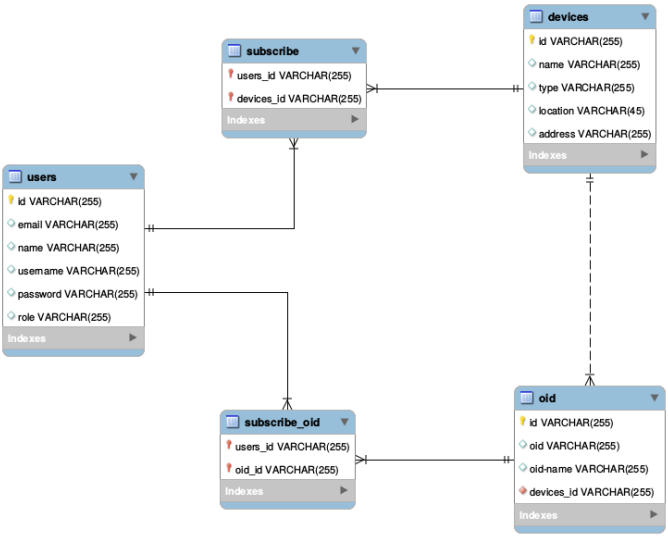
UUID versi 4. Pembuatan *queue* dan *exchange* akan dilakukan jika *queue* dan *exchange* belum terdaftar pada pub/sub *server*. jika *queue* dan *exchange* sudah terdaftar, maka tidak akan ada *queue* dan *exchange* yang akan dibuat. Setelah *queue* dan *exchange* berhasil dibuat. *queue* tersebut akan diikat dengan satu atau lebih *exchange*. lewat *queue* tersebutlah data akan dikirim. ilustrasi cara kerja *queue* dan *exchange* dapat dilihat pada gambar 3.6.




**Gambar 3.6:** Ilustrasi Cara Kerja *Queue* dan *Exchange*

### 3.2.6 Desain Database Server

Desain *database* pada sistem ini adalah seperti yang digambarkan pada gambar 3.7. Terdapat tiga tabel utama yang mewakili tiap entitas yang terlibat dalam sistem ini, yaitu: *users*, *devices*, dan oid. selain itu, terdapat dua *table many-to-many* untuk menyimpan data pengguna yang telah berlangganan kepada tiap perangkat dan pengguna yang berlangganan kepada tiap OID (untuk mengetahui informasi apa saja yang ada pada tiap perangkat. tiap OID memiliki informasi yang berbeda).




**Gambar 3.7:** Desain *Database*

### 3.2.7 Desain Antarmuka

Desain antarmuka adalah desain untuk halaman yang nantinya akan digunakan oleh *client* baik itu pengguna (*user*) ataupun pengelola (*admin*). Antarmuka yang nantinya dibuat berbasis web dan menggunakan Bootstrap 3 dan HTML. terdapat beberapa perbedaan pada antarmuka yang digunakan oleh pengelola dan pengguna. Misal, pada antarmuka yang digunakan pengguna tidak ada tombol untuk menghapus data perangkat, sedangkan pada antarmuka yang digunakan oleh pengelola terdapat tombol untuk menghapus data perangkat yang telah terdaftar.

Desain antarmuka untuk menampilkan daftar seluruh perangkat yang tersedia pada sistem dapat dilihat pada gambar 3.8. pada halaman ini pengguna dan pengelola dapat meliahat daftar perangkat yang tersedia dan beberapa informasinya,

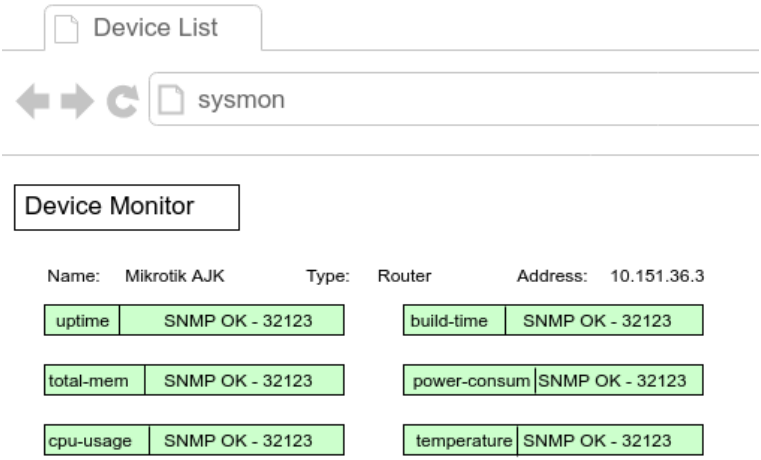
seperti: nama perangkat, alamat, dan lokasi dari perangkat tersebut. pada halaman ini pengguna dan pengelola juga bisa langsung berlangganan atau berhenti berlangganan dengan menekan sebuah tombol yang ada pada halaman ini.


**Gambar 3.8:** Desain Antarmuka Menampilkan Daftar Perangkat Yang Tersedia

Desain antarmuka untuk menampilkan rincian dari antarmuka terkait dapat dilihat pada gambar 3.9. pada halaman ini pengguna dan pengolola dapat melihat seluruh rincian data yang ada pada perangkat. mulai dari nama perangkat, tipe perangkat, alamat perangkat, lokasi perangkat dan informasi apa saja yang dapat dipantau memalui OID.

Pada halaman ini pengguna dan pengelola juga dapat belangganan dengan cara menekan sebuah tombol. tidak hanya berlangganan perangkatnya saja, pengguna dan pengelola juga dapat memilih untuk berlangganan informasi apa saja yang ingin didapatkan dari perangkat tersebut.


**Gambar 3.9:** Desain Antarmuka Menampilkan Rincian dari Perangkat Terkait




**Gambar 3.10:** Desain Antarmuka Pemantauan Perangkat

Desain antamuka pemantauan pernangkat dapat dilihat pada gambar 3.10. pada halaman ini, pengguna dan pengelola akan mendapatkan data dari seluruh perangkat yang sudah dilanggani. data yang ditampilkan pada halaman ini dipilih berdasarkan informasi yang dipilih pada antarmuka menampilkan rincian dari antarmuka terkait yang bisa dilihat pada gambar 3.9.

*(Halaman ini sengaja dikosongkan)*

# BAB IV

# IMPLEMENTASI

Bab ini membahas implementasi sistem pemantauan perangkat jaringan secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada, yaitu: REST API, *Publisher server*, Pub/sub *Server*, *Server* aplikasi dan *websocket*, *Database server* dan Antarmuka.

## 4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan virtualisasi Proxmox dengan spesifikasi *Host* komputer adalah Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz dengan memori 8 GB di Laboratorium Arsitektur dan Jaringan Komputer, Teknik Informatika ITS. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

- Sistem Operasi Linux Ubuntu Server 16.04 LTS
- RabbitMQ 3.7.5-1
- MySQL Ver 15.1 Distrib 10.0.34-MariaDB
- Python 2.7
- Flask 0.12.2
- Node.js v6.11.4
- Nagios 4.3.4
- Express.js 4.16.3

## 4.2 Implementasi REST API

REST API digunakan untuk memudahkan aplikasi agar ringan dan mudah untuk dikembangkan. pada tugas akhir ini, REST API memiliki fungsi utama untuk menyimpan data pelanggan (*user*) yang berlangganan pada perangkat jaringan atau berlangganan OID (Informasi didalam perangkat jaringan). REST API dibangun dengan kerangka kerja `Python` yaitu `Flask`

dan dilengkapi ORM (*Object-relational mapping*) Database yaitu `Peewee`.

### 4.2.1 Pemasangan Python Flask dan Peewee

Pemasangan `Python Flask` dapat dilakukan dengan mudah, cukup dengan memasangnya dengan manajer paket yang dimiliki oleh `Python` yaitu `Pip`. Setelah `Flask` berhasil terpasang, selanjutnya adalah tahap pemasangan ORM `Peewee`. `Peewee` adalah *Object-relational Mapping* dimana fungsi utamanya adalah memudahkan pengembang agar dapat menyambungkan aplikasi dengan database dan melakukan *query* dengan mudah. pemasangan ORM `Peewee` dapat dilakukan dengan cara mengambil berkas instalasinya pada `git https://github.com/coleifer/peewee.git` dan pasang `Peewee` sesuai dengan instruksi yang tertera pada situs `git` tersebut.

### 4.2.2 Implementasi *Endpoint* pada REST API

REST API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti *developer*. URL API biasa disebut *endpoint* dalam pemanggilannya.

Pada sistem ini terdapat beberapa *endpoint*, beberapa *endpoint* dibagi menjadi beberapa *endpoint* sesuai dengan perintah yang diajalankannya. misal: *create*, *read*, *delete*, *update* dan lain-lain.

