# IMPLEMENTASI SISTEM PEMANTAUAN SUHU DAN KELEMBAPAN UDARA BERBASIS PROTOKOL AMQP

**SKRIPSI**

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Edgar Juvianno Santoso
NIM: 135150200111012

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

# PENGESAHAN

IMPLEMENTASI SISTEM PEMANTAUAN SUHU DAN KELEMBAPAN UDARA BERBASIS PROTOKOL AMQP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Edgar Juvianno Santoso
NIM: 135150200111012

Skripsi ini telah diuji dan dinyatakan lulus pada
26 Desember 2018
Telah diperiksa dan disetujui oleh:

| Dosen Pembimbing I | Dosen Pembimbing II |
|---|---|
| Rakhmadhany Primananda, S.T, M.Kom | Kasyful Amron, S.T, M.Sc |
| NIK: 20160986 0406 1 001 | NIP: 19750803 200312 1 003 |

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

# PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 24 Desember 2018

Edgar Juvianno Santoso
NIM: 135150200111012

# KATA PENGANTAR

Puji syukur atas karunia Allah SWT yang Maha Pengasih lagi Maha Penyayang karena dengan rahmat dan karuniaNya, penulis dapat menyelesaikan skripsi yang berjudul *"**IMPLEMENTASI SISTEM PEMANTAUAN SUHU DAN KELEMBAPAN UDARA BERBASIS PROTOKOL AMQP"*** sebagai salah satu syarat memperoleh gelar Sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya.

Penulis menyadari bahwa dalam menyelesaikan skripsi ini tidak lepas dari peran berbagai pihak yang telah memberi semangat, doa, bimbingan serta kritik dan nasehatnya. Dalam kesempatan ini, penulis ingin mengucapkan rasa terima kasih kepada:

1. Allah SWT yang telah memberikan rahmat dan karuniaNya dalam pelaksanaan skripsi ini.
2. Bapak Rakhmadhany Primananda, S.T, M.Kom selaku dosen pembimbing I yang telah sabar dalam membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
3. Bapak Kasyful Amron, S.T, M.Sc selaku dosen pembimbing II yang telah sabar dalam membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
4. Seluruh Dosen Program Studi Informatika Fakultas Ilmu Komputer Universitas Brawijaya atas ilmu yang sudah diberikan kepada penulis.
5. Ayah Amin Santoso dan ibu Rini Trianawati atas segala dukungan, nasehat, motivasi dan doa yang telah dipanjatkan untuk menyemangati penulis dalam menyelesaikan skripsi ini.
6. Seluruh teman-teman Fakultas Ilmu Komputer angkatan 2013 Universitas Brawijaya yang telah memberikan saran dan motivasi bagi penulis untuk menyelesaikan skripsi ini.
7. Seluruh pihak yang belum tersebutkan diatas, atas segala dukungan, motivasi dan doa dalam menyelesaikan skirpsi ini.

Penulis menyadari masih banyak kekurangan dalam penyusunan skripsi ini, oleh karena itu, saran dan kritik yang membangun dari semua pihak sangat dibutuhkan demi kesempurnaan selanjutnya.

Malang, 24 Desember 2018

Edgar Juvianno Santoso

edgarjuvianno2@gmail.com

# ABSTRAK

Kualitas udara dalam ruangan merupakan masalah yang perlu mendapat perhatian karena akan berpengaruh terhadap kesehatan manusia. Untuk menjaga kualitas udara dalam ruangan diperlukan adanya sebuah sistem yang dapat memantau suhu dan kelembapan udara dalam ruangan. Sistem pemantauan ini mampu diwujudkan dengan menggunakan teknologi *Internet of Things* (*IoT*). Dalam menerapkan teknologi *IoT*, dibutuhkan protokol komunikasi data untuk berbagi data antar setiap *node* yang ada pada infrastruktur *IoT*. Salah satu protokol komunikasi data yang sering digunakan adalah *AMQP* (*Advanced Message Queueing Protocol*). Penggunaan protokol *AMQP* pada penelitian ini didukung dengan kondisi jaringan rumah yang dianggap memenuhi kebutuhan dari *AMQP*. Dengan dibuatnya sistem pemantauan suhu dan kelembapan udara, diharapkan nantinya mampu memantau kualitas udara didalam ruangan. Penelitian ini memberikan hasil bahwa implementasi protokol *AMQP* pada sistem pemantauan suhu dan kelembapan udara berhasil dilakukan dan dari hasil pengujian keandalan sistem menunjukkan bahwa sistem yang dibangun mampu menangani hingga 210 *Producer* dengan hasil penggunaan memori pada perangkat *Broker* sebesar 0.4132 *MB* dan perangkat *Consumer* sebesar 0.8345 *MB*.

Kata kunci: *internet of things, sistem pemantauan, AMQP*

# ABSTRACT

*Indoor air quality is a problem that needs attention because it will affect human health. To maintain indoor air quality, there is a need for a system that can monitor temperature and humidity in the room. This monitoring system can be realized by using the Internet of Things (IoT) technology. In implementing IoT technology, a data communication protocol is needed to share data between each node in the IoT infrastructure. One of the frequently used data communication protocols is AMQP (Advanced Message Queuing Protocol). The use of the AMQP protocol in this study is supported by the condition of home networks that are considered to meet the needs of AMQP. With air temperature and humidity monitoring system, it is expected that the air quality in the room can be monitored properly. This study provides results that the implementation of the AMQP protocol on air temperature and humidity monitoring systems was successfully carried out and from the test results of the system reliability testing showed that the system was able to handle up to 210 Producers with Broker's devices memory usage of 0.4132 MB and Consumer's devices at 0.8345 MB.*

*Keywords*: *internet of things, monitoring system, AMQP*

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# DAFTAR LAMPIRAN

# BAB 1 PENDAHULUAN

## 1.1 Latar belakang

Kualitas udara dalam ruangan perlu mendapat perhatian karena merupakan salah satu faktor yang berpengaruh terhadap kesehatan manusia. Terdapat berbagai macam faktor yang dapat mempengaruhi kualitas udara di dalam ruangan, salah satunya adalah tingkat kelembapan. Kelembapan udara yang rendah dapat mengakibatkan terjadinya gejala *Sick Building Syndrome (SBS)* seperti iritasi mata, iritasi tenggorokan dan batuk-batuk. Selain itu rendahnya kelembapan udara dapat meningkatkan kerentanan terhadap penyakit infeksi, serta penyakit asthma. Jika kondisi udara ruang yang terlalu lembab dapat menyebabkan tumbuhnya bermacam-macam jamur dan spora (Jayanti, 2016).

Pentingnya kesehatan manusia membuat kualitas udara dalam ruangan harus tetap terjaga. Untuk itu, diperlukan adanya sebuah sistem yang dapat memantau suhu dan kelembapan udara dalam ruangan. Sistem pemantauan ini mampu diwujudkan dengan menggunakan teknologi *Internet of Things* (*IoT*), *IoT* merupakan sebuah konsep infrastruktur jaringan global atau lokal yang memanfaatkan konektivitas internet. Dengan kemampuan seperti berbagi data, kontrol jarak jauh dan sebagainya (Anon., 2013). Penelitian terbaru yang dilakukan dalam *Internet of Things (IoT)* menciptakan berbagai teknologi untuk mengumpulkan observasi *real-world* dengan menghubungkan sensor, aktuator, *Radio-Frequency Identification (RFID)* dan telepon seluler di *Web* (B. Babovic, et al., 2016). Sehingga teknologi ini memungkinkan pembangunan sistem pemantauan suhu dan kelembapan udara.

Pembangunan sistem pemantauan suhu dan kelembapan udara yang menerapkan teknologi *IoT* membutuhkan protokol komunikasi data sebagai sarana untuk berbagi data antar setiap *node*. Terdapat berbagai macam protokol komunikasi data yang dapat digunakan dalam arsitektur *IoT*, salah satu protokol yang sering digunakan adalah *AMQP* (*Advanced Message Queueing Protocol*). *AMQP* merupakan standar terbuka yang dirancang untuk mendukung perpesanan yang handal dan berkinerja tinggi melalui Internet. Protokol ini digunakan dalam pesan klien/server dan manajemen perangkat *IoT*. *AMQP* memungkinkan pengiriman pesan yang terenkripsi dan interoperabilitas (dapat dijalankan pada berbagai macam sistem operasi) antara organisasi dan aplikasi. *AMQP* memiliki keunggulan yaitu efisien, portabel, *multichannel* dan aman. Protokol biner menawarkan autentikasi dan enkripsi dengan cara *SASL* atau *TLS*, bergantung pada protokol *transport* seperti *TCP*. Protokol perpesanan yang cepat dan fitur pengiriman yang terjamin dengan *acknowledgement* saat pesan diterima (Rouse, 2018).

Penelitian sebelumnya yang berjudul *A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks* yang dilakukan oleh E. Luzuriaga, et al membandingkan protokol *AMQP* dan *MQTT* pada sebuah jaringan yang tidak stabil. Peneliti merancang eksperimen dimana *Producer* atau *Publisher* mengirim pesan dengan ukuran dan frekuensi pengiriman yang telah ditentukan

kepada *Broker*. Pada skenario yang digunakan oleh peneliti, *Consumer* atau *Subscriber* terhubung kepada *Broker* dan selalu siap untuk melakukan *consume* atau mengambil pesan. *Message Broker* dan klien *Consumer* atau *Subscriber* dieksekusi pada komputer yang sama, *Producer* atau *Publisher* terhubung pada sebuah akses poin jaringan nirkabel (*WiFi*) yang sama.

Kesimpulan dari E. Luzuriaga, et al menyatakan bahwa *MQTT* merupakan protokol yang cocok untuk digunakan pada kondisi jaringan yang tidak stabil, sedangkan protokol *AMQP* cocok digunakan pada kondisi jaringan yang lebih stabil dan dapat menyediakan *resource* yang besar (E. Luzuriaga, et al., 2015). Berdasarkan kesimpulan tersebut, penelitian ini akan menerapkan protokol *AMQP* sebagai protokol komunikasi data pada sistem pemantauan suhu dan kelembapan udara yang nantinya akan berjalan pada sebuah jaringan nirkabel yang stabil dan mampu untuk menyediakan *resource* yang cukup.

Berdasarkan permasalahan dan penelitian sebelumnya, penelitian ini ingin menggunakan protokol *AMQP* sebagai protokol pengiriman data pada sistem pemantauan suhu dan kelembapan udara. Penggunaan protokol *AMQP* pada penelitian ini didukung dengan kondisi jaringan rumah yang dianggap memenuhi kebutuhan dari *AMQP* (E. Luzuriaga, et al., 2015). Dalam penerapan protokol *AMQP*, mikrokomputer Raspberry Pi 3 dan sensor DHT11 akan berperan sebagai *Producer* yang mengambil data suhu dan kelembapan udara kemudian mengirimkan data tersebut kepada *Broker*. *Broker* tersebut akan berjalan pada sebuah mikrokomputer Raspberry Pi 3 dan *Consumer* yang berperan sebagai pengambil data dari *Broker* akan berjalan pada sebuah laptop. Dengan dibuatnya sistem pemantauan suhu dan kelembapan udara, diharapkan kualitas udara di dalam ruangan mampu terpantau dengan baik dan hasil pemantauan tersebut dapat ditampilkan pada sebuah aplikasi *web*.

## 1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan dan identifikasi masalah yang telah ditemukan, maka dirumuskan beberapa permasalahan yaitu:

1. Bagaimana penerapan protokol *AMQP* pada Sistem Pemantauan Suhu dan Kelembapan Udara?
2. Bagaimana performa dari Sistem Pemantauan Suhu dan Kelembapan Udara yang menerapkan protokol *AMQP*?

## 1.3 Tujuan

Berdasarkan latar belakang yang telah dijabarkan, maka diharapkan penelitian ini dapat mendapatkan beberapa tujuan seperti:

1. Menerapkan protokol *AMQP* pada Sistem Pemantauan Suhu dan Kelembapan Udara.
2. Menganalisis hasil uji dari penerapan protokol *AMQP* pada Sistem Pemantauan Suhu dan Kelembapan Udara.