Berikut ini adalah *endpoint* yang dibuat dalam sistem ini:

**Tabel 4.1:** Daftar *Endpoint* pada REST API

| No | Endpoint (Route) | Metode | Aksi |
|---|---|---|---|
| 1 | /register | POST | Membuat data baru pada tabel *user* di *database* |
| 2 | /login | POST | Mengambil data pada tabel *user* dan mencocokkannya dengan JSON yang dikirimkan lewat *body*. setelah data *username* dan *password* cocok, lalu dibuatkan sebuah token JWT. |
| 3 | /logout | POST | Memasukkan *token* JWT yang terdaftar pada server kedalam daftar hitam agar *token* tidak dapat digunakan lagi. |
| 4 | /users | GET | Menampilkan seluruh data *user* yang terdaftar pada sistem |

**Tabel 4.1:** Daftar *Endpoint* pada REST API

| No | Endpoint (Route) | Metode | Aksi |
|---|---|---|---|
| 5 | /users/<string:username> | GET | Menampilkan data user berdasarkan username yang tertulis pada URL |
| 6 | /devices/create | POST | Membuat data baru pada tabel *devices* di database |
| 7 | /devices/edit/<string:id> | PUT | Mengubah data pada tabel *devices* di *database* yang ID nya sama dengan ID yang ada pada URL. |
| 8 | /devices/delete | DELETE | Menghapus data pada tabel *devices* di *database* yang ID nya tertulis pada *body* yang bertipe JSON. |
| 9 | /devices | GET | Menampilkan seluruh data perangkat yang terdaftar pada sistem |

**Tabel 4.1:** Daftar *Endpoint* pada REST API

| No | Endpoint (Route) | Metode | Aksi |
|---|---|---|---|
| 10 | /devices/<string:id> | GET | Menampilkan data *user* berdasarkan *username* yang tertulis pada URL |
| 11 | /oid/create | POST | Membuat data baru pada tabel oid |
| 12 | /oid/edit | POST | Mengubah data pada tabel oid di *database* yang ID nya tertulis pada *body* yang bertipe JSON. |
| 13 | /oid/delete | POST | Menghapus data pada tabel oid di *database* yang ID nya tertulis pada *body* yang bertipe JSON. |
| 14 | /subscribe/devices | POST | Membuat data baru pada tabel *subscribe* |

**Tabel 4.1:** Daftar *Endpoint* pada REST API

| No | Endpoint (Route) | Metode | Aksi |
|---|---|---|---|
| 15 | /unsubscribe/devices | POST | Menghapus data pada tabel *subscribe* di *database* yang ID nya tertulis pada *body* yang bertipe JSON. |
| 16 | /subscribe/oid | POST | Membuat data baru pada tabel *subscribe_oid* |
| 17 | /unsubscribe/oid | POST | Menghapus data pada tabel *subscribe_oid* di *database* yang ID nya tertulis pada *body* yang bertipe JSON. |

## 4.3 Implementasi *Publisher Server*

*Publisher server* merupakan *server* yang berfungsi untuk mengambil data pada perangkat jaringan secara berkala dan mengirimkannya menuju pub/sub *server*. *publisher server* menggunakan *plugin* `check_snmp` bawaan program Nagios, Sehingga untuk melakukan pengambilan data, kita perlu memasang Nagios pada server.

setelah data berhasil dikumpulkan, data yang diambil pada tiap perangkat dikirimkan menuju pub/pub *server* melalui *thread* yang berbeda. proses ini dinamakan *multithreading*.

### 4.3.1 Pemasangan Nagios Sebagai Pemantau dan Pengumpul Data Perangkat

Pemasangan `Nagios` dapat dilakukan dengan beberapa cara, namun cara yang dipakai pada kasus ini adalah memasang `Nagios` langsung dari sumbernya untuk mendapatkan fitur terbaru, pembaharuan keamanan, dan pembetulan *bug*.

Berikut ini adalah sumber untuk mendapatkan Nagios yang siap untuk dipasang: `https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.3.4.tar.gz`

`Nagios` perlu beberapa perintah khusus yang hanyak bisa dilakukan oleh *user* yang bernama ”nagios” maka dari itu diperlukan *user* pada *server* yang bernama ”nagios” dengan nama *group* ”nagcmd”. Selain *user*, `Nagios` juga perlu beberapa paket yang harus terpasang sebelum memasang nagios itu sendiri. Beberapa paket diantaranya adalah: `build-essential, libgd2-xpm-dev, openssl, libssl-dev, unzip`

Setelah `Nagios` terpasang, direktori kerja dari Nagios dapat dilihat pada direktori `/usr/local/nagios`

### 4.3.2 Pengumpulan Data dan Pembuatan *Script* Pengiriman

Untuk mengumpulkan data perangkat jaringan, dibutuhkan *plugin* bawaan Nagios yang bernama `check_snmp`. *plugin* tersebut berada pada direktori `/usr/local/nagios/libexec`. Jika tidak terdapat plugin pada direktori tersebut maka kita harus mengunduh *plugin* SNMP tersebut lewat *website* Nagios (`https://exchange.nagios.org/directory/Plugins/Network-Protocols/SNMP/Check-SNMP-plugins/details`). untuk menjalankan *plugin* tersebut dibutuhkan dua parameter, yaitu: alamat dari perangkat jaringan yang ingin dipantau dan OID dari data yang ingin didapatkan dari perangkat jaringan terkait. perintah dijalankan lewat *console* pada

komputer server. perintah yang dijalankan untuk mendapatkan data pada perangkat jaringan lewat protokol SNMP adalah seperti yang tertulis pada kode sumber 4.1

```
$ /usr/local/nagios/check_snmp –H <
    alamat_perangkat> –o <oid_perangkat>
```

**Kode Sumber 4.1:** Perintah Mengumpulkan Data Perangkat dengan SNMP

Setelah data dapat dikumpulkan, sebuah *script* diperlukan untuk mengirim data tersebut menuju pub/sub *server* yang didukung boleh RabbitMQ sebagai *Message Broker*.

Sebuah pustaka bernama `pika` dibutuhkan untuk mengirim data tersebut ke pub/sub *server*. tiap perangkat yang dikumpulkan datanya dan dikirimkan ke pub/sub *server*, diproses didalam sebuah *thread* yang berbeda. oleh karena itu inisiasi *database* dibutuhkan pada awal *script* untuk menggetahui ada berapa perangkat yang terdaftar pada sistem.

pertama-tama, masukkan pustaka yang dibutuhkan untuk pembuatan *script* (termasuk `pika`), lalu dilanjutkan dengan potongan kode untuk menginisiasi *database*. *Pseudocode* untuk inisasi kelas *database* dapat dilihat pada kode sumber 4.2

```
1 initClassUsers()
2 initClassDevices()
3 initClassOid()
```

**Kode Sumber 4.2:** Pseudocode inisiasi Kelas Database

Setelah itu buat fungsi sebagai target menjalankan *thread*, nantinya tiap thread akan mengeksekusi kode yang ada didalam fungsi tersebut. didalam fungsi tersebut meliputi pegumpulan data dengan `check_snmp`. Data perangkat jaringan yang dikumpulkan dengan `check_snmp` dimasukkan kedalam sebuah `python` *dictionary* yang nantinya *dictionary* tersebut akan dikirimkan menuju pub/sub *server*. *Pseudocode* fungsi tersebut dapat dilihat pada kode sumber 4.3

```
rabbitMq(exchange, address):
  try:
    add getOidData() into array of dictionary
  except:
    add NULL into array of dictionary

  try:
    add getSnmpDeviceData() into JSON
  except:
    add Error Message into JSON
```

**Kode Sumber 4.3:** Pseudocode Target *Thread* Untuk Mengambil Data Perangkat

Untuk mengirimkan data menuju pub/sub *server* diperlukan *pusaka* `pika` yang akan membuat koneksi dengan RabbitMQ yang berada di pub/sub *server*. Pseudocode untuk mengirimkan data tersebut dapat dilihat pada kode sumber 4.4

```
pika.openConnection()
if exchange does not exist:
  createExchage()
  if queue does not exist:
    createQueue()
    bindExchangetoQueue()
  else:
    bindExchangetoQueue()
else:
  pass
sendMessage()
```

**Kode Sumber 4.4:** Pseudocode Pengiriman Data Dengan Pika

Setelah seluruh fungsi selesai dibuat, langkah terakhir adalah membuat *thread* agar tiap thread nantinya akan mejalankan

fungsi yang telah dibuat dan menjalankannya secara berkala. *Pseudocode* untuk membuat *thread* dapat dilihat pada kode sumber 4.5

```
1 Thread
2 while true:
3   getDeviceId() as exchangename
4   getDeviceAddress as deviceaddress
5   thread(target=rabbitmq(), argument=(
        exchangename, deviceaddress))
6   sleep(2)
```

**Kode Sumber 4.5:** Pseudocode Menjalankan Thread

## 4.4 Implementasi Pub/Sub *Server*

Pada pub/sub *server*, dipasang aplikasi message broker RabbitMQ. pada kasus ini RabbitMQ menerima seluruh data yang dikirmkan oleh *publisher*. Setelah itu, RabbitMQ menyimpannya dan menunggu hingga ada *subscriber* yang meminta data pada RabbitMQ. kriteria data yang dikirimkan harus dispesifikkan sesuai dengan apa yang diminta oleh *subscriber*.

Pemasangan aplikasi RabbitMQ membutuhkan bahasa pemrograman `erlang`. untuk itu sebelum memasang RabbitMQ, harus terlebih dahulu memasang `erlang` pada sistem. Selain `erlang`, beberapa paket juga harus terpasang pada sistem, beberapa diantaranya adalah: `init-system-helpers`, `socat`, `adduser`, `logrotate`

Setelah RabbitMQ server terpasang, selanjutnya dibutuhkan sebuah web admin untuk RabbitMQ agar mudah untuk melakukan manajemen data, user dan lain-lain pada *web admin* tersebut. RabbitMQ sudah menyediakan *plugin* agar *web admin*

dapat langsung digunakan. hanya dengan menjalankan perintah untuk mengaktifkan *web admin* dengan `rabbitmqctl`

Pada pub/sub server dibuat sebuah akun yang memiliki otoritas untuk mengirim data lewat protokol internet atau LAN. Karena pada keadaan *default* RabbitMQ akan menggunakan akun guest yang hanya bisa diakses lewat *localhost*.

## 4.5 Implementasi *Subscriber* pada Server Aplikasi dan *Websocket*

Pada kasus ini, terdapat *subscriber* yang diimplementasikan pada *websocket*. *Websocket* ini bertugas untuk meminta data pada pub/sub *server* yang telah dipasang RabbitMQ. *Websocket* ini disambungkan dengan suatu *endpoint* pada server aplikasi, sehingga ketika *client* mengakses *endpoint* pada aplikasi tersebut, `javascript` pada halaman tersebut akan menyambungkan halaman pada *server websocket* yang berada pada alamat dan *port* tertentu.

Server *websocket* dibangun dengan menggunakan `node.js` dengan beberapa tambahan pustaka. pustaka yang digunakan pada websocket ini diataranya adalah: `Express.js` yang berperan sebagai kerangka kerja untuk membuat *web* dengan `node.js`, `http` untuk membuat *webserver* sederhana dengan `node.js`, `socket.io` unutk membuat komunikasi antara *server* dengan *client* menggunakan protokol *websocket*, dan `amqplib` untuk berkomunikasi dengan pub/sub *server* yang telah depasang RabbitMQ.

implementasi inisiasi koneksi *websocket* dengan *client* dan pub/sub *server* dapat dilihat pada pseudocode yang terdapat pada kode sumber 4.6

```
connectToRabbitMQServer()
createWebSocketConnection()
if websocketConnected():
  sendToClient('Connected')
else if websocketDisconnected():
  sendToClient('Disconnected')
```

**Kode Sumber 4.6:** Pseudocode Inisiasi Komunikasi Websokcet dengan Client dan Pub/Sub Server

Pada `javascript` yang terdapat pada *client* terdapat fungsi untuk menangkap pesan dari *server* bahwa *websocket* terlah berhasil tersambung. bersamaan saat *websocket* berhasil tersambung, *client* mengirimkan seluruh id pada tabel *devices* yang telah dilangggani oleh pengguna yang aktif untuk dijadikan nama pada *exchange* dan UUID versi 4 yang baru dibuat untuk memberi nama pada *queue*. Pseudocode untuk fungsi tersebut dapat dilihat pada kode sumber 4.7

```
deviceID = getSubscribedDeviceIDbyUser()
pushDeviceIDToArray()
sendtoServer(deviceIDArray, uuid4())
```

**Kode Sumber 4.7:** Pseudocode Aktivitas Client Saat Terkoneksi dengan Websocket

Setelah itu, selanjutnya *server* akan memproses data tersebut untuk pembuatan *queue* dan *exchange* agar data pada pub/sub *server* dapat disalurkan lewat *exchange* dan *queue* tersebut. *pseudocode* aktivitas *websocket server* saat pembuatan *queue* dan *exchange* lalu menyalurkan data pada *client* dapat dilihat pada kode sumber 4.8

```
if exchange does not exist:
  createExchage()
  if queue does not exist:
    createQueue()
    bindExchangetoQueue()
  else:
    bindExchangetoQueue()
else:
  pass

listenMessageFromRabbitMQServer()
sendMessageToClient()
```

**Kode Sumber 4.8:** Pseudocode Aktivitas Websocket Saat Pembuatan Queue dan Exchange Untuk Penyaluran Data ke Client

## 4.6 Implementasi *Database Server*

Sebagai media penyimpanan, sebuah *database* diperlukan untuk menyimpan data pengguna, perangkat, dan data berlangganan. Terdapat tiga tabel utama yang mewakili tiap entitas yang terlibat dalam sistem ini, yaitu: *users*, *devices*, dan oid selain itu, terdapat dua table *many-to-many* untuk menyimpan data pengguna yang telah berlangganan kepada tiap perangkat dan pengguna yang berlangganan kepada tiap OID (untuk mengetahui informasi apa saja yang ada pada tiap perangkat. tiap OID memiliki informasi yang berbeda).

Pada Tugas Akhir ini, sistem basis data yang digunakan adalah `Mysql Server` yang dimana `Mysql Server` termasuk kedalam RDBMS (*Relational Database Management System*). Berikut adalah rincian dari tabel yang diimplementasikan. rincian tabel *users* dapat dilihat pada tabel 4.2

**Tabel 4.2:** Rincian Tabel *users* pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | id | varchar(255) | Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4 |
| 2 | name | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna. |
| 3 | username | varchar(255) | Data yang berbentuk string. Digunakan untuk keperluan autentikasi. |
| 4 | password | varchar(255) | Data yang berbentuk string, implementasinya berupa hash. Digunakan untuk keperluan autentikasi. |
| 5 | email | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna |
| 6 | role | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna dan pembeda peran agar setiap user memiliki hak istimewa masing-masing. |

Terdapat juga tabel *devices* yang digunakan untuk menyimpan seluruh data perangkat. pada tabel ini terdapat lima kolom. rincian tabel *devices* dapat dilihat pada tabel 4.3

**Tabel 4.3:** Rincian Tabel *devices* pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | id | varchar(255) | Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4 |
| 2 | name | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan. |
| 3 | type | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan. |
| 4 | location | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan. |

**Tabel 4.3:** Rincian Tabel *devices* pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 5 | address | varchar(255) | Data yang berbentuk string, implementasinya barbentuk alamat IP dari tiap perangkat jaringan. Digunakan untuk mengkoleksi data pada publisher server. |

lalu terdapat juga tabel OID yang digunakan untuk menyimpan seluruh data informasi yang tersedia pada tiap perangkat. Pada tabel ini terdapat tiga kolom utama dan satu kolom *foreign key* yang terhubung dengan tabel *devices*. Rincian tabel OID dapat dilihat pada tabel 4.4

**Tabel 4.4:** Rincian Tabel OID pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | id | varchar(255) | Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4 |

**Tabel 4.4:** Rincian Tabel OID pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 2 | oid | varchar(255) | Data yang berbentuk string, implementasinya berbentuk OID (Object-Identifier). Digunakan mengkoleksi perangkat jaringan pada publisher server. |
| 3 | oidname | varchar(255) | Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan. |
| 4 | devices_id | varchar(255) | Merupakan foreign key dari id pata tabel `devices`. Data ini berbentuk string, nilai pada kolom ini berformat UUID versi 4. |

lalu terdapat juga tabel *subscribe* yang digunakan untuk menyimpan seluruh data pengguna (*user*) yang berlangganan informasi pada tiap perangkat. Tabel ini bersifat *many-to-many*, pada tabel ini terdapat dua kolom *foreign key* yang terhubung dengan tabel *devices* dan *users*. Rincian tabel *subscribe* dapat dilihat pada tabel 4.4

**Tabel 4.5:** Rincian Tabel *subscribe* pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | users_id | varchar(255) | Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel `users`, nilai pada kolom ini berformat UUID versi 4 |
| 2 | devices_id | varchar(255) | Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel `devices`, nilai pada kolom ini berformat UUID versi 4. |

terakhir, terdapat tabel *subscribeoid* yang digunakan untuk menyimpan seluruh data pengguna (*user*) yang berlangganan informasi pada tiap perangkat. Tabel ini bersifat *many-to-many*, pada tabel ini terdapat dua kolom *foreign key* yang terhubung dengan tabel OID dan *users*. Rincian tabel *subscribe*oid dapat dilihat pada tabel 4.6