## 1.4 Manfaat

Manfaat dari penelitian ini adalah:

1. Penelitian ini diharapkan dapat membantu memahami sistem konsep dari protokol *AMQP* dalam Sistem Pemantauan Suhu dan Kelembapan Udara Berbasis Protokol *AMQP*.
2. Dengan penelitian ini diharapkan dapat memberikan hasil analisis dari penerapan protokol *AMQP* dari sistem yang dibuat.

## 1.5 Batasan masalah

Berdasarkan pada rumusan masalah maka didapatkan batasan masalah yang akan dianalisa pada laporan ini yaitu:

1. Perangkat sensor yang akan digunakan adalah sensor temperatur suhu dan kelembapan udara DHT11.
2. Perangkat *platform* yang akan digunakan adalah Raspberry Pi 3.
3. *Broker* yang digunakan adalah RabbitMQ.
4. *Producer* dan *Consumer* menggunakan klien library Pika.
5. *Web framework* yang digunakan adalah Tornado.

## 1.6 Sistematika pembahasan

**BAB I : PENDAHULUAN**

Bab ini memuat tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika pembahasan penelitian.

**BAB II : LANDASAN KEPUSTAKAAN**

Bab ini membahas tentang kajian pustaka dan dasar teori yang mendasari implementasi protokol *AMQP* pada sistem pemantauan suhu dan kelembapan udara.

**BAB III : METODOLOGI**

Bab ini menguraikan tentang metode dan langkah kerja yang terdiri dari studi literatur, analisis kebutuhan simulasi, perancangan sistem, implementasi dan analisis serta pengambilan kesimpulan.

**BAB IV : REKAYASA KEBUTUHAN**

Bab ini memuat tentang deskripsi dari sistem dan kebutuhan dari sistem yang terbagi menjadi kebutuhan antarmuka eksternal, kebutuhan fungsional dan kebutuhan non-fungsional.

**BAB V : PERANCANGAN DAN IMPLEMENTASI**

Bab ini membahas tentang perancangan dan implementasi dari sistem pada penelitian ini, yang dimulai dari gambaran umum sistem, perancangan tiap-tiap komponen yang dibutuhkan oleh sistem, implementasi pada tiap-tiap komponen sistem.

**BAB VI : PENGUJIAN DAN ANALISIS**

Bab ini menjelaskan tentang hasil dari pengujian sistem ini dan analisis dari hasil pengujian yang telah dilakukan untuk mengetahui apakah kebutuhan fungsional dan non-fungsional sistem telah terpenuhi.

**BAB VII : PENUTUP**

Bab ini memuat tentang kesimpulan dan saran dari penenlitian yang telah dilakukan dari proses perancangan, implementasi dan pengujian.

# BAB 2 LANDASAN KEPUSTAKAAN

## 2.1 Kajian Pustaka

Kajian pustaka berisi dasar-dasar penelitian yang mendukung usulan peneliti yang akan dibandingkan dengan penelitian-penelitian sebelumnya. Penelitian sebelumnya menggunakan beberapa objek yang sama yaitu, penerapan sistem pemantauan suhu dan kelembapan udara yang hasilnya ditampilkan melalui sebuah halaman *web* dan implementasi protokol *AMQP*.

Penelitian sebelumnya yang berjudul *A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks* yang dilakukan oleh E. Luzuriaga, et al membandingkan protokol *AMQP* dan *MQTT* pada sebuah jaringan yang tidak stabil. Peneliti merancang eksperimen dimana *Producer* atau *Publisher* mengirim pesan dengan ukuran dan frekuensi pengiriman yang telah ditentukan kepada *Broker*. Pada skenario yang digunakan oleh peneliti, *Consumer* atau *Subscriber* terhubung kepada *Broker* dan selalu siap untuk melakukan *consume* atau mengambil pesan. *Message Broker* dan klien *Consumer* atau *Subscriber* dieksekusi pada komputer yang sama, *Producer* atau *Publisher* terhubung pada sebuah akses poin jaringan nirkabel (*WiFi*) yang sama.

Kesimpulan dari E. Luzuriaga, et al menyatakan bahwa *MQTT* merupakan protokol yang cocok untuk digunakan pada kondisi jaringan yang tidak stabil, sedangkan protokol *AMQP* cocok digunakan pada kondisi jaringan yang lebih stabil dan dapat menyediakan *resource* yang besar (E. Luzuriaga, et al., 2015). Berdasarkan kesimpulan tersebut, penelitian ini akan menerapkan protokol *AMQP* sebagai protokol komunikasi data pada sistem pemantauan suhu dan kelembapan udara yang nantinya akan berjalan pada sebuah jaringan nirkabel yang stabil dan mampu untuk menyediakan *resource* yang cukup. Penelitian ini bertujuan untuk mengetahui performa dari protokol *AMQP* yang akan diterapkan.

Pembahasan mengenai evaluasi performa terhadap beberapa protokol yang sering digunakan pada arsitektur *IoT* yaitu, *MQTT*, *AMQP*, *XMPP*, *DDS* dan performa *platform web* dengan protokol *WebSocket*, *HTTP Streaming* dan *Socket* yang dilakukan oleh B. Babovic, Protic, & Milutinovic dengan judul *Web Performance Evaluation for Internet of Things Applications*. Peneliti melakukan simulasi infrastruktur dimana sensor yang memberikan data terletak pada sebuah komponen *gateway* dan sensor mengirimkan pesan tersebut kepada *message Broker*. *Gateway* juga mengirimkan data kepada klien JavaScript yang berjalan pada *platform HTML* standar. Peneliti menggunakan library Node.js untuk mengimplementasikan server *WebSocket*, sehingga peneliti menggunakan bahasa pemrograman Java dan JavaScript. Sedangkan pengimplementasian protokol *AMQP* menggunakan *message Broker Apache Qpid* 6.03 yang berbasis Java, pada bagian *Producer*/*Publisher* menggunakan library Rabbit Java Client dan klien Kaazing JavaScript *AMQP* yang diperlukan oleh aplikasi *web* untuk mengambil data dari *message Broker* (B. Babovic, et al., 2016).

Berdasarkan penelitian yang dilakukan oleh B. Babovic, Protic, & Milutinovic, penelitian ini akan menerapkan sebuah sistem pemantauan suhu dan kelembapan udara dengan menggunakan protokol *AMQP* dan *WebSocket*. Penelitian ini ingin

mencoba menerapkan protokol *AMQP* dengan menggunakan *message broker* RabbitMQ karena menerapkan protokol *AMQP* 0-9-1. Sedangkan pada sisi klien *Producer* dan *Consumer* menggunakan library Pika sehingga aplikasi dapat ditulis menggunakan bahasa pemrograman Python dan menggunakan library Tornado untuk menerapkan server *WebSocket*.

Pada penelitian yang berjudul Implementasi Protokol Websocket Pada Perangkat Non IP Berbasis NRF24L01 (Studi Kasus: Sistem Monitoring Suhu dan Kontroling Lampu LED), dijelaskan mengenai penerapan protokol *WebSocket* dan perangkat radio pada sistem pemantauan dan kontroling. Peneliti merancang sebuah arsitektur dimana, sensor atau aktuator dijalankan pada sebuah mikrokomputer dan data yang didapatkan dari sensor dikirimkan menggunakan modul NRF24L01 menuju *bridging device* yang berjalan pada sebuah mikrokomputer. *Bridging device* tersebut terdiri dari *bridge* dan *web* server, lalu data yang ada pada *bridging device* dikirim kepada semua klien yang terhubung pada *web* server. Semua proses tersebut akan berjalan jika klien yang terhubung pada *bridging device* memberikan perintah untuk melakukan kontrol atau pemantauan.

Labib, et al. menggunakan sensor DHT11 untuk mendapatkan data suhu dan kelembapan udara, sedangkan penerapan protokol *WebSocket* menggunakan library Node.js dan Express.js. Mikrokomputer yang digunakan untuk menjalankan fungsi pengambilan data suhu dan kelembapan udara dari sensor DHT11 adalah mikrokomputer Arduino Nano, sedangkan mikrokomputer yang digunakan untuk *bridging device* adalah Raspberry Pi 3 (Labib, et al., 2018). Berdasarkan penelitian yang dilakukan oleh Labib, Adhitya, & Reza, penelitian ini akan menggunakan sensor DHT11 dan mikrokomputer Raspberry Pi 3. Sensor DHT11 akan digunakan pada penelitian ini untuk mendapatkan data suhu dan kelembapan udara, sedangkan mikrokomputer Raspberry Pi 3 digunakan untuk menjalankan fungsi pengambilan data dari sensor dan sebagai *Producer* dalam menerapkan protokol *AMQP*.

**Tabel 2.1 Kajian Pustaka**

| No | Nama Penulis, Tahun dan Judul | Persamaan | Perbedaan | |
|---|---|---|---|---|
| | | | Penelitian Terdahulu | Rencana Penelitian |
| 1 | *E. Luzuriaga, et al. [2015] A comparative evaluation of AMQP and MQTT protocols over* | Penggunaan RabbitMQ sebagai *message Broker* pada sistem. | Menganalisis perbandingan performa antara dua protokol yaitu, *AMQP* dan *MQTT* yang diimplementasikan pada kondisi jaringan | Melakukan implementasi protokol *AMQP* pada sistem yang berada dalam jaringan yang lebih stabil dan |

| | | | | |
|---|---|---|---|---|
| | *unstable and mobile networks* | | yang tidak stabil dan jaringan *mobile*. | menggunakan sebuah *wifi*. |
| 2 | *B. Babovic, Protic, & Milutinovic [2016] Web Performance Evaluation for Internet of Things Applications* | Data sensor ditampilkan melalui sebuah halaman *web* yang diakses oleh klien. | Melakukan implementasi protokol *AMQP* dengan menggunakan bahasa pemrograman Java dan JavaScript. Penggunaan Apache Qpid sebagai *message Broker*, Rabbit Java Client sebagai library pada sisi klien dan *Kaazing* JavaScript *AMQP client* yang digunakan agar aplikasi halaman *web* dapat berkomunikasi dengan *message Broker*. | Melakukan implementasi protokol *AMQP* dengan menggunakan bahasa pemrograman Python. Menggunakan RabbitMQ sebagai message *Broker* pada sistem. Menggunakan Pika Python *client* sebagai library pada sisi *Producer* dan *Consumer* yang digunakan untuk berkomunikasi dengan *message Broker*. Menggunakan Tornado yang digunakan agar aplikasi halaman *web* dapat menampilkan data sensor pada halaman *web*. |
| 3 | *Labib, Adhitya, & Reza [2018]* | Penggunaan protokol *WebSocket*, DHT11 untuk mengambil | Melakukan implementasi protokol *WebSocket* dan modul | Melakukan implementasi protokol *WebSocket* dengan |

| | | | | |
|---|---|---|---|---|
| | *Implementasi Protokol Websocket Pada Perangkat Non IP Berbasis NRF24L01 (Studi Kasus : Sistem* Pemantauan *Suhu dan Kontroling Lampu LED)* | data suhu dan kelembapan udara dan mikrokomputer Raspberry Pi 3. | NRF24L01 pada sistem pemantauan suhu dan mengatur *LED*. Node.js, socket.io dan express.js digunakan untuk melakukan implementasi dari protokol *WebSocket*. Implementasi sensor DHT11 dilakukan pada mikrokomputer Arduino Nano. | menggunakan Tornado. Implementasi sensor DHT11 dilakukan pada mikrokomputer Raspberry Pi 3 dan menggunakan library Adafruit_DHT untuk mengambil data dari sensor. |

Seperti yang sudah dijelaskan pada bab sebelumnya dan sesuai dengan penelitian terkait yang telah dijelaskan, penelitian ini akan membangun sebuah aplikasi pemantauan suhu dan kelembapan udara dengan memanfaatkan infrastruktur *IoT* yang menggunakan protokol *AMQP* dan *WebSocket* sebagai protokol komunikasi data. Dalam penelitian ini, protokol *AMQP* akan digunakan sebagai protokol komunikasi data dari komponen yang memberikan data suhu dan kelembapan udara hingga diterima oleh protokol *WebSocket*. Terdapat tiga komponen utama dalam penerapan protokol *AMQP* yaitu, *Producer* yang berperan sebagai pemberi informasi atau data, *Broker* yang berperan sebagai penerima data dari *Producer* dan meneruskan data tersebut kepada *Consumer* yang melakukan fungsi untuk mengambil data dari *Broker* dan *Consumer* yang berperan sebagai pengambil data dari *Broker*.