**Tabel 4.6:** Rincian Tabel *subscribeoid* pada Database

| No | Kolom | Tipe Data | Keterangan |
|---|---|---|---|
| 1 | users_id | varchar(255) | Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel `users`, nilai pada kolom ini berformat UUID versi 4 |
| 2 | oid_id | varchar(255) | Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel `oid`, nilai pada kolom ini berformat UUID versi 4. |

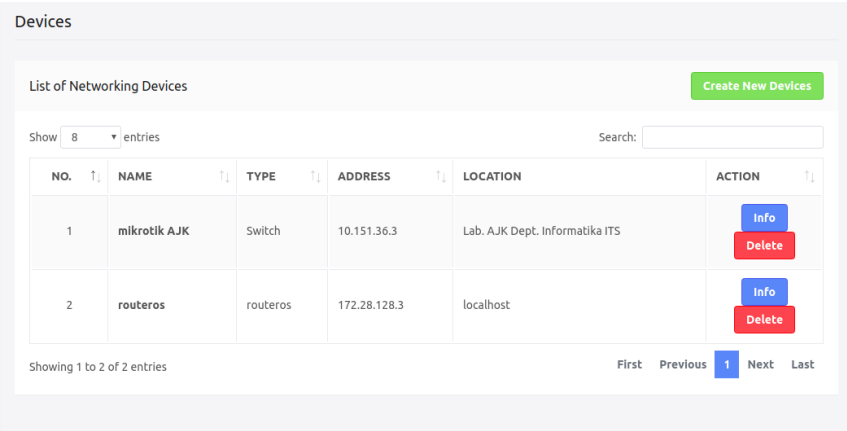
## 4.7 Implementasi Antarmuka

Antarmuka sistem dibangun dengan menggunakan `Bootstrap4` dan `Jquery` pada *frontend* dan kerangka kerja `Flask` pada *backend*nya. Antarmuka yang utama digunakan pada tugas akhir ini digunakan untuk mempermudah pengelolaan data perangkat dan halaman pemantauan perangkat jaringan. Antarmuka yang diimplementasikan pada tugas akhir ini adalah sebagai berikut:

- Menampilkan seluruh data perangkat yang terdaftar pada sistem
- Menampilkan rincian data perangkat jaringan yang terdaftarpada sistem
- Menampilkan data yang ingin dipantau pengguna pada sistem

### 4.7.1 Menampilkan seluruh data perangkat yang terdaftar pada sistem

Halaman ini berfungsi untuk menampilkan seluruh data yang terdaftar pada sistem. Seluruh data disajikan dalam bentuk tabel data yang dibangun dengan menggunakan bootstrap4, sehingga memungkinkan pengguna untuk mencari dan mengurutkan dapat pada tabel. Antarmuka daftar perangkat pada sistem ditunjukkan pada Gambar 4.1.



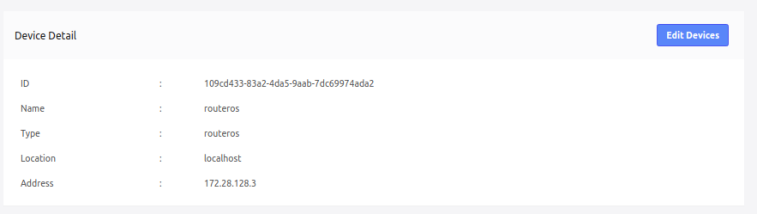

**Gambar 4.1:** Antarmuka Daftar Perangkat yang Tersedia

### 4.7.2 Menampilkan rincian data perangkat jaringan yang terdaftar pada sistem

Halaman ini berfungsi untuk menunjukkan informasi tiap perangkat. Informasi yang disajikan meliputi informasi umum tiap perangkat (nama perangkat, tipe perangkat, alamat perangkat dan lokasi perangkat), daftar pengguna yang berlangganan ke perangkat tersebut, dan daftar OID (informasi keadaan perangkat) yang tersedia pada perangkat tersebut.

Pada halaman ini pengguna dapat berlangganan kepada

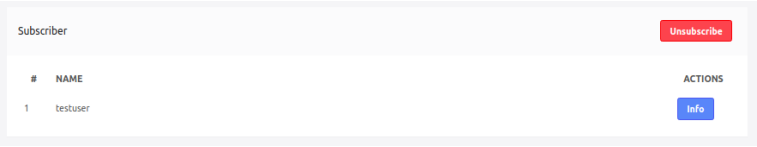
perangkat dengan menekan tombol *subscribe* pada kolom ”*Subscriber*”. setelah berhasil berlangganan kepada perangkat, sistem akan menampilkan tombol ”*subscribe*” pada kolom OID, dimana tombol tersebut berfungsi untuk memilih informasi apa saja yang ingin didapatkan oleh pengguna. Antarmuka informasi rincian perengkat ditunjukkan pada Gambar 4.2.




**Gambar 4.2:** Antarmuka Rincian Data Profil Perangkat

Dapat dilihat pada gambar diatas, antarmuka tersebut menampilkan rincian informasi dari perangkat yang terdaftar pada sistem. informasi yang disajikan antara lain adalah: ID perangkat (UUID), nama perangkat, tipe perangkat, lokasi perangkat dan alamat IP perangkat terserbut.

Selain informasi profil perangkat, terdapat kolom untuk melihat daftar pengguna yang berlangganaan ke perangkat tersebut. antarmuka daftar pengguna yang berlangganan ditunjukkan pada gambar 4.3.

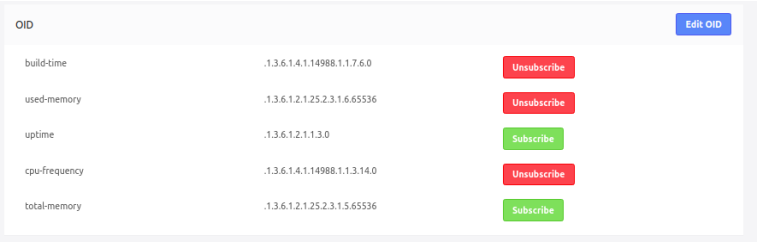



**Gambar 4.3:** Antarmuka Daftar Pengguna yang Berlangganan Kepada Perangkat

Pada antarmuka yang ditunjukkan oleh gambar 4.3 terdapat

tombol untuk belangganan, jika tombol tersebut ditekan maka *username* yang menekan akan berada di daftar tersebut.

Pada halaman ini juga terdapat sebuah kolom untuk menampilkan OID yang dapat dilihat pada gambar 4.4.

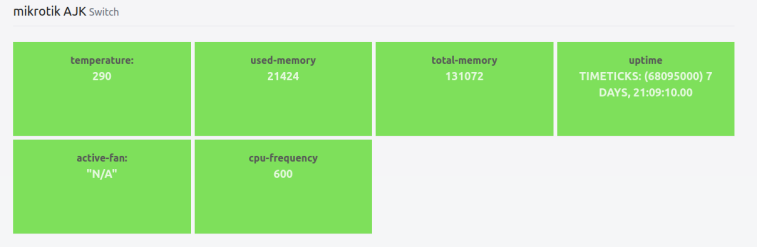



**Gambar 4.4:** Antarmuka Daftar OID yang Tersedia Pada Perangkat

Pada gambar 4.4 terdapat juga tombol berlangganan untuk pada tiap info yang terdaftar pada perangkat tersebut. tombol berlangganan tersebut digunakan untuk menyaring informasi yang ingin didapatkan oleh pengguna.

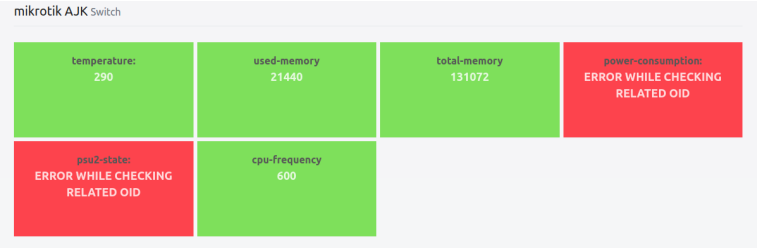
### 4.7.3 Menampilkan data yang ingin dipantau pengguna pada sistem

Pada halaman ini, pengguna dapat memantau atau melihat kondisi dari tiap perangkat yang telah dilanggani oleh tiap pengguna. Informasi yang disajikan pada halaman ini tergantung dari banyaknya informasi pada tiap perangkat yang dilanggani oleh pengguna. pada halaman ini juga, *websocket* bekerja. jika *client* (*browser*) terkoneksi dengan *websocket*, maka akan terdapat kotak berwarna hijau pada bagian atas halaman, sedangkan jika *client* tidak terhubung dengan *websocket* maka kotak tersebut akan berwarna merah. Antarmuka untuk menampilkan informasi keadaan perangkat ditunjukkan pada Gambar 4.5 dan pada gambar 4.6 menunjukkan keadaan perangkat dimana kondisinya kurang baik.




**Gambar 4.5:** Antarmuka Saat Sukses Menampilkan Informasi Keadaan Perangkat

Dapat dilihat pada gambar 4.5, panel info yang dipantai berwarna hijau yang menandakan bahwa data dapat di-*query* dengan baik. hal ini juga ditunjukkan dengan kata ”SNMP OK” yang berarti data pada perangkat jaringan telah di-*query* dengan baik lewat protokol SNMP.




**Gambar 4.6:** Antarmuka Saat Terjadi Kegagalan Menampilkan Informasi Keadaan Perangkat

Pada gambar 4.6, terdapat panel yang berwarna merah dan bertuliskan ”SNMP CRITICAL”, hal ini menujukkan bahwa data tidak ter-*query* dengan baik oleh sistem. umumnya kasus ini terjadi karena perangkat jaringan terkait tidak dapat menyediakan data tersebut.

*(Halaman ini sengaja dikosongkan)*

# BAB V

# PENGUJIAN DAN EVALUASI

## 5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu *server publisher*, satu *server publish/subscribe*, satu *server aplikasi*, satu *server API*, satu *server database*, satu agen SNMP dan satu komputer penguji. Semua *server* menggunakan *vitual machine* yang dipasang pada *Hypervisor* Proxmox, kecuali untuk pub/sub *server* yang diimplementasikan pada sebuah komputer yang memiliki spesifikasi yang dijelaskan pada tabel 5.1. Lalu, untuk komputer penguji menggunakan satu buah laptop sebagai *client* yang digunakan untuk menerima data yang dikirim oleh *publisher* dan melakukan skenario pengetesan pada REST API dengan aplikasi Apache JMeter. Pengujian dilakukan di Laboratoriom Arsitektur dan Jaringan Komputer Jurusan Teknik Informatika ITS.

Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1.

**Tabel 5.1:** Spesifikasi Komponen

| No | Komponen | Perangkat Keras | Perangkat Lunak |
|---|---|---|---|
| 1 | *Publisher Server* | 1 core processor, 512 MB RAM, 20GB HDD pada virtualisasi Proxmox | Ubuntu 16.04 LTS, Python2.7, Nagios |
| 2 | Pub/Sub *Server* | 4 core processor, 4096 MB RAM, 250GB HDD | Ubuntu 16.04 LTS, RabbitMQ |
| 3 | *Application Server* | 1 core processor, 512 MB RAM, 20GB HDD pada virtualisasi Proxmox | Ubuntu 16.04 LTS, Node.JS, Python 2.7 |

**Tabel 5.1:** Spesifikasi Komponen

| No | Komponen | Perangkat Keras | Perangkat Lunak |
|---|---|---|---|
| 4 | REST API | 1 core processor, 512 MB RAM, 20GB HDD pada virtualisasi Proxmox | Ubuntu 16.04 LTS, Docker 17.03.0-ce, Python 2.7 |
| 5 | *Database Server* | 1 *core processor*, 512 MB RAM, 20GB HDD pada virtualisasi Proxmox | Ubuntu 16.04 LTS, MySQL Server |
| 6 | Agen SNMP | Mikrotik *Routerboard Cloud Switch Series* | Agen SNMP |
| 7 | Komputer penguji | 4 *core processor* Intel i5-3210M, 8 GB RAM | Ubuntu 16.04 LTS, Apache JMeter 4.0 |

Untuk akses ke masing-masing komponen, digunakan IP *private* yang disediakan untuk masing-masing komponen tersebut. Detailnya ditunjukkan pada Tabel 5.2.

**Tabel 5.2:** IP dan Domain Server

| No | Server | IP dan Domain |
|---|---|---|
| 1 | *Publisher Server* | 10.151.36.97 |
| 2 | Pub/Sub *Server* | 10.151.36.98 |
| 3 | *Application Server* | 10.151.36.99 |
| 4 | REST API | 10.151.36.100 |
| 5 | *Database Server* | 10.151.36.101 |
| 6 | Agen SNMP | 10.151.36.3 |
| 7 | Komputer Penguji | 10.151.36.153 |

## 5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**
  Pengujian ini didasarkan pada fungsionalitas yang disajikan sistem.
- **Uji Performa**
  Pengujian ini untuk menguji kecepatan respon sistem terhadap sejumlah permintaan ke aplikasi secara bersamaan. Pengujian dilakukan dengan melakukan *benchmark* pada sistem.

### 5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas dibagi menjadi 2, yaitu uji fungsionalitas antarmuka aplikasi dan uji fungsionalitas *endpoint* REST API.

#### 5.2.1.1 Uji Fungsionalitas Antarmuka Aplikasi

Pengujuian ini dilakukan untuk memeriksa apakah semua fungsi yang berada pada aplikasi dapat dijalankan dengan benar. Pengujian dilakukan dengan cara mengakses antarmuka yang berhubungan dengan tugas akhir ini lalu menjalankan fitur yang disediakan pada tiap-tiap antarmuka.

Rancangan pengujian dan hasil yang diharapkan dapat dilihat pada tabel 5.3.

**Tabel 5.3:** Skenario Uji Fungsionalitas Antarmuka Aplikasi

| No | Fitur | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 1 | Autentikasi pengguna untuk masuk kedalam sistem. | Pengguna memasukkan *username* dan *password* masing-masing milik pengguna pada form yang telah disediakan. | Pengguna dapat masuk ke halaman utama aplikasi setelah menekan tombol ”*login*” |

**Tabel 5.3:** Skenario Uji Fungsionalitas Antarmuka Aplikasi

| No | Fitur | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 2 | Menampilkan seluruh data perangkat. | Pengguna menekan *menu* ”DEVICE MANAGEMENT” pada *sidebar*. | Sistem menampilkan seluruh data perangkat yang terdaftar pada sistem. data yang ditampilkan meliputi: nama perangkat, tipe perangkat, alamat perangkat dan lokasi perangkat. serta terdapat tombol informasi untuk melihat data masing-masing perangkat secara rinci dan tombol hapus untuk menghapus data perangkat yang terdaftar pada sistem. |

**Tabel 5.3:** Skenario Uji Fungsionalitas Antarmuka Aplikasi

| No | Fitur | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 3 | Menampilkan rincian data perangkat | Pengguna menekan tombol informasi yang tersedia pada tabel pada halaman menampilkan seluruh data perangkat. | Sistem menampilkan data perangkat terkait secara rinci. data yang ditampilkan meliputi: profil perangkat, pelanggan dari perangkat dan OID (Informasi yang disediakan pada perangkat terkait). |
| 4 | Menghapus data perangkat. | Pengguna menekan tombol hapus yang tersedia pada tabel pada halaman menampilkan seluruh data perangkat. | Sistem menghapus data perangkat terkait secara permanen. |
| 5 | Menyunting data perangkat. | Pengguna menekan tombol ubah data pada halaman rincian data perangkat lalu mengubah data yang tersedia pada form yang berisi data sebelumnya. | Sistem mengubah data yang lama dengan data yang baru dimasukkan oleh pengguna. |

**Tabel 5.3:** Skenario Uji Fungsionalitas Antarmuka Aplikasi

| No | Fitur | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 6 | Berlangganan Data Perangkat. | Pengguna menekan tombol ”Subscribe” yang tersedia pada halaman rincian data perangkat. | Sistem menandai bahwa perangkat atau OID (Informasi pada perangkat) yang terkait telat dilanggani. tombol akan berubah menjadi ”Unsubscribe” |
| 7 | Memantau kondisi perangkat yang telah dilanggani. | Pengguna menekan *menu* ”MONITOR” pada *sidebar*. | Sistem menampilkan kondisi dari seluruh perangkat yang telah dilanggani informasinya oleh pengguna. |

### 5.2.1.2 Uji Fungsionalitas Endpoint REST API

Pengujian ini dilakukan untuk memeriksa apakah seluruh fungsi dan *endpoint* yang berhubungan dengan tugas akhir ini dan tersedia pada sistem dapat bekerja dengan semestinya. Pengujian dilakukan dengan cara mengakses *endpoint* dengan metode tertentu disertai dengan *header* autentikasi JWT . Rancangan pengujian dan hasil yang diharapkan ditunjukkan dengan Tabel 5.4.