Komponen *Producer* terdiri sebuah mikrokomputer dan sensor yang berfungsi untuk mengambil data suhu dan kelembapan udara menggunakan library Adafruit_DHT, yang selanjutnya data tersebut dikirim menuju *Broker* dengan menggunakan library Pika. Komponen *Broker* terdiri dari sebuah mikrokomputer dan menggunakan perangkat lunak *message broker* RabbitMQ yang berperan sebagai *Broker* pada penerapan protokol *AMQP*, semua data yang didapatkan dari *Producer* akan disimpan secara sementara pada *queue* yang telah dibuat di dalam *Broker*. Sedangkan pada komponen *Consumer*, proses pengambilan data dari *Broker* dilakukan dengan menggunakan library Pika dan data tersebut akan digunakan oleh *WebSocket*.

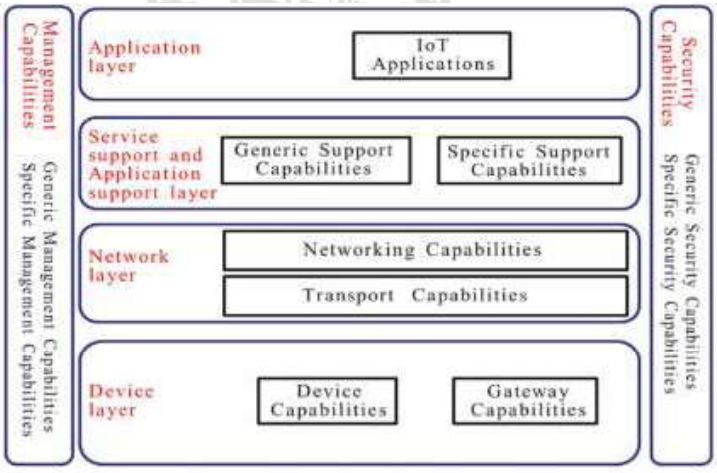
Protokol *WebSocket* digunakan untuk meneruskan data dari *Consumer* hingga diterima oleh aplikasi berbasis *web* yang akan digunakan oleh pengguna. Pembangunan server *WebSocket* dilakukan pada perangkat keras yang sama dengan *Consumer*, server dibangun menggunakan library Tornado. Dengan

menginisialisasi *port*, pengguna dapat mengakses *IP* dari server dan *port* yang akan digunakan melalui *web browser* yang mendukung protokol *WebSocket*. Setelah pengguna terhubung dengan server, server akan mengirimkan aplikasi halaman *web* dan data yang didapatkan dari *Consumer*. Data tersebut akan ditampilkan pada aplikasi halaman *web*.

## 2.2 Internet of Things (IoT)

Teknologi berbasis internet atau yang sering disebut *Internet of Things (IoT)*, merupakan sebuah konsep infrastruktur jaringan global atau lokal yang memanfaatkan konektivitas internet. Dengan kemampuan seperti berbagi data, kontrol jarak jauh dan sebagainya (Anon., 2013). Menurut Guoqiang et. al. (2013), *Internet of Things* (*IoT*) merujuk pada suatu jaringan yang menghubungkan perangkat fisik di berbagai jaringan menggunakan berbagai protokol berbeda. *IoT* bertujuan untuk memperluas manfaat dari konektivitas internet dengan menjadikan benda-benda disekitar kita dapat terhubung ke internet, sehingga dapat dikontrol dan diakses dari jarak jauh. *IoT* membuat perangkat di dunia fisik dapat diidentifikasi secara unik, mengumpulkan dan mengirimkan data, serta mengakses suatu layanan yang relevan melalui internet. *IoT* dapat diaplikasikan kedalam berbagai aktivitas disegala bidang sehingga pertukaran informasi melalui internet menjadi lebih mudah dan efisien (Guoqiang, et al., 2013).

Penelitian terbaru yang dilakukan pada bidang *Internet of Things (IoT)* menciptakan berbagai teknologi untuk mengumpulkan observasi *real-world* dengan menghubungkan sensor, aktuator, *Radio-Frequency Identification (RFID)* dan telepon seluler di *Web*. Teknologi ini memungkinkan pengembangan aplikasi secara luas seperti lingkungan cerdas yang mencakup dari rumah cerdas, perkotaan, perkantoran, lingkungan industri, transportasi dan logistic, kesehatan, pemantauan lingkungan dan lain-lain (B. Babovic, et al., 2016). *Internet of Things* terdiri dari 4 buah *layer* yaitu, *application layer*, *service support* dan *application support layer*, *network layer*, *device layer*. Gambar 2.1 merupakan model referensi *IoT*. (Anon., 2013)




Sumber: Rec. ITU-T Y.2060 (2013)

**Gambar 2.1 Model Referensi *IoT***

- ***Application Layer***
  *Application Layer* berisi aplikasi dari *IoT*.
- ***Service Support* dan *application support Layer***
  Terdiri dari pengelompokan kemampuan berikut,
  1. *Generic support capabilities* merupakan kemampuan umum yang dapat digunakan oleh berbagai aplikasi *IoT*, seperti pemrosesan data atau penyimpanan data. Kemampuan ini dapat juga dipanggil oleh *specific support capabilities*, misalnya untuk membangun *specific support capabilities* lainnya.
  2. *Specific support capabilities* merupakan kemampuan khusus yang memenuhi persyaratan dari aplikasi yang sudah golongkan. *Specific support capabilities* dapat terdiri dari berbagai macam kelompok kemampuan khusus, untuk menyediakan fungsi pendukung untuk aplikasi *IoT* yang berbeda.
- ***Network Layer***
  Terdiri dari 2 jenis kemampuan,
  1. *Networking capabilities*, menyediakan fungsi control yang relevan dari konektivitas jaringan, seperti akses dan fungsi kontrol sumber daya transportasi dan *AAA* (*authentication, authorization, and accounting*).
  2. *Transport capabilities*, fokus pada penyediaan konektivitas untuk transportasi layanan *IoT* dan informasi data spesifik dari aplikasi, serta kontrol dan manajemen transportasi yang terkait dengan *IoT*.
- ***Device Layer***
  Secara logis dapat dikategorikan kedalam dua jenis kemampuan,
  1. *Device capabilities*, berikut yang termasuk pada kemampuan perangkat namun tidak hanya terbatas pada hal dibawah ini,
     Interaksi langsung dengan jaringan komunikasi: Perangkat dapat mengumpulkan dan mengunggah informasi secara langsung (yaitu, tanpa menggunakan *gateway capabilities*) ke jaringan komunikasi dan dapat langsung menerima informasi (misalnya, perintah) dari jaringan komunikasi.
     Interaksi tidak langsung dengan jaringan komunikasi: Perangkat dapat mengumpulkan dan mengunggah informasi ke jaringan komunikasi secara tidak langsung, yaitu, melalui *gateway capabilities*. Di sisi lain, perangkat dapat secara tidak langsung menerima informasi (misalnya, perintah) dari jaringan komunikasi.
     Jaringan *ad-hoc*: Perangkat mungkin dapat membangun jaringan secara *ad-hoc* dalam beberapa skenario yang memerlukan peningkatan skalabilitas dan penyebaran cepat.
     Tidur dan bangun: Kemampuan perangkat dapat mendukung mekanisme "tidur" dan "bangun" untuk menghemat energi.
  2. *Gateway capabilities*, berikut yang termasuk pada kemampuan *gateway* namun tidak hanya terbatas pada hal dibawah ini,
     Dukungan banyak antarmuka: Pada *device layer*, kemampuan *gateway* mendukung perangkat-perangkat yang terhubung melalui berbagai jenis teknologi kabel atau nirkabel, seperti *controller area nerwork* (*CAN*) *bus*,

*ZigBee*, *Bluetooth* atau *Wi-Fi*. Pada *network layer*, kemampuan *gateway* dapat berkomunikasi melalui berbagai teknologi, seperti jaringan telepon umum (*PSTN*), jaringan generasi kedua atau generasi ketiga (*2G* atau *3G*), *long-term evolution network* (*LTE*), *Ethernet* atau *digital Subscriber line* (*DSL*).

Konversi protokol: Ada dua situasi dimana kemampuan *gateway* diperlukan. Satu situasi adalah ketika komunikasi pada *device layer* menggunakan protokol *device layer* yang berbeda, misalnya, protokol teknologi *ZigBee* dan protokol teknologi *Bluetooth*, yang lainnya adalah ketika komunikasi yang melibatkan *device layer* dan *network layer* menggunakan protokol yang berbeda misalnya, protokol teknologi *ZigBee* pada *device layer* dan protokol teknologi *3G* pada *network layer*.

- ***Management Capabilities***

  Kemampuan manajemen umum yang penting dalam *IoT* meliputi,

  1. Manajemen perangkat, seperti aktivasi perangkat jarak jauh dan de-aktivasi, diagnostik, pembaruan *firmware* dan / atau perangkat lunak, manajemen status kerja perangkat.
  2. Manajemen topologi jaringan lokal.
  3. Manajemen lalu lintas dan kemacetan, seperti deteksi kondisi *overflow* pada jaringan dan implementasi reservasi sumber daya untuk arus data.

- ***Security Capabilities***

  Terdapat 2 jenis kemampuan keamanan: kemampuan keamanan umum dan kemampuan keamanan khusus.

  1. Kemampuan keamanan umum,

     Pada *application layer*: otorisasi, autentikasi, kerahasiaan data aplikasi dan perlindungan integritas, perlindungan privasi, audit keamanan dan *anti-virus*.

     Pada *network layer*: otorisasi, autentikasi, penggunaan data dan sinyal kerahasiaan data dan memberi sinyal perlindungan integritas.

     Pada *device layer*: autentikasi, otorisasi, validasi integritas perangkat, kontrol akses, kerahasiaan data dan perlindungan integritas.
  2. Kemampuan keamanan khusus,

     Kemampuan keamanan khusus sangat berkaitan dengan persyaratan khusus aplikasi, misalnya, pembayaran seluler, persyaratan keamanan.

Penelitian ini akan membangun sebuah arsitektur *Internet of Things* dimana menggunakan sebuah sensor yang dapat mengambil data suhu dan kelembapan udara dari lingkungan penelitian. Sensor yang digunakan akan terhubung langsung secara *wired* kepada sebuah Raspberry Pi 3 yang memberikan perintah kepada sensor untuk melakukan pengambilan data, selain itu Raspberry Pi 3 juga mengolah data yang didapat dari sensor sehingga dapat dikirim pada *end-user*. Data tersebut akan dikirim dengan menggunakan protokol *AMQP* dan *WebSocket*, semua alat yang dibutuhkan untuk membangun arsitektur *Internet of Things* yang menerapkan protokol *AMQP* dan *WebSocket* akan terhubung pada sebuah jaringan nirkabel yang sama, begitu pula dengan *end-user*.