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 1 | /login. | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *username* dan *password*. | REST API mengembalikan respon berupa pesan berhasil dan token JWT jika berhasil, lalu mengembalikan pesan gagal jika gagal. |
| 2 | /devices. | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode GET. | jika berhasil, REST API mengembalikan respon berupa seluruh data yang tersedia pada sistem dalam bentuk json. tiap datanya meliputi: nama perangkat, tipe perangkat, alamat perangkat dan lokasi perangkat. Jika terjadi kegagalan, REST API akan mengembalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 3 | /devices/ <string:id> | Mengakses *endpoint* dengan header autentikasi JWT, metode GET dan menyertakan ID perangkat pada *endpoint*. | jika berhasil, REST API mengembalikan respon berupa seluruh data yang tersedia pada sistem dalam bentuk json. tiap datanya meliputi: nama perangkat, tipe perangkat, alamat perangkat, lokasi perangkat, *subscriber* perangkat dan OID (informasi yang tersedia pada perangkat). Jika terjadi kegagalan, REST API akan mengembalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 4 | /devices /create. | Mengakses *endpoint* dengan header autentikasi JWT dan metode POST, disertai dengan body bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *name*, *type*, *address* dan *location* | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |
| 5 | /devices /edit /<string:id> | Mengakses *endpoint* dengan *header* autentikasi JWT, metode POST, menyertakan ID perangkat pada *endpoint* dan disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *name*, *type*, *address* dan *location*. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 6 | /devices /delete | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode DELETE, disertai dengan body bertipe JSON yang dilengkapi dengan parameter ID perangkat. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |
| 7 | /oid /create | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oidname, oid dan devices_id | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 8 | /oid /edit | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oidname, oid dan devices_id | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |
| 9 | /oid /delete | Mengakses *endpoint* dengan header autentikasi JWT dan metode POST, disertai dengan body bertipe JSON yang dilengkapi dengan parameter parameter ID perangkat. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 10 | /subscribe /devices | Mengakses *endpoint* dengan header autentikasi JWT dan metode POST, disertai dengan body bertipe JSON yang dilengkapi dengan beberapa parameter seperti: device_id dan users_id. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |
| 11 | /unsubscribe /devices | Mengakses *endpoint* dengan header autentikasi JWT dan metode POST, disertai dengan body bertipe JSON yang dilengkapi dengan beberapa parameter seperti: device_id dan users_id. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |

**Tabel 5.4:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 12 | /subscribe /oid | Mengakses *endpoint* dengan header autentikasi JWT dan metode POST, disertai dengan body bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oid_id dan users_id. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |
| 13 | /unsubscribe /oid | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oid_id dan users_id. | jika berhasil, REST API mengembalikan respon berupa pesan berhasil. Jika gagal, maka REST API akan mengambalikan pesan gagal. |

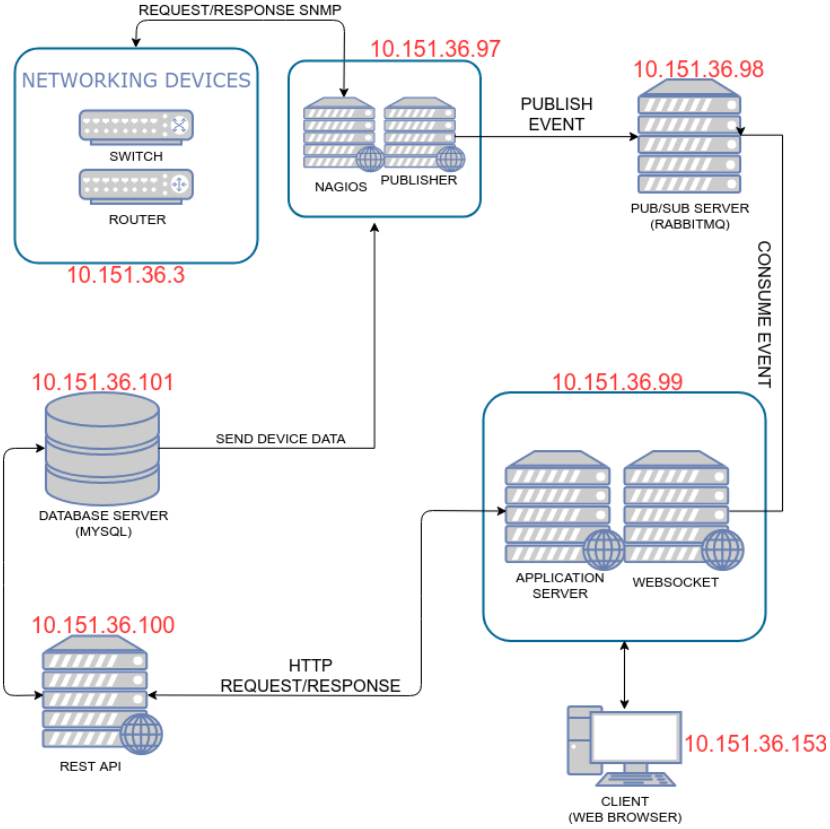
### 5.2.2 Skenario Uji Coba Performa

Uji performa dilakukan dengan dua langkah, yaitu uji performa REST api dan uji performa *publish/subscribe*. Arsitektur sistem yang dirancang untuk pengujian dapat dilihat pada gambar 5.1.

Uji coba REST API dilakukan dengan menggunakan satu buah laptop untuk melakukan akses secara bersamaan ke REST

API menggunakan aplikasi JMeter. Laptop akan mencoba mengaskses REST API yang sudah berjalan pada suatu *server*, dengan alamat IP 10.151.36.100.

Uji performa selanjutnya yaitu uji performa *publish/subscribe* dilakukan dengan cara menghitung waktu pengiriman. pengiriman dilakukan dari *publisher* yang berada pada *server* dengan alamat IP 10.151.36.97 menuju *subscriber* yang berada pada alamat IP 10.151.36.99.




**Gambar 5.1:** Arsitektur Sistem yang Digunakan Untuk Pengujian

#### 5.2.2.1 Uji Performa Kecepatan dan KeberhasilanREST API dalam Menangani *Request*

Pengujian dilakukan dengan mengukur jumlah waktu respon yang diperlukan oleh REST API dan persentase keberhasilan REST API untuk menyelesaikan *request* yang dilakukan oleh komputer penguji. Waktu yang diukur adalah perbedaan jarak antara *request* pertama dan terakhir dilakukan oleh klien yang mendapatkan balasan dari *server* dan persentase keberhasilan diukur dari banyaknya request yang berhasil dikirimkan dibagi total *request* yang dikirimkan dikalikan seratus persen.

Percobaan dilakukan dengan membuat *thread* untuk setiap akses ke REST API. Pengujian akan berlangsung bertahap mulai dari 300, 600, 900, 1200 dan 1500 *thread* dengan pengulangan 5 kali untuk setiap *event*. Dari hasil pengujian akan didapatkan waktu respon terhadap permintaan. Waktu tersebut digunakan untuk membuat grafik kesimpulan.

#### 5.2.2.2 Uji Performa Kecepatan Pengiriman Data Dari Publisher Menuju Subscriber

Pengujian dilakukan dengan menghitung waktu yang diperlukan *publisher* untuk mengirimkan data hingga sampai kepada *subscriber* melalui pub/sub *server*. Pengujian dilakukan dengan cara mengurangi waktu (*timestamp*) yang dideklarasikan saat *publisher* mengirimkan data dengan waktu (*timestamp*) saat pesan sampai di *subscriber* dan siap untuk dikirimkan kepada *client* melalui *websocket*. Waktu respon yang didapatkan dengan metode *publish/subscribe* dibandingkan dengan pengeambilan data perangkat jaringan menggunakan *plugin* SNMP.

## 5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

### 5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

#### 5.3.1.1 Uji Fungsionalitas Antarmuka Aplikasi

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.3. Hasil pengujian seperti tertera pada Tabel 5.5.

**Tabel 5.5:** Hasil Uji Coba Mengelola Aplikasi Berbasis Docker

| No | Uji Coba | Hasil |
|---|---|---|
| 1 | Pengguna memasukkan *username* dan *password* masing-masing milik pengguna pada *form* yang telah disediakan. | Sukses |
| 2 | Pengguna menekan *menu* "*DEVICE* MANAGEMENT" pada *sidebar*. | Sukses |
| 3 | Pengguna menekan tombol informasi yang tersedia pada tabel pada halaman menampilkan seluruh data perangkat. | Sukses |
| 4 | Pengguna menekan tombol hapus yang tersedia pada tabel pada halaman menampilkan seluruh data perangkat. | Sukses |
| 5 | Pengguna menekan tombol ubah data pada halaman rincian data perangkat lalu mengubah data yang tersedia pada *form* yang berisi data sebelumnya. | Sukses |

**Tabel 5.5:** Hasil Uji Coba Mengelola Aplikasi Berbasis Docker

| No | Uji Coba | Hasil |
|---|---|---|
| 6 | Pengguna menekan tombol ”*Subscribe*” yang tersedia pada halaman rincian data perangkat. | Sukses |
| 7 | Pengguna menekan *menu* ”*MONITOR*” pada *sidebar*. | Sukses |

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.3, hasil uji coba menunjukkan semua skenario berhasil ditangani.

### 5.3.1.2 Uji Fungsionalitas Endpoint REST API

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.2 dan pada Tabel 5.4. Hasil pengujian seperti tertera pada Tabel 5.6.

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil Harapan |
|---|---|---|---|
| 1 | /login. | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *username* dan *password*. | Sukses - REST API mengembalikan respon berupa pesan berhasil |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 2 | /devices. | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode GET. | Sukses - REST API mengembalikan respon berupa seluruh data yang tersedia pada sistem dalam bentuk json. tiap datanya meliputi: nama perangkat, tipe perangkat, alamat perangkat dan lokasi perangkat. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 3 | /devices/ <string:id> | Mengakses *endpoint* dengan *header* autentikasi JWT, metode GET dan menyertakan ID perangkat pada *endpoint*. | Sukses - REST API mengembalikan respon berupa seluruh data yang tersedia pada sistem dalam bentuk json. tiap datanya meliputi: nama perangkat, tipe perangkat, alamat perangkat, lokasi perangkat, *subscriber* perangkat dan OID (informasi yang tersedia pada perangkat). |
| 4 | /devices /create. | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *name*, *type*, *address* dan *location* | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 5 | /devices/edit /<string:id> | Mengakses *endpoint* dengan *header* autentikasi JWT, metode POST, menyertakan ID perangkat pada *endpoint* dan disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: *name*, *type*, *address* dan *location*. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |
| 6 | /devices /delete | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode DELETE, disertai dengan *body* bertipe JSON yang dilengkapi dengan parameter ID perangkat. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 7 | /oid /create | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oidname, oid dan devices_id | Sukses - REST API mengembalikan respon berupa pesan berhasil. |
| 8 | /oid /edit | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oidname, oid dan devices_id | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 9 | /oid /delete | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan parameter parameter ID perangkat. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |
| 10 | /subscribe /devices | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: device_id dan users_id. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 11 | /unsubscribe /devices | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: device_id dan users_id. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |
| 12 | /subscribe /oid | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oid_id dan users_id. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

**Tabel 5.6:** Skenario Uji Fungsionalitas REST API

| No | *Endpoint* | Uji Coba | Hasil |
|---|---|---|---|
| 13 | /unsubscribe /oid | Mengakses *endpoint* dengan *header* autentikasi JWT dan metode POST, disertai dengan *body* bertipe JSON yang dilengkapi dengan beberapa parameter seperti: oid_id dan users_id. | Sukses - REST API mengembalikan respon berupa pesan berhasil. |

### 5.3.2 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa REST API dilakukan dengan menghitung respon dari sistem dan keberhasilan sistem dalam menangani sejumlah permintaan (*request*) secara bersamaan. Pengujian dilakukan menggunakan Apache JMeter dengan menjalankan skenario *login*, membaca seluruh data perangkat jaringan, membuat data perangkat jaringan, menyunting data perangkat jaringan dan menghapus data perangkat jaringan. Skenario tersebut dijalankan secara bersama-sama oleh beberapa *thread* secara berulang-ulang sebanyak lima kali.

Jumlah *thread* dan request yang dilakukan oleh keseluruhan *thread*, dapat dilihat pada tabel 5.7

**Tabel 5.7:** Jumlah *Request* ke REST API

| ***Thread*** | **Jumlah *Request*** |
|---|---|
| 300 | 7500 |
| 600 | 15.000 |
| 900 | 22.500 |
| 1.200 | 30.000 |
| 1.500 | 37.500 |

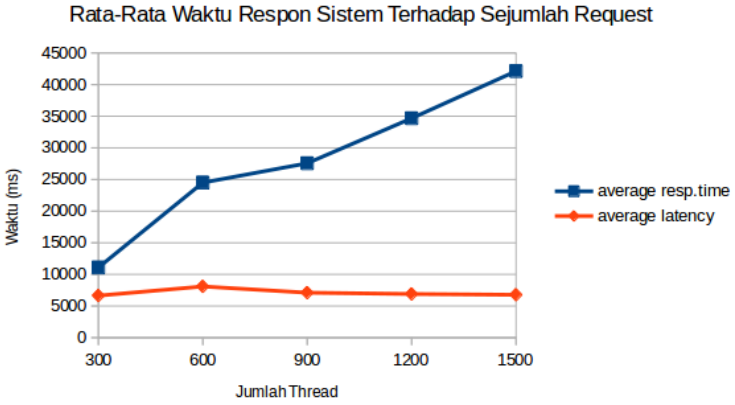
Untuk pengujuan performa *publish/subscribe*, dilakukan perhitungan waktu respon yang dibutuhkan untuk mengirimkan sebuah pesan dari *publisher* menuju *subscriber*. waktu respon dihitung dengan cara mengurangi waktu (*timestamp*) yang dideklarasikan saat *publisher* mengirimkan data dengan waktu (*timestamp*) saat pesan sampai di *subscriber* dan siap untuk dikirimkan kepada *client* melalui *websocket*. Waktu respon yang didapatkan dengan metode *publish/subscribe* dibandingkan dengan pengeambilan data perangkat jaringan menggunakan *plugin* SNMP.

### 5.3.2.1 Uji Performa Kecepatan dan Keberhasilan REST API dalam Menangani *Request*

Pengujian Kecepatan REST API dalam menangani *request* dilakukan dengan *request* yang diberikan kepada REST API sesuai dengan tabel 5.7, dari tabel tersebut didapatkan hasil rata-rata waktu respon beserta *latency* yang dapat dilihat pada tabel 5.8 dan direpresentasikan dalam grafik yang dapat dilihat pada gambar 5.2

**Tabel 5.8:** Hasil Uji Coba Mengetahui Respon REST API

| ***Thread*** | **Waktu Respon (ms)** | ***Latency*** |
|---|---|---|
| 300 | 11,087,59 ms | 6.635,03 |
| 600 | 24,503,60 ms | 8.084,38 |
| 900 | 27,564,91 ms | 7.099,12 |
| 1.200 | 34,679,32 ms | 6.896,99 |
| 1.500 | 42.133,07 ms | 6.773,26 |

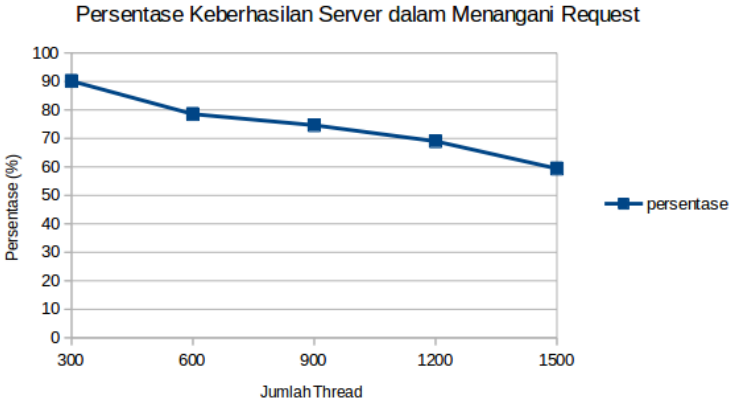



**Gambar 5.2:** Grafik Kecepatan Menangani *Request*

Hasil Pengujian persentase keberhasilan REST API dalam menagani sejumlah permintaan (*request*) dapat dilihat pada tabel 5.9 dan direpresentasikan pada grafik yang dapat dilihat pada gambar 5.3.

**Tabel 5.9:** Skenario Uji Fungsionalitas REST API

| Thread | Jumlah Request | Request Berhasil | Persentase Keberhasilan |
|---|---|---|---|
| 300 | 7.500 | 6.759 | 90,12% |
| 600 | 15.000 | 11.781 | 78,54% |
| 900 | 22.500 | 16.793 | 74,63% |
| 1200 | 30.000 | 20.703 | 69,01% |
| 1500 | 37.500 | 22.273 | 59,39% |




**Gambar 5.3:** Grafik Keberhasilan Menangani *Request*

Dapat dilihat dari dua pengujian diatas, bahwa jumlah pengguna (client) yang mengakses data REST API sangat berpengaruh kepada performa REST API itu sendiri.