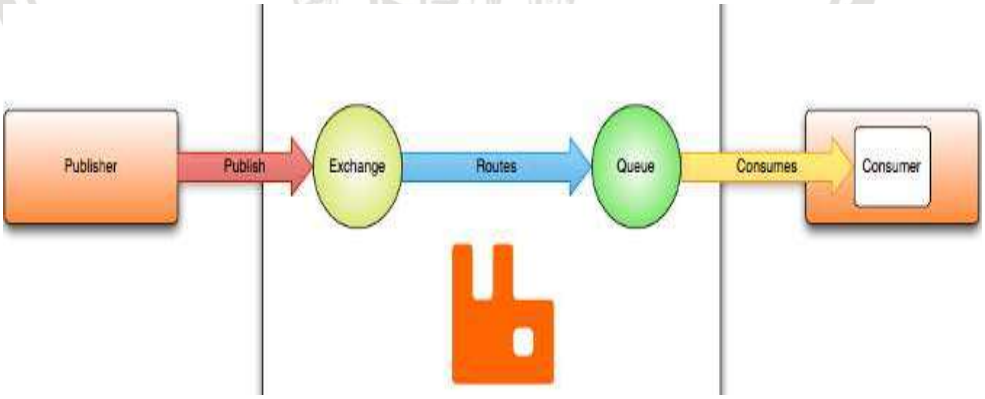
## 2.3 Advanced Message Queueing Protocol (AMQP)

*AMQP* adalah standar terbuka yang dirancang untuk mendukung perpesanan yang handal dan berkinerja tinggi melalui Internet. Protokol ini digunakan dalam pesan klien/server dan manajemen perangkat *IoT*. *AMQP* memungkinkan pengiriman pesan yang terenkripsi dan interoperabilitas (dapat dijalankan pada berbagai macam sistem operasi) antara organisasi dan aplikasi. *AMQP* memiliki keunggulan yaitu efisien, portabel, *multichannel* dan aman. Protokol biner menawarkan autentikasi dan enkripsi dengan cara *SASL* atau *TLS*, bergantung pada protokol *transport* seperti *TCP*. Protokol perpesanan yang cepat dan fitur pengiriman yang terjamin dengan *acknowledgement* saat pesan diterima. *AMQP* bekerja dengan baik di lingkungan multi-klien dan menyediakan sarana untuk mendelegasikan tugas dan membuat server menangani permintaan dengan lebih cepat. Karena *AMQP* adalah sistem perpesanan biner, interoperabilitas klien dari *vendor* yang berbeda terjamin (Rouse, 2018).

*AMQP* memungkinkan berbagai mode pengiriman pesan yang ditentukan untuk mengirim pesan, yaitu:

- *At-most-once* (dikirim sekali dengan kemungkinan data akan hilang atau terlewatkan)
- *At-least-once* (data pasti terkirim dengan kemungkinan terdapat adanya duplikasi data)
- *Exactly-once* (menjamin satu kali pengiriman saja) (Rouse, 2018)

*AMQP* sebagai protokol menyediakan *synchronous method* dan *asynchronous* yang dapat dipanggil oleh klien (Pope, 2012). Setiap bit data sudah ditentukan sebelum dikirimkan. Dengan karakteristik tersebut, library dapat dituliskan dalam berbagai macam bahasa dan dijalankan dalam berbagai sistem operasi dan arsitektur *CPU* (Anon., 2010). Terdapat tiga komponen utama pada arsitektur *AMQP* yaitu, *Producer* yang berperan sebagai pengirim data, *Consumers* yang berperan sebagai penerima data dan *Brokers* yang berperan sebagai jembatan antara *Producer* dan *Consumers* dalam komunikasi data (Marsh, 2009). Cara kerja *AMQP* seperti pada Gambar 2.2.




Sumber: rabbitmq (2007)

**Gambar 2.2 Cara Kerja AMQP**

*Broker AMQP* dengan versi 0-9-1 menyediakan beberapa tipe *exchange* yaitu,

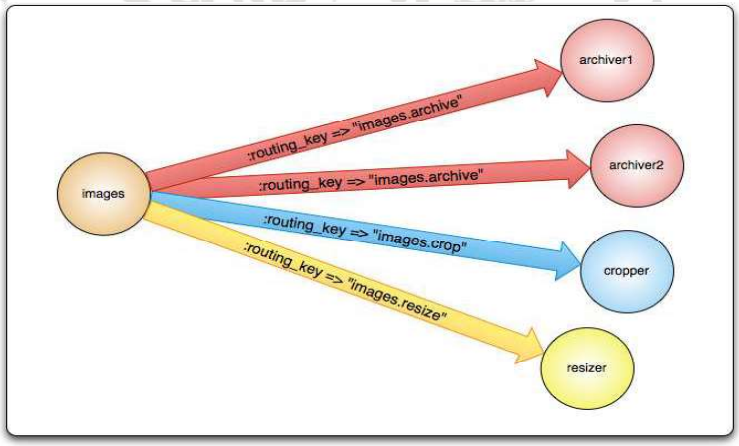
a. *Default Exchange*

*Default exchange* merupakan sebuah *direct exchange* yang tidak memiliki sebuah nama yang telah dideklarasikan oleh *Broker*. *Default exchange* tidak memiliki atribut khusus, sehingga menjadikan *default exchange* sangat cocok dalam penerapan *amqp* yang sederhana. Setiap *queue* dibuat, *default exchange* akan mengikat *queue* tersebut dengan sebuah *routing key* yang memiliki nama yang sama dengan *queue* tersebut.

Contoh, setiap kali *queue* dengan nama 'contoh' dibuat, maka *Broker* akan mengikat *queue* tersebut dengan *default exchange* dengan nama 'contoh' sebagai *routing key*. Maka, setiap kali sebuah pesan dimuat pada *default exchange* dengan menggunakan *routing key* 'contoh' akan diarahkan pada *queue* 'contoh'.

b. *Direct Exchange*

*Direct exchange* mengantar pesan pada sebuah *queue* berdasarkan *routing key* yang dimiliki oleh pesan tersebut. *Direct exchange* merupakan tipe *exchange* yang ideal untuk perpesanan dengan *unicast routing*, walaupun *direct exchange* dapat digunakan pada *multicast routing* juga. Misal, sebuah *queue* yang terikat pada *direct exchange* dengan menggunakan *routing key* K. Setiap terdapat sebuah pesan baru dengan *routing key* R sampai pada *direct exchange*, *direct exchange* akan mengarahkan pada *queue* tersebut jika K = R.




Sumber: rabbitmq (2007)

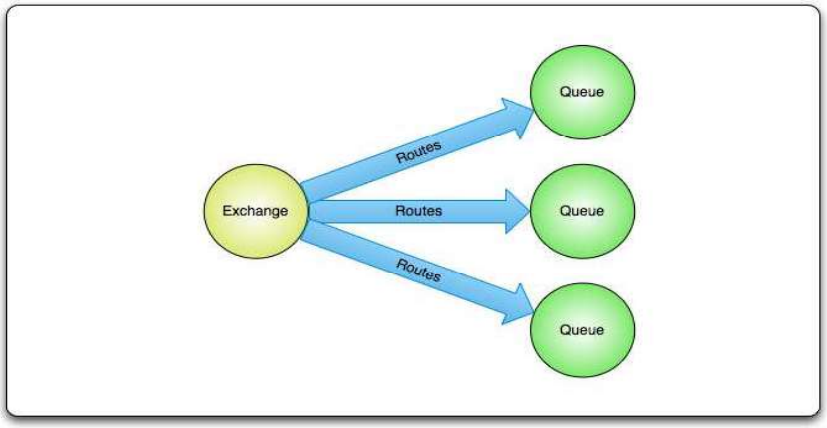
**Gambar 2.3 Cara Kerja Direct Exchange**

*Direct exchange* sering digunakan untuk mendistribusikan tugas antara beberapa fungsi dalam sebuah aplikasi dengan menggunakan *round robin*. Dalam *Broker amqp* versi 0-9-1, pesan-pesan yang masuk akan dilakukan *load balancing* untuk tiap-tiap *Consumer* bukan pada tiap-tiap *queue*.

c. *Fanout Exchange*

*Fanout exchange* mengarahkan pesan-pesan kepada semua *queue* yang terikat pada *fanout exchange* dan tidak membutuhkan sebuah *routing key*. Jika terdapat 10 *queue* yang terikat pada *fanout exchange*, setiap terdapat

sebuah pesan baru dimuat pada *fanout exchange*, salinan dari pesan tersebut akan dikirim kepada 10 *queue* yang terikat pada *fanout exchange*.




Sumber: rabbitmq (2007)

**Gambar 2.4 Cara Kerja Fanout Exchange**

d. *Topic Exchange*
   *Topic exchange* mengarahkan pesan-pesan pada satu atau banyak *queue* dengan menyamakan antara *routing key* yang dimiliki pesan dan pola yang digunakan untuk mengikat *queue* pada sebuah *exchange*. *Topic exchange* sering digunakan untuk mengimplementasi berbagai variasi pola *publish/subscribe*. *Topic exchange* umumnya digunakan untuk *multicast routing* pesan-pesan. Contoh penggunaan, pembaharuan bursa saham.
e. *Headers Exchange*
   *Headers exchange* dirancang untuk *routing* pada beberapa atribut yang lebih mudah disebut sebagai *header* pesan dibandingkan dengan *routing key*. *Header exchange* tidak memerlukan *routing key*. Sebagai gantinya, atribut yang digunakan untuk *routing* diambil dari atribut pada *headers*. Sebuah pesan dianggap sesuai apabila nilai dari *header* pesan tersebut sama dengan nilai yang sudah ditentukan saat melakukan pengikatan. (Anon., 2007)

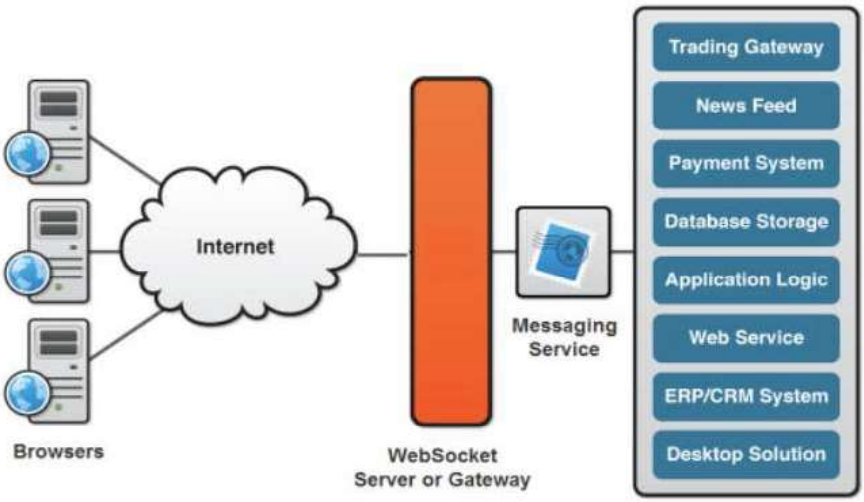
Penerapan protokol *AMQP* pada penelitian ini akan dilakukan pada tiga alat yaitu, pada dua buah Raspberry Pi 3 dan satu PC atau laptop. Terdapat tiga aktor yang akan diterapkan pada tiga buah alat yang telah disebutkan sebelumnya, yaitu *Producer*, *Broker* dan *Consumer*. Penerapan *Producer* dan *Consumer* akan dilakukan dengan menggunakan bahasa pemrograman Python. Penerapan pada Raspberry Pi 3 yang pertama akan berperan sebagai *Producer* untuk melakukan *publish* data kepada *Broker*, data tersebut didapatkan dari sensor yang terhubung pada Raspberry Pi 3 dengan menggunakan kabel. Penerapan *Producer* akan menggunakan library Pika yang menerapkan protokol *AMQP* 0-9-1.

Raspberry Pi 3 yang kedua akan digunakan sebagai *Broker* dengan melakukan pemasangan *message broker* RabbitMQ, setelah pemasangan RabbitMQ telah selesai akan dilakukan pengaktifan sebuah *plugin RabbitMQ_management*. *RabbitMQ_management* merupakan sebuah *API* berbasis *HTTP* yang digunakan

untuk melakukan pemantauan dan pengelolaan *Broker* RabbitMQ, *plugin* ini dapat diakses melalui *browser* dengan memasukkan *IP* dari *Broker* dengan *port* 15672. Pada penelitian ini, *plugin RabbitMQ_management* akan digunakan untuk membuat sebuah *queue* secara manual. Penerapan *Consumer* akan dilakukan pada sebuah PC atau laptop yang berperan sebagai penerima pesan dari *Broker*, *Consumer* diterapkan dengan menggunakan library yang sama dengan *Producer*. Proses komunikasi yang dimulai dari *Producer* mengirim data kepada *Broker* hingga *Consumer* menerima data tersebut dilakukan dengan menggunakan *port* 5672.

## 2.4 WebSocket

*WebSocket* adalah standar baru untuk komunikasi *realtime* pada *Web* dan aplikasi *mobile*. *WebSocket* dirancang untuk diterapkan di *browser web* dan server *web*, tetapi dapat digunakan oleh aplikasi klien atau server. *WebSocket* adalah protokol yang menyediakan saluran komunikasi *full-duplex* melalui koneksi *TCP* tunggal. *WebSocket* merupakan bagian dari *HTML5*. *WebSocket* menghadirkan pengurangan besar dalam lalu-lintas jaringan yang tidak penting dan *delay* dibandingkan dengan solusi *polling* dan *long-polling* yang telah digunakan untuk mensimulasikan koneksi dua arah dengan cara menjaga dua koneksi tetap terhubung (Darsiwan, 2016).




Sumber: websocket (2007)

**Gambar 2.5 Cara Kerja WebSocket**

*WebSocket* merupakan sebuah protokol komunikasi dua arah yang dapat digunakan oleh *browser*. Jika pada *AJAX* pengguna hanya dapat melakukan komunikasi satu arah dengan mengirimkan *request* kepada server dan menunggu balasannya, maka menggunakan *WebSocket* pengguna tidak hanya dapat mengirimkan *request* kepada server, tetapi juga menerima data dari server tanpa harus mengirimkan *request* terlebih dahulu. Hal ini berarti ketika menggunakan protokol *WebSocket* pengguna harus terus menerus terkoneksi dengan server. Model keamanan yang digunakan pada *WebSocket* adalah model keamanan

berbasis *origin-based* yang biasa digunakan oleh *web browsers* (Alex, 2015). Protokol ini terdiri dari *handshake* yang diikuti oleh *framing* pesan dasar, berlapis melalui *TCP*. Tujuan dari teknologi ini adalah untuk menyediakan sebuah mekanisme untuk aplikasi berbasis *browser* yang membutuhkan komunikasi dua arah dengan server yang tidak bergantung pada membuka beberapa koneksi *HTTP* (misalnya, menggunakan *XMLHttpRequest* atau *<iframe>* dan *polling* panjang) (Fette & Melnikov, 2011). Dibawah ini merupakan langkah-langkah yang dilakukan untuk membangun koneksi *WebSocket*,

1. Klien dari *WebSocket* mengirim sebuah permintaan *HTTP Upgrade*

   Terdapat dua skema *URI* pada protokol *WebSockets* yaitu, `ws-URI = "ws:" "//" host [ ":" port ] path [ "?" query ]` dan `wss-URI = "wss:" "//" host [ ":" port ] path [ "?" query ]`. Sedangkan *default port* untuk "ws" adalah 80 dan untuk "wss" adalah 443. Berikut merupakan contoh pesan *handshake* dari klien,

   ```
   GET /chat HTTP/1.1
   Host: server.example.com
   Upgrade: websocket
   Connection: Upgrade
   Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
   Origin: http://example.com
   Sec-WebSocket-Protocol: chat, superchat
   Sec-WebSocket-Version: 13
   ```

   *Upgrade*: *Upgrade* merupakan komponen penyusun *header* pada *HTTP1.1* yang mendefinisikan perubahan protokol. Jika server mendukung protokol yang diminta oleh klien, klien ingin beralih ke protokol *WebSocket*.

   *Connection*: *HTTP1.1* menetapkan bahwa *Upgrade* hanya bisa digunakan dalam *"direct connection"*, jadi pesan pesan *HTTP1.1* dengan *Upgrade header* harus mengandung *Connection header*. *Upgrade* menunjukkan bahwa permintaan yang dikirim oleh klien merupakan permintaan *upgrade* protokol.

   *Sec-WebSocket-Key*: merupakan sebuah rangkaian karakter acak yang berukuran 24-bit yang dihasilkan oleh klien untuk proses autentikasi saat melakukan *handshake*.

   *Sec-WebSocket-Version*: Angka 13 menunjukkan versi dari protokol *WebSocket* yang didukung oleh klien.

2. Server membaca pembukaan *handshake* dari klien dan server mengirim pembukaan *handshake*-nya.

   Setelah server menerima permintaan dari klien, server menentukan bahwa permintaan dari klien merupakan permintaan *WebSocket* berdasarkan informasi *header* dari permintaan tersebut. Lalu, server mengambil informasi dari komponen *Sec-WebSocket-Key*, membuat rangkaian karakter baru sesuai dengan algoritme dan menulis susunan karakter tersebut kedalam komponen *Sec-WebSocket-Accept* pada *header*. Dibawah ini merupakan respon server *WebSocket* standar,