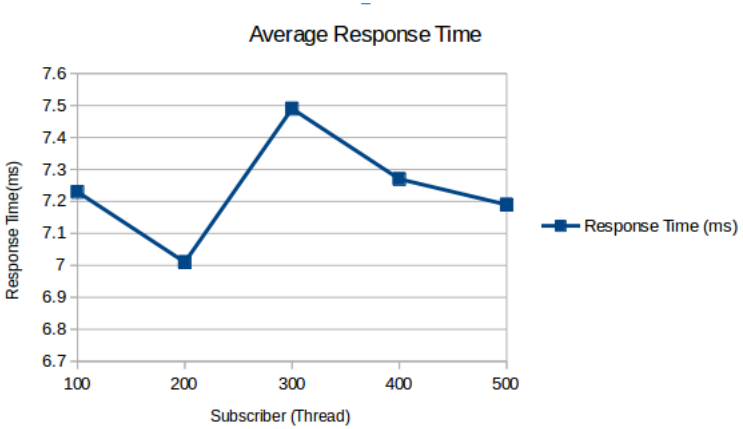
#### 5.3.2.2 Uji Performa Kecepatan Pengiriman Data Dari Publisher Menuju Subscriber

Pengujian performa kecepatan pengiriman data dari *publisher* menuju *subscriber* dengan metode *publish/subscribe* dilakukan dengan cara mengirimkan 1000 data keadaan perangkat jaringan dari *publisher* dan menyiapkan *subcriber* sejumlah 1000 yang siap menerima data dan representasikan dalam bentuk *thread* yang dinaikkan jumlahnya secara berlipat ganda. hasil pengukuran *response time* pengiriman data dari *publisher* menuju *subscriber* dilakukan dengan cara mengurangi waktu (*timestamp*) saat *subscriber* menerima data dengan waktu (*timestamp* )saat *publisher* mengirimkan data, setelah data terkirim, setelah itu seluruh data *response time* dicari rata-ratanya. hasil pengian *response time publish/subscribe* dapat dilihat pada tabel 5.10 dan pada grafik 5.4

**Tabel 5.10:** Tabel Hasil Pengujian *Publish/Subscribe*

| ***Subscriber (Thread)*** | ***Average Response Time (ms)*** |
|---|---|
| 100 | 7.23 ms |
| 200 | 7.01 ms |
| 300 | 7.49 ms |
| 400 | 7.27 ms |
| 500 | 7.19 ms |

Dari hasil uji coba, jumlah *subscriber* yang menerima data tidak berpengaruh terhadap waktu pengiriman data. hal ini menunjukkan bahwa *publish/subscribe* berjalan dengan baik dan menunjukkan bahwa dengan menggunakan metode *publish/subscribe*, data perangkat jaringan dapat diterima dengan waktu yang relatif konstan walaupun jumlah penerima data semakin banyak. Hasil uji coba performa *response time* ditunjukkan oleh grafik pada Gambar 5.4.




**Gambar 5.4:** Grafik *Response Time* Pengiriman Data dengan *Metode Publish/Subscribe*

# BAB VI

# PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

## 6.1 Kesimpulan

Dari proses perencangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Sistem dapat mengirimkan data keadaan perangkat jaringan dengan metode *publish/subscribe*.
2. RESTful API dapat berfungsi dengan baik, dapat dilihat dari semua *endpoint* yang telah diuji secara fungsional dan secara performa.
3. Aplikasi dapat berfungsi dengan baik, dapat dilihat dari semua *endpoint* yang telah diuji secara fungsional.
4. semakin banyak *request* yang ditangani dalam waktu yang sama, semakin lama juga waktu yang dibutuhkan yang dibutuhkan untuk menangani sebuah *request*.
5. semakin banyak *request* yang ditangani dalam waktu yang sama, semakin banyak pula *request* yang tidak berhasil ditangani.
6. Performa *publish/subscribe* dalam kecepatan pengiriman data tidak dipengaruhi oleh banyaknya jumlah penerima data (*subscriber*).

## 6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Meningkatkan sumber daya (spesifikasi) *server*, agar server dapat bekerja lebih baik dalam menangani banyak *request* dalam waktu yang bersamaan.
2. Memasang *load balancer* dan menambah *worker* pada setiap komponen *server* agar dapat bekerja lebih baik dalam menangani *request* yang banyak dalam waktu yang bersamaan.

# DAFTAR PUSTAKA


[1] L. Vargas, L. I. W. Pesonen, E. Gudes, dan J. Bacon, "Transactions in Content-Based Publish/Subscribe Middleware," 2007.

[2] Y. Hu dan W. Cheng, "Research and Implementation of Campus Information Push System Based on WebSocket," 2017.

[3] A. Roohi, K. Raeisifard, dan S. Ibrahim, "An Application for Management and Monitoring the Data Centers Based on SNMP," 2014.

[4] J. Case, J. Davin, M. Fedor, dan M. Schoffstall, "Internet Network Management Using the Simple Network Management Protocol," 1989.

[5] "What is an Object Identifier (OID)?" *IEEE STANDARDS ASSOCIATION*, 2014.

[6] J. Renita dan N. E. Elizabeth, "Network's Server Monitoring and Analysis Using Nagios," 2017.

[7] L. Richardson dan S. Ruby, "Restful Web Services," hal. xvi–xvii, 2016.

*(Halaman ini sengaja dikosongkan)*

# LAMPIRAN A

# INSTALASI PERANGKAT LUNAK

## Pemasangan SNMP Nagios

Proses pemasangan SNMP Nagios dilakukan sesuai tahap berikut:

- Membuat user linux khusus Nagios
  Langkah ini dilakukan untuk menambahkan user pada linux dengan nama ”nagios”. Untuk melakukannya, jalankan perintah berikut:

```
$ sudo useradd nagios
$ sudo groupadd nagcmd
```

  perintah diatas juga sekaligus membuat group user dengan nama ”nagcmd”. setelah grup berhasil dibuat, Lnagkah berikutnya adalah memasukkkan user ”nagios” ke group ”nagcmd” dengan perindah berikut:

```
$ sudo usermod –a –G nagcmd nagios
```

- Mengunduh Nagios
  Untuk mengunduh Nagios Core, jalankan perintah berikut:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
    libgd2-xpm-dev openssl libssl-dev
    unzip
$ cd ~
$ curl -L -O https://assets.nagios.com/
    downloads/nagioscore/releases/nagios
    -4.3.4.tar.gz
```

- Pemasangan Nagios
  Langkah terakhir adalah memasang Nagios itu sendiri, jalankan perintah berikut untuk memasang Nagios Core:

```
$ tar zxf nagios-*.tar.gz
$ cd nagios-* libgd2-xpm-dev openssl
    libssl-dev unzip
$ ./configure --with-nagios-group=
    nagios --with-command-group=nagcmd
$ make all
$ sudo make install
$ sudo make install-commandmode
$ sudo make install-init
$ sudo make install-config
```

  Setelah Nagios terpasang, kita dapat menggunakan plugin snmp yang terletak pada direktori `/usr/local/nagios/bin/libexec/check_snmp`.

**Pemasangan RabbitMQ Sebagai Pub/Sub Server**

Untuk memasang RabbitMQ sebagai publish/subscribe server, jalankan perintah dibawah ini:

- Pemasangan Program Prasyarat RabbitMQ
  sebelum memasang RabbitMQ diperlukan beberapa program prasyarat yang harus terpasang seperti `erlang`, `init-system-helpers`, `socat`, `adduser` dan `logrotate`. pemasangan program prasayarat dilakukan dengan menjalankan perintah berikut:

```
$ wget http://packages.erlang-solutions
    .com/site/esl/esl-erlang/
    FLAVOUR_1_general/esl-erlang_21.0-1~
    ubuntu~xenial_amd64.deb
$ sudo dpkg -i esl-erlang_21.0-1~ubuntu
    ~xenial_amd64.deb
$ sudo apt-get -f install
$ sudo apt-get install socat
$ sudo apt-get install logrotate
```

- Pemasangan RabbitMQ
  Setelah seluruh program prasaray terpasang, langkah selanjutnya adalah memasang RabbitMQ. untuk memasang RabbitMQ, jalankan perintah dibawah ini:

```
$  wget https://github.com/rabbitmq/
    rabbitmq-server/releases/download/v3
    .7.6/rabbitmq-server_3.7.6-1_all.deb
$ sudo dpkg -i rabbitmq-server_3.7.6-1
    _all.deb
```

*(Halaman ini sengaja dikosongkan)*

## BIODATA PENULIS

**Afif Ridho Kamal P**, akrab dipanggil Afif lahir pada tanggal 3 Oktober 1996 di Jakarta. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki hobi antara lain makan dan bermain basket. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Dalam Negeri Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2 dan Kepala Departemen Dalam Negeri Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-3. Pernah menjadi staff Perlengkapan dan Transportasi Schematics tahun 2014 dan 2015. Selain itu penulis pernah menjadi asisten dosen dan asisten praktikum pada mata kuliah Sistem Operasi dan Jaringan Komputer.