   ```
   HTTP/1.1 101 Switching Protocols
   ```

```
Upgrade: websocket.
Connection: Upgrade.
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Nilai dari komponen *Sec-WebSocket-Accept* pada *header* dibangun dengan menggabungkan nilai komponen *Sec-WebSocket-Key* didapat dari *header* permintaan yang dikirim oleh klien, dengan *string* "258EAFA5-E914-47DA-95CAC5AB0DC85B11", hasil penggabungan dimasukkan pada fungsi *SHA-1 hash* untuk mendapatkan nilai 20-*byte* dan nilai 20-*byte hash* tersebut akan diubah dengan menggunakan skema *encoding Base64*. Informasi *header* lainnya menunjukkan bahwa server telah menerima permintaan untuk meningkatkan klien dari protokol *HTTP* menjadi protokol *WebSocket*.

3. Klien memvalidasi respon dari server

   Setelah menerima respon dari server, klien harus memvalidasi respon tersebut dan memverifikasi bahwa server telah menerima permintaan *upgrade* protokol dan mengembalikan pesan balasan dari *handshake*, lalu menghitung *Sec-WebSocket-Accept* pada klien sesuai dengan algoritme yang sama seperti algoritme yang digunakan oleh server untuk menghasilkan *Sec-WebSocket-Accept*. Kemudian hasilnya dibandingkan dengan konten *Sec-WebSocket-Accept* yang dikembalikan oleh server. Jika perbandingan informasi dinyatakan valid, koneksi dibuat. Jika tidak, pembentukan koneksi gagal. Jika koneksi antara klien dan server berhasil dibuat, maka klien dan server dapat berkomunikasi secara *full-duplex*, dimana klien dan server dapat bertukar data kapanpun hingga sisi klien atau server menutup koneksi. (Hu & Cheng, 2017)

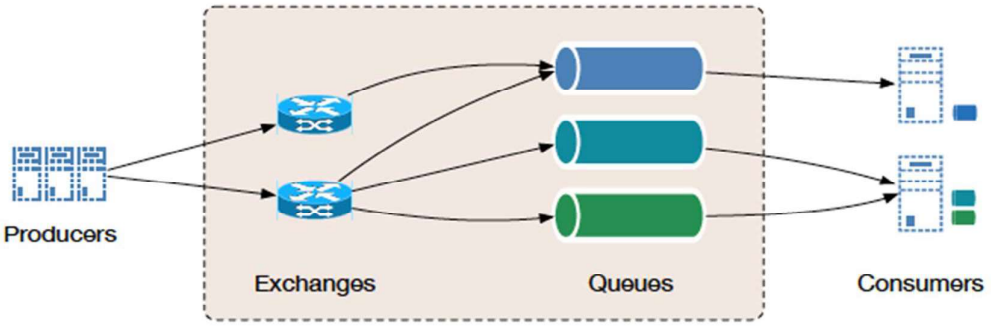
Penelitian ini akan menerapkan protokol *WebSocket* sebagai sarana komunikasi data, *WebSocket* akan diterapkan pada perangkat keras PC atau laptop. Perangkat keras yang digunakan untuk menerapkan protokol *WebSocket* sama dengan perangkat keras yang digunakan oleh *Consumer* pada penerapan protokol *AMQP* yang telah dijelaskan sebelumnya. Penerapan protokol *WebSocket* akan menggunakan bahasa pemrograman Python dan menggunakan library Tornado yang merupakan *web framework* berbasis Python, *port* yang akan digunakan adalah *port* 8888. *WebSocket* akan digunakan untuk meneruskan data yang didapatkan dari aplikasi *Consumer* kepada klien yang sudah terhubung dengan mengakses alamat *IP* dan *port* dari server *WebSocket* melalui sebuah halaman *web HTML*, yang akan berperan sebagai klien pada penelitian ini adalah sebuah *web browser* yang mendukung protokol *Websocket*.

## 2.5 RabbitMQ

RabbitMQ merupakan sebuah perangkat lunak yang ringan dan mudah untuk diterapkan pada perangkat keras dan pada sistem *cloud*. RabbitMQ dapat digunakan untuk komunikasi pesan dengan model *publish/subscribe*, pemrosesan secara asinkron, atau antrian kerja dan mendukung berbagai macam protokol perpesanan. RabbitMQ dapat diterapkan pada konfigurasi *distributed* dan *federated* sehingga dapat memenuhi skalabilitas yang tinggi dan ketersediaan

yang tinggi. RabbitMQ menawarkan berbagai fitur untuk memungkinkan pengguna menggantikan kinerja dengan keandalan, termasuk *persistence*, *delivery acknowledgements*, *Publisher confirmation* dan ketersediaan tinggi (Anon., 2007).

RabbitMQ adalah salah satu perangkat lunak *open-source* broker pesan yang mengimplementasikan *AMQP*. RabbitMQ adalah broker pesan yang mengambil pesan dan mengirimnya ke tempat lain dengan cara yang cukup cerdas. *AMQP* adalah protokol yang diterapkan RabbitMQ. RabbitMQ dapat menerima berbagai macam bahasa pemrograman, sehingga memudahkan pengguna dalam menulis program atau membaca program. Keuntungan besar lainnya adalah RabbitMQ berjalan di semua sistem operasi utama dan mendukung sejumlah besar platform pengembang seperti Java, *.NET*, Python, *PHP*, *Erlang* dan banyak lagi. Server RabbitMQ ditulis dalam bahasa pemrograman *Erlang* dan dibangun di atas *framework Open Telecom Platform (OTP)* untuk *clustering* dan *failover* (Singh, 2008).




Sumber: Pieter (2017)

**Gambar 2.6 Arsitektur RabbitMQ**

Komunikasi pada RabbitMQ dapat berupa *synchronous* atau *asynchronous* sesuai kebutuhan. *Publisher* mengirim pesan kepada *exchange* dan *Consumer* menerima pesan dari *queues*. Memisahkan *Producer* dari *queue* melalui *exchanges* memastikan bahwa *Producer* tidak terbebani dengan keputusan *routing*. RabbitMQ merupakan solusi untuk perpesanan, sering digunakan untuk memungkinkan *web* server untuk menanggapi *request* dengan cepat daripada dipaksa untuk melakukan prosedur yang membutuhkan banyak sumber daya selama pengguna menunggu hasilnya. RabbitMQ juga bagus untuk mendistribusikan sebuah pesan kepada beberapa penerima atau untuk menyeimbangkan beban antara pekerja dengan beban tinggi. Ketika kebutuhan melampaui *throughput*, RabbitMQ menawarkan beberapa fitur seperti, pengiriman yang dapat diandalkan, *routing*, federasi, *high availability*, keamanan, alat pengelolaan dan fitur lainnya (Pieter, 2017).

Pada penelitian ini, *message broker* RabbitMQ akan dipasang pada perangkat keras Raspberry Pi 3. RabbitMQ mengimplementasikan protokol *AMQP* 0-9-1, dimana terdapat lima tipe *exchange* yang telah dijelaskan sebelumnya, penelitian ini akan menggunakan tipe *default exchange* dimana tipe *exchange* ini sebuah *direct exchange* yang tidak memiliki sebuah nama yang telah dideklarasikan oleh *Broker*. *Default exchange* tidak memiliki atribut khusus, sehingga menjadikan

*default exchange* sangat cocok dalam penerapan *amqp* yang sederhana. Setiap *queue* dibuat, *default exchange* akan mengikat *queue* tersebut dengan sebuah *routing key* yang memiliki nama yang sama dengan *queue* tersebut.

Penelitian ini akan mengaktifkan sebuah *plugin* yang disediakan oleh RabbitMQ yaitu, *RabbitMQ_management*. *RabbitMQ_management* akan digunakan untuk memantau dan mengelola *Broker*, penelitian ini akan menggunakan *plugin* tersebut untuk membuat sebuah *queue* secara manual. Pada penerapan protokol *AMQP*, RabbitMQ akan berperan sebagai *Broker* yang menerima data berdasarkan tipe *exchange* dan memasukkan data tersebut kedalam *queue* yang terikat pada *exchange* tersebut. Selain itu, *Broker* juga berperan sebagai pengirim data kepada *Consumer* yang melakukan *consume* pada sebuah *queue*.

## 2.6 Tornado

Tornado adalah *web framework* Python. Dengan menggunakan jaringan *non-blocking I / O*, Tornado dapat menangani hingga puluhan ribu koneksi terbuka, sehingga ideal untuk *polling* yang panjang, *WebSockets* dan aplikasi lain yang memerlukan koneksi jangka panjang untuk setiap pengguna (Authors, 2009). Tornado hadir dengan dukungan *built-in* dan menemukan solusi untuk sebagian besar aspek dari pengembangan *Web* seperti *template*, lokalisasi, *cookie* yang ditandatangani dan lain-lain. Tornado juga memungkinkan pengguna untuk mencampurnya dengan *frameworks* lainnya, dengan cuplikan yang cocok, sesuai dengan kebutuhan pengguna.

Tornado menawarkan layanan *real-time* dan mendukung sejumlah besar koneksi bersamaan, *HTTP streaming* (protokol komunikasi yang diterapkan oleh *Apple Inc*) dan *polling* panjang (teknologi *push*, yang memungkinkan mekanisme *push emulatif* dalam keadaan di mana *push* nyata tidak mungkin). Dengan Tornado, sangat mudah untuk menulis layanan *real-time*. Tornado sangat cepat dibandingkan dengan semua *Web frameworks* berbasis Python lainnya. Tornado berjalan disebagian besar *platform* seperti *UNIX*, *Windows* dan seterusnya. Untuk kinerja dan skalabilitas yang lebih baik, hanya *Linux* dengan *EPoll* (sistem *kernel Linux* yang menggantikan *call* lama untuk mencapai kinerja yang lebih baik) dan *BSD* (*Berkeley Standard Distribution*) dengan *KQueue* (antarmuka pemberitahuan kejadian) direkomendasikan untuk penyebaran produksi yang efektif. *Windows* secara resmi tidak didukung tetapi direkomendasikan hanya untuk tujuan pengembangan (S, 2014).

Fitur *real-time web* membutuhkan koneksi yang berumur panjang dan sebagian besar dalam kondisi *standby*. Dalam *synchronous web* server tradisional, berarti memberikan satu *thread* untuk setiap pengguna yang ada, dimana biaya yang dibutuhkan sangat mahal. Untuk meminimalkan biaya dari koneksi secara bersamaan, Tornado menggunakan satu *thread* untuk tiap *event* secara berulang kali. Ini berarti bahwa semua kode aplikasi harus bertujuan menjadi *asynchronous* dan *non-blocking*, karena hanya satu operasi yang dapat aktif pada satu waktu (Authors, 2009).

Dengan *blocking*, ketika klien membuat permintaan untuk melakukan koneksi pada server, *threads* yang menangani koneksi tersebut diblokir sampai ada

beberapa data untuk dibaca, atau data sepenuhnya ditulis. Sampai operasi yang relevan selesai, *thread* tersebut tidak dapat melakukan apapun selain menunggu (Rukshani, 2017). Fungsi *asynchronous* kembali sebelum selesai dan umumnya menyebabkan beberapa pekerjaan berjalan pada *background* sebelum memicu beberapa tindakan selanjutnya di dalam sebuah aplikasi (Authors, 2009). Terdapat beberapa *class web framework* di dalam Tornado, salah satunya adalah Tornado.*websocket* yang merupakan implementasi dari protokol *WebSocket*.

*Class* yang ada pada Tornado yaitu, *class listen()* yang akan digunakan untuk menerima koneksi dan menerima permintaan dari klien melalui *port* yang sudah ditentukan, pada penelitian ini *port* yang akan digunakan adalah 8888. Lalu *class open()* yang digunakan untuk membuka layanan *WebSocket* saat terdapat permintaan dari klien untuk menggunakan protokol *WebSocket*. *Class write_message()* akan digunakan untuk mengirim pesan yang didapatkan dari *Consumer* kepada semua klien yang telah terhubung pada server *WebSocket*, pada penelitian ini pesan yang akan dikirimkan berupa *string*. Dan *class close()* yang akan digunakan untuk mengakhiri layanan dari server *WebSocket*.

## 2.7 Raspberry Pi 3

Raspberry Pi 3 adalah komputer berukuran kartu kredit yang dirancang dan diproduksi oleh Raspberry Pi 3 *Foundation*, sebuah organisasi nirlaba yang didedikasikan untuk membuat komputer dan instruksi pemrograman yang dapat diakses semaksimal mungkin untuk jumlah orang yang paling banyak (Fitzpatrick, 2016). Raspberry Pi 3 merupakan sebuah modul komputer mikro yang memiliki *input* dan *output digital port* seperti pada *board* mikrokontroler. Raspberry Pi 3 memiliki *port* yang dapat menghubungkan sebuah TV atau layar komputer dan juga memiliki *port USB* yang merupakan kelebihan dari Raspberry Pi 3 dibanding *board* mikrokontroler lainnya. Raspberry Pi 3 dibuat di inggris oleh *Raspberry Pi Foundation*. Pada awalnya Raspberry Pi 3 ditunjukan untuk modul pembelajaran ilmu komputer disekolah (Alfian, 2016). Berikut merupakan gambar dari Raspberry Pi 3.

Sumber: Fitzpatrick (2016)

**Gambar 2.7 Raspberry Pi 3**

Berikut merupakan spesifikasi yang dimiliki oleh Raspberry Pi 3,

- *1.2 Ghz ARM processor Systems-On-a-Chip (SoC) with integrated 1GB RAM.*

- *1 HDMI port for digital audio/video output*
- *1 3.5mm jack that offers both audio and composite video out (when paired with an appropriate cable).*
- *4 USB 2.0 ports for connecting input devices and peripheral add-ons.*
- *1 microSD card reader for loading the operating system.*
- *1 Ethernet LAN port.*
- *1 Integrated Wi-Fi/Bluetooth radio antenna.*
- *1 microUSB power port.*
- *1 GPIO (General Purpose Input/Output) interface.*

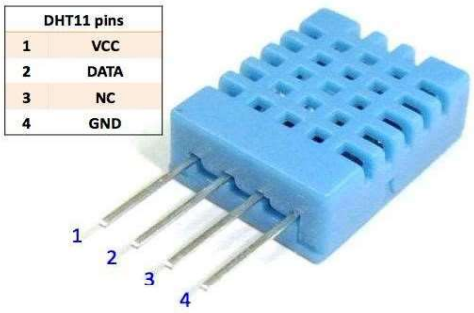
Raspberry Pi 3 dilengkapi dengan satu set 26 pin vertikal di papan. Pin ini adalah antarmuka *Input / Output* yang tidak terkait dengan fungsi asli tertentu pada papan Raspberry Pi 3. Sebaliknya, pin *GPIO* tersedia secara eksplisit bagi *end user* untuk memiliki akses langsung perangkat keras tingkat rendah ke papan untuk keperluan memasang perangkat keras papan lainnya, *peripherals*, layar tampilan *LCD* dan perangkat-perangkat keras lainnya ke Raspberry Pi 3 (Fitzpatrick, 2016).

Penelitian ini akan menggunakan perangkat keras Raspberry Pi 3 sebagai wadah untuk melakukan implementasi protokol *AMQP*, terdapat dua Raspberry Pi 3 yang akan digunakan pada penelitian ini. Pada Raspberry Pi 3 yang pertama akan dilakukan pemasangan sistem operasi *Raspbian Jessie*, library Adafruit_DHT yang akan digunakan oleh aplikasi *Producer* untuk mengambil data dari sensor DHT11 yang telah dihubungkan melalui kabel dan library Pika yang akan digunakan oleh aplikasi *Producer* untuk menerapkan protokol *AMQP*. Pada penerapan protokol *AMQP*, Raspberry Pi 3 ini akan berperan sebagai *Producer* yang mengolah dan mengirimkan data sensor kepada *Broker* melalui *port* 5672.

Sedangkan Raspberry Pi 3 yang kedua akan dilakukan pemasangan *message broker* RabbitMQ yang akan digunakan untuk menerapkan protokol *AMQP*. Pada penerapan protokol *AMQP*, Raspberry Pi 3 ini akan berperan sebagai *Broker* yang menerima data dari *Producer* dan mengirimkan data tersebut kepada *Consumer* melalui *port* 5672. *Broker* yang telah dipasang akan langsung berjalan setelah Raspberry Pi 3 ini hidup dan beroperasi dengan benar. Kedua Raspberry Pi 3 ini akan terhubung melalui sebuah jaringan nirkabel dan untuk menjalankan aplikasi *Producer* akan digunakan aplikasi *PuTTY* setelah mengaktifkan *interface SSH* pada konfigurasi Raspberry Pi 3.

## 2.8 DHT11

DHT11 adalah salah satu sensor yang dapat mengukur dua parameter lingkungan sekaligus, yakni suhu dan kelembapan udara (*humidity*). Dalam sensor ini terdapat sebuah thermistor tipe *NTC (Negative Temperature Coefficient)* untuk mengukur suhu, sebuah sensor kelembapan tipe resistif dan sebuah mikrokontroler 8-bit yang mengolah kedua sensor tersebut dan mengirim hasilnya ke pin output dengan format *single-wire bi-directional* (kabel tunggal dua arah) (ajie, 2016). Berikut merupakan gambar dari DHT11.




Sumber: amazon (2010)

**Gambar 2.8 Sensor DHT11**

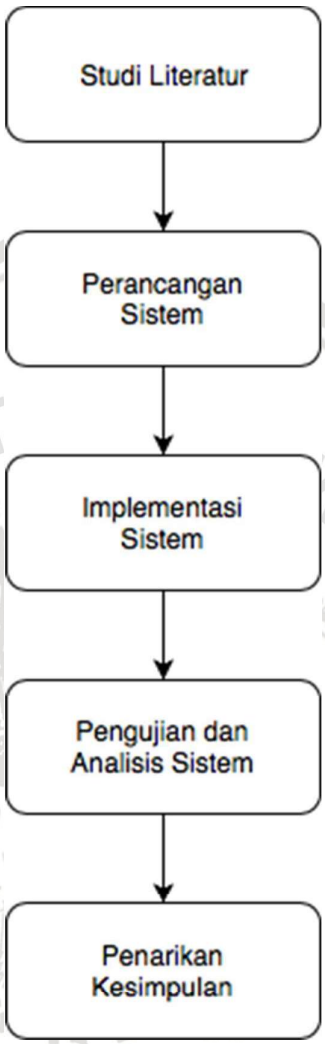
Berikut merupakan spesifikasi dari DHT11,

- *3 to 5V power and I/O*
- *2.5mA max current use during conversion (while requesting data)*
- *Good for 20-80% humidity readings with 5% accuracy*
- *Good for 0-50°C temperature readings ±2°C accuracy*
- *No more than 1 Hz sampling rate (once every second)*
- *Body size 15.5mm x 12mm x 5.5mm*
- *4 pins with 0,1" spacing* (lady, 2017)

Pada penelitian ini, sensor DHT11 akan digunakan sebagai sumber data untuk aplikasi *Producer* yang dalam penerapan protokol *AMQP*, aplikasi *Producer* berperan sebagai pemberi informasi atau data bagi *end-user*. Data yang akan diberikan oleh sensor berupa data suhu dan kelembapan udara yang telah diolah dengan menggunakan library Adafruit_DHT, pengambilan data sensor akan dilakukan setiap jeda yang ditentukan dalam detik dan akan dikirimkan melalui aplikasi *Producer* menuju *Broker*. Pada penelitian ini, sensor DHT11 akan dihubungkan kepada perangkat keras Raspberry Pi 3 dengan menggunakan kabel *jumper*. Kabel *jumper* tersebut akan menghubungkan bagian sensor *VCC*, *DATA* dan *GND* kepada pin yang memberi tegangan 3.3V, pin *GPIO* untuk memberikan input atau output dan pin yang berfungsi sebagai *ground* pada Raspberry Pi 3.

# BAB 3 METODOLOGI

Pada bagian ini akan menjelaskan tentang langkah-langkah yang akan dilakukan dalam penelitian. Berikut merupakan diagram alir yang menjelaskan tentang tahapan-tahapan metodologi penelitian.




**Gambar 3.1 Diagram Alir Metodologi Penelitian**

Gambar 3.1 menjelaskan metodologi penelitian yang akan dilaksanakan oleh peneliti. Langkah pertama adalah dengan mencari dasar teori atas penelitian baik melalui jurnal, buku maupun artikel yang berkaitan dengan penelitian. Dilanjutkan dengan menganalisis kebutuhan dalam penelitian, seperti kebutuhan apa saja yang diperlukan dalam membangun sistem pemantauan suhu dan kelembapan udara menggunakan protokol *AMQP* tersebut. Kemudian melakukan perancangan sebagai dasar dalam pengimplementasian sistem tersebut. Lalu dilakukan pengujian dan analisis untuk dilakukan penarikan kesimpulan dari penelitian.

## 3.1 Studi Literatur

Studi literatur berisikan dasar teori dalam melakukan penelitian. Jenis literatur yang dapat digunakan meliputi artikel, buku, jurnal, *e-book*, *website* dan laporan penelitian sebelumnya. Studi literatur yang digunakan dalam penelitian ini meliputi penelitian sebelumnya yang memiliki konsep sistem pemantauan suhu dan kelembapan ruangan, penggunaan protokol *AMQP* dalam infrastruktur *Internet of Things* serta melakukan pengujian dan analisis untuk menarik kesimpulan. Studi literatur dibutuhkan dalam penelitian, pengkajian dan penggunaan metode dari penelitian terkait yang telah dilakukan sebelumnya sehingga proses penelitian dapat lebih tepat dan minim kesalahan.

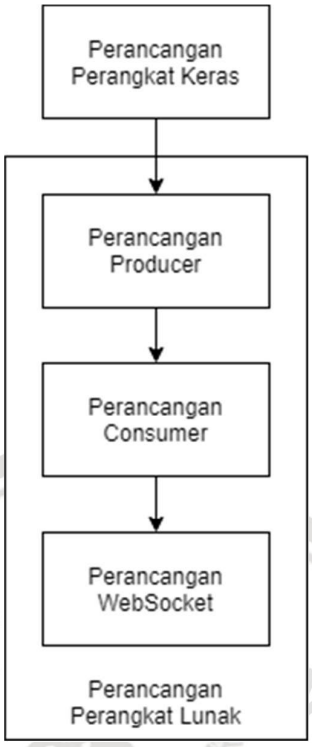
Terdapat beberapa studi literatur yang dibahas pada bab sebelumnya yaitu, kajian pustaka, *Internet of Things (IoT)*, *Advanced Message Queueing Protocol (AMQP)*, *WebSocket*, RabbitMQ, Tornado, Raspberry Pi 3 dan DHT11. Kajian pustaka dibahas mengenai penelitian-penelitian terkait dengan penelitian yang akan dilakukan lalu dianalisa persamaan dan perbedaannya. Definisi dari *Internet of Things (IoT)* dijelaskan pada bab sebelumnya. Selain itu, dijelaskan juga mengenai bagian-bagian yang dibutuhkan dan bagaimana penelitian ini akan membangun sistem pemantauan suhu dan kelembapan udara yang menerapkan infrastruktur dari *Internet of Things* menggunakan perangkat keras yang telah ditentukan.

Pada bab sebelumnya juga dijelaskan mengenai definisi dan cara kerja dari protokol *AMQP* yang akan digunakan sebagai protokol komunikasi data pada sistem pemantauan suhu dan kelembapan udara yang akan dibangun. Untuk menerapkan protokol *AMQP* dan *WebSocket* dibutuhkan beberapa perangkat lunak pendukung yaitu, RabbitMQ dan Tornado. Bagian-bagian dari RabbitMQ dan Tornado yang akan digunakan untuk menerapkan protokol *AMQP* dan *WebSocket*. Selain penjelasan mengenai bagian-bagian dari RabbitMQ dan Tornado, pada bab sebelumnya dijelaskan mengenai definisi dari perangkat lunak RabbitMQ dan Tornado.

Definisi dari perangkat keras Raspberry Pi 3 dan DHT11 yang akan digunakan pada penelitian ini dijelaskan pada bab sebelumnya. Bagian-bagian dari Raspberry Pi 3 dan DHT11 yang akan digunakan untuk membangun sistem pemantauan suhu dan kelembapan udara juga dijelaskan pada bab sebelumnya. Selain itu, pada bab sebelumnya juga dijelaskan mengenai peran dari tiap-tiap bagian untuk membangun sistem pemantauan suhu dan kelembapan udara sehingga dapat memberikan gambaran umum dari sistem yang akan dibangun.

## 3.2 Perancangan Sistem

Perancangan sistem memberikan gambaran mengenai alur kerja sistem berdasarkan kebutuhan yang telah ditentukan. Perancangan sistem dibuat sebagai pedoman untuk mempermudah tahap implementasi dan dapat dibagi ke dalam empat komponen berdasarkan peran dan fungsinya, yaitu perancangan perangkat keras, perangkat lunak *Producer*, perangkat lunak *Consumer* dan perangkat lunak *WebSocket*. Perangkat keras yang akan dirancang merupakan perangkat keras yang digunakan oleh *Producer*.




**Gambar 3.2 Tahap Perancangan Sistem**

Pada komponen *Producer* akan dirancang sehingga sensor DHT11 dan mikrokomputer Raspberry Pi 3 dapat terhubung, lalu mikrokomputer dapat mengambil data dari sensor tersebut. Sensor akan dihubungkan dengan mikrokomputer menggunakan beberapa kabel. Selain itu perancangan ini dilakukan supaya *Producer* dapat berjalan dengan baik. Selain perancangan perangkat keras, terdapat pula beberapa perancangan perangkat lunak yang terdiri dari *Producer*, *Consumer* dan *WebSocket*. Terdapat beberapa bagian pada perangkat lunak *Producer* yang akan dirancang yaitu, proses pengambilan data dari sensor DHT11, koneksi dengan *Broker* dan pengiriman data menuju *Broker*. Proses pengambilan data akan dijalankan pada mikrokomputer dengan menggunakan library. Proses pengiriman data atau *publish* menuju *Broker* dilakukan setelah *Producer* berhasil terhubung dengan *Broker* menggunakan parameter *IP* dari *Broker*, *port* yang digunakan oleh protokol *AMQP* 0-9-1, *username* dan *password* yang sudah terdaftar pada *Broker*. Sedangkan proses *publish* akan dilakukan menggunakan parameter *exchange* dan *routing key* yang telah ditentukan sebelumnya. Tahap perancangan ini dilakukan supaya *Producer* dapat mengambil dari sensor DHT11, mengirim data menuju *Broker* dan data tersebut dapat diambil oleh *Consumer*.

*Consumer* akan dirancang sehingga dapat melakukan pengambilan data atau *consume* pada sebuah *queue* di dalam *Broker*. Sebelum *Consumer* dapat mengambil data, *Consumer* akan melakukan proses koneksi dengan *Broker* menggunakan parameter *IP* dari *Broker*, *port* yang digunakan oleh protokol *AMQP* 0-9-1, *username* dan *password* yang sudah terdaftar pada *Broker*. Setelah proses koneksi berhasil dilakukan, *Consumer* akan mulai melakukan proses *consume*

dengan menggunakan parameter *queue* yang telah ditentukan sebelumnya. Data yang didapatkan melalui proses *consume* akan digunakan oleh perangkat lunak *WebSocket* untuk dikirim menuju pengguna.

Sebelum perangkat lunak *WebSocket* dapat mengirimkan data menuju pengguna, akan dilakukan penentuan *port* yang digunakan untuk melakukan layanan dan pengguna dapat mengakses server menggunakan *IP* dan *port* dari server. Setelah pengguna berhasil terhubung dengan server, perangkat lunak *WebSocket* akan menjalankan fungsi untuk mengirimkan aplikasi halaman *web* kepada setiap pengguna yang terhubung. Dan perangkat lunak *WebSocket* akan mengirimkan data yang didapat dari *Consumer* kepada pengguna, data tersebut akan ditampilkan pada aplikasi halaman *web*. Perancangan perangkat lunak *WebSocket* dilakukan supaya server dapat memberi layanan dan mengirim data yang didapat dari perangkat lunak *Consumer* kepada pengguna.

## 3.3 Implementasi Sistem

Implementasi sistem dibuat berdasarkan perancangan sistem yang telah dibuat sebelumnya. Implementasi perangkat keras *Producer* dimulai dengan memasang sensor DHT11 pada Raspberry Pi 3 yang dihubungkan dengan menggunakan kabel *jumper*. Beberapa bagian dari sensor yang harus terhubung pada Raspberry Pi 3 adalah *VCC* terhubung pada sumber tegangan listrik, *DATA* terhubung pada pin *input-output* dan *GND* terhubung pada pin *ground*. Selain itu, mikrokomputer Raspberry Pi 3 yang digunakan oleh *Producer* dan *Broker* harus terhubung pada sumber tegangan listrik.

Setelah proses implementasi perangkat keras *Producer* berhasil dilakukan, perangkat lunak *Producer* akan dibangun menggunakan library Adafruit_DHT dan Pika. Library Adafruit_DHT digunakan untuk proses pengambilan data suhu dan kelembapan udara dari sensor DHT11 dan library Pika digunakan untuk penerapan protokol *AMQP*. Dengan menggunakan library Pika, *Producer* dapat melakukan koneksi dengan *Broker* menggunakan parameter *IP* dari *Broker*, *port* yang digunakan oleh protokol *AMQP* 0-9-1, *username* dan *password* yang sudah terdaftar pada *Broker*. Library Pika juga digunakan oleh *Producer* dalam melakukan *publish* data menuju *Broker* dengan menggunakan parameter *exchange* dan *routing key*. Proses pengambilan data dari sensor DHT11 dan proses *publish* data menuju *Broker* dilakukan setiap 5 menit sekali.

Untuk melakukan pengambilan data dari *Broker*, perangkat lunak *Consumer* memerlukan library Pika yang telah terpasang. Proses pengambilan data atau *consume* dilakukan dengan menggunakan parameter *queue*. Namun sebelum perangkat lunak dapat melakukan proses *consume*, perangkat lunak *Consumer* harus terhubung dengan *Broker*. Proses koneksi dengan *Broker* dilakukan menggunakan parameter *IP* dari *Broker*, *port* yang digunakan oleh protokol *AMQP* 0-9-1, *username* dan *password* yang sudah terdaftar pada *Broker*. Implementasi ini bertujuan supaya *Consumer* dapat mengambil data dari *queue* yang ada pada *Broker*.

Implementasi *WebSocket* dimulai dengan melakukan instalasi library Tornado yang digunakan untuk pembangunan server *WebSocket*. Pembangunan server *WebSocket* dimulai dengan menentukan *port* yang akan digunakan oleh server.

Selain penentuan *port*, pembangunan server *WebSocket* membutuhkan beberapa *class* dan *method* yang ada pada library Tornado. *Class* dan *method* yang ada pada library Tornado akan digunakan untuk menangani klien yang membuka koneksi dan menutup koneksi pada server *WebSocket*, menangani *request* dan *response HTTP* (*response* yang diberikan yaitu melakukan *render* sebuah halaman *web* yang dikirimkan pada semua klien yang terhubung) dan mengirimkan data yang didapat dari *Consumer* kepada semua klien yang terhubung.

## 3.4 Pengujian dan Analisis Sistem

Dalam tahap pengujian, peneliti akan melakukan pengujian terhadap integritas dari protokol *AMQP* yang diterapkan pada sistem pemantauan suhu dan kelembapan udara. Tujuan dilakukan pengujian integritas sistem adalah untuk mengetahui apakah sistem berhasil memenuhi kebutuhan atau tidak.

Pelaksanaan pengujian menggunakan jaringan lokal dengan parameter integritas data, *delay* dan memori. Parameter *delay* dan memori digunakan untuk mengukur performa dari sistem. Skenario pengujian yang akan dilakukan pada penelitian ini dibagi menjadi tiga yaitu, pengujian untuk integritas data dilakukan dengan membandingkan data yang dikirimkan oleh *Producer* dengan data yang ditampilkan pada halaman *web*. Pengujian dilakukan dengan aplikasi *Producer* mulai mengambil data suhu dan kelembapan udara dari sensor lalu mengirim data tersebut menuju *Broker*, setiap proses pembacaan data sensor dan pengiriman data dilakukan dengan jeda 5 detik.

Dengan menggunakan parameter *delay*, pengujian keandalan sistem akan dilakukan ketika *Producer* mulai mengirim pesan hingga ditampilkan pada halaman *web* yang selanjutnya akan dihitung *average delay* dari tiap proses pengiriman pesan. Proses pengujian dilakukan dengan melakukan *capture time* disaat *Producer* mengirim data sensor kepada *Broker* hingga pada *Consumer*, disaat data sampai pada sisi *Consumer* akan dilakukan juga *capture time*. Dan disaat data tersebut ditampilkan di halaman *web*, akan dilakukan *capture time* juga. Kemudian dilakukan penghitungan nilai *delay* dengan cara mengurangi waktu saat data ditampilkan di halaman *web* dan waktu saat data sensor dikirim.

Parameter *delay* juga digunakan untuk pengujian keandalan sistem saat *Producer* mengirim pesan hingga ditampilkan pada halaman *web*. Proses pengujian dilakukan dengan melakukan *capture time* disaat *Producer* mengirim data sensor kepada *Broker* hingga data tersebut ditampilkan pada halaman *web*. Disaat data ditampilkan pada halaman *web* akan dilakukan *capture time*. Proses tersebut dilakukan hingga terdapat 25 data *delay (delay)* dengan berbagai macam variasi jeda.

Sedangkan parameter memori akan digunakan pada pengujian keandalan sistem dengan skenario dimana terdapat beberapa variasi jumlah *Producer* yang melakukan pengiriman data menuju *Broker* dan terdapat satu *Consumer* yang mengambil data tersebut dari *Broker*. Proses pengujian dilakukan dengan menggunakan perangkat lunak *psrecord* untuk mengambil data *log* penggunaan memori saat proses pengujian tersebut berjalan.

Setelah pengujian integritas sistem dan keandalan sistem berhasil dilakukan, akan dilakukan analisis berdasarkan hasil dari pengujian yang telah dilakukan.

Analisis integritas data dilakukan dengan membandingkan 50 data yang dikirim oleh *Producer* dengan data yang ditampilkan pada halaman *web*, dari 50 data tersebut akan dibandingkan apakah data dari *Producer* dan halaman *web* sudah sesuai dan berurutan. Selain itu, berdasarkan hasil pengujian keandalan sistem yang menggunakan parameter *delay* dan memori akan dilakukan analisis mengenai bagaimana performa dari sistem sehingga dapat ditentukan apakah protokol-protokol yang digunakan pada implementasi sistem sudah sesuai.

## 3.5 Penarikan Kesimpulan

Penarikan kesimpulan dapat dilakukan jika semua tahap sebelumnya telah selesai dibuat dan dilakukan. Kesimpulan dapat diambil melalui data hasil pengujian terhadap penerapan protokol *AMQP* pada sistem pemantauan suhu dan kelembapan udara yang telah dibangun. Selain itu penulisan saran juga diperlukan sebagai acuan dalam memberikan pertimbangan mengenai pengembangan dan penelitian lanjut mengenai penerapan protokol *AMQP* pada sistem pemantauan suhu dan kelembapan udara.

# BAB 4 PERANCANGAN SISTEM

Bab perancangan sistem menjelaskan secara detail proses perancangan sistem. Penjelasan dari perancangan sistem meliputi gambaran umum dari sistem, analisis kebutuhan sistem, alur kerja sistem, perancangan perangkat keras, perancangan perangkat lunak dan perancangan pengujian.

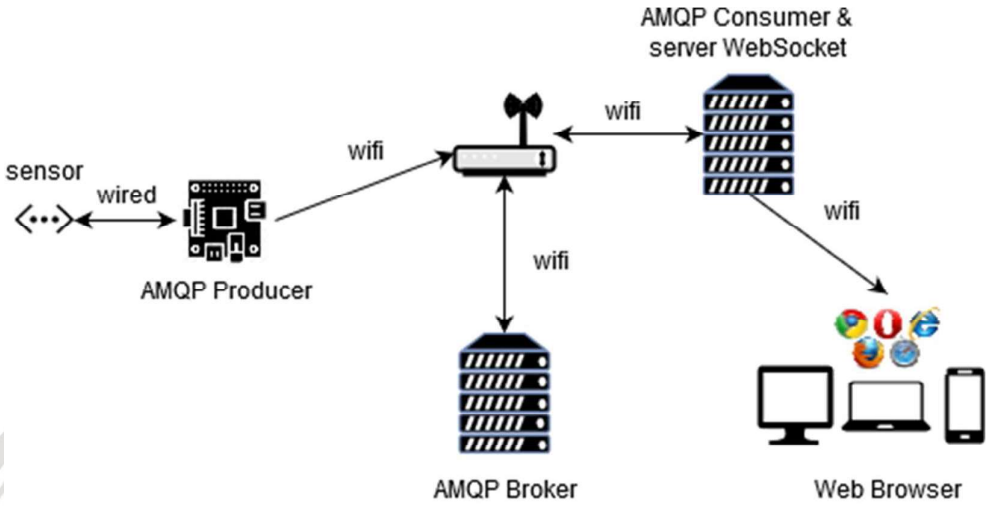
## 4.1 Gambaran Umum Sistem

Gambaran umum sistem menjelaskan peran dari setiap komponen dalam menjalankan sistem yang akan dibangun. Pada penelitian ini, pembangunan sistem pemantauan suhu dan kelembapan udara menggunakan protokol *AMQP* sebagai protokol komunikasi data antar tiap *node*. Penerapan protokol *AMQP* terbagi menjadi tiga komponen yaitu, *Broker*, *Producer* dan *Consumer*. Komponen *Broker* diterapkan pada sebuah mikrokomputer Raspberry Pi 3 dengan melakukan instalasi perangkat lunak RabbitMQ, *Broker* berperan sebagai jembatan bagi *Producer* dan *Consumer*. *Broker* akan menerima data yang dikirimkan oleh *Producer* melalui sebuah *default exchange*, setelah data diterima *Broker* akan menaruh data tersebut pada sebuah *queue* yang telah dibuat melalui aplikasi *RabbitMQ_management*. Lalu *Broker* akan mengirim data tersebut kepada *Consumer* yang melakukan *consume* pada *queue* tersebut.

Komponen *Producer* dibangun pada sebuah mikrokomputer Raspberry Pi 3 yang berperan sebagai pembaca data sensor dan mengirimkan data tersebut pada *Broker*. Perangkat lunak *Producer* menggunakan library Adafruit_DHT yang digunakan untuk membaca data suhu dan kelembapan udara dari sensor DHT11 yang telah dihubungkan dengan menggunakan kabel *jumper*. Data tersebut akan dikirimkan melalui protokol *AMQP* dengan menggunakan library Pika, aplikasi *Producer* mengirimkan data tersebut kepada *Broker* dengan menginisialisasi parameter *exchange* dan *routing key*. Proses pengiriman data dilakukan setelah proses koneksi pada *Broker* berhasil dilakukan dengan menggunakan *username* dan *password* yang sudah terdaftar pada *Broker*. Selain itu, proses koneksi juga dilakukan menggunakan parameter *IP* dari *Broker* dan *port* yang digunakan oleh protokol *AMQP* 0-9-1. Proses pembacaan data dari sensor dan *publish* dilakukan setiap 5 menit sekali.

Sedangkan pembangunan komponen *Consumer* dilakukan pada perangkat keras laptop. Perangkat lunak *Consumer* menggunakan library Pika untuk berkomunikasi dengan *Broker*, *Consumer* melakukan koneksi dengan *Broker* menggunakan parameter *username* dan *password* yang sudah terdaftar pada *Broker*. Selain itu, parameter *IP* dari *Broker* dan *port* yang digunakan oleh protokol *AMQP* 0-9-1 juga dibutuhkan untuk proses koneksi. Setelah proses koneksi berhasil dilakukan, *Consumer* akan memulai proses *consume* menggunakan parameter *queue* untuk mengambil data dari *queue* yang ada pada *Broker*. Penerapan protokol *WebSocket* dilakukan pada perangkat keras laptop yang sama dengan *Consumer*, komponen *WebSocket* berperan sebagai penyedia layanan yang dapat diakses melalui *web browser*. Pembangunan aplikasi *WebSocket* menggunakan library Tornado, aplikasi ini menjadi server yang dapat diakses oleh

klien *web browser* yang terhubung pada jaringan nirkabel yang sama dengan menggunakan *IP* dari server ini dan *port*. Saat server berhasil diakses halaman *web* akan dikirimkan kepada klien dan ditampilkan melalui *web browser*. Server akan mengirimkan data yang didapatkan dari *Consumer* kepada semua klien yang telah terhubung pada server, data tersebut akan ditampilkan pada halaman *web* yang telah ditampilkan pada *web browser* sebelumnya. Gambaran umum dari sistem dapat dilihat pada gambar 4.1.




**Gambar 4.1 Gambaran Umum Sistem**

## 4.2 Analisis Kebutuhan

Analisis Kebutuhan bertujuan untuk mengetahui hal-hal yang dibutuhkan dalam penelitian sehingga penelitian ini tepat sasaran. Terdapat 5 tahapan analisis kebutuhan yaitu, kebutuhan pengguna, kebutuhan fungsional, kebutuhan non-fungsional, kebutuhan perangkat keras dan kebutuhan perangkat lunak.

### 4.2.1 Kebutuhan Pengguna

Pada penelitian ini, pengguna membutuhkan sistem untuk mengambil data suhu dan kelembapan udara dari sensor DHT11 yang telah dihubungkan pada Raspberry Pi 3. Proses pengambilan data oleh sistem dilakukan menggunakan library Adafruit_Python_DHT. Informasi tersebut diolah sehingga menjadi pesan yang dikirimkan menuju *Broker* pada tipe *exchange* dan *queue* yang telah ditentukan. Setelah data diterima oleh *Broker*, data tersebut diteruskan kepada *Consumer* yang selanjutnya data tersebut diterima oleh server *WebSocket.* Setelah data diterima pada server *WebSocket*, data tersebut dikirim kepada semua klien yang terhubung pada server *WebSocket* dan ditampilkan melalui sebuah halaman *web*. Halaman *web* dapat diakses melalui *web browser* dengan memasukkan *IP* dan *port* yang telah ditentukan, halaman *web* dapat diakses melalui berbagai *device* yang memiliki *web browser* dan mendukung protokol *WebSocket* yang terhubung pada satu jaringan nirkabel yang sama dengan server *WebSocket*.

### 4.2.2 Kebutuhan Fungsional

Di dalam sistem yang akan dibangun, terdapat lima bagian dari kebutuhan fungsional yang diantaranya adalah fungsi komunikasi *Producer*, fungsi komunikasi *Broker*, fungsi komunikasi *Consumer*, fungsi komunikasi *WebSocket* dan fungsi komunikasi *web browser*. Setiap bagian yang telah disebutkan diperlukan supaya sistem yang akan dibangun dapat berjalan dengan baik. Dalam menjalankan sistem, *Producer* harus bisa dalam mengambil data suhu dan kelembapan udara dari sensor DHT11, melakukan koneksi dengan *Broker* menggunakan parameter *IP* dari *Broker*, *port* yang digunakan oleh protokol *AMQP*, *username* dan *password* yang sudah terdaftar pada *Broker* dan melakukan *publish* data suhu dan kelembapan udara menuju *Broker* dengan menggunakan parameter *exchange* dan *routing key* yang sudah ditentukan. Proses koneksi dan pengiriman data kepada *Broker* dilakukan dengan jeda pengiriman yang sudah ditentukan.

Sedangkan pada sisi fungsi komunikasi *Broker*, sistem mengharuskan *Broker* dapat menerima pesan yang dikirimkan oleh *Producer* sesuai dengan *exchange* dan *routing key* lalu memasukkan pesan tersebut kedalam *queue* yang sudah ditentukan oleh *Producer*. Proses penerimaan pesan dilakukan setelah *Broker* bisa menerima koneksi dari *Producer* berdasarkan *username* dan *password* yang digunakan oleh *Producer*. Selain itu, *Broker* juga harus bisa menerima koneksi dari *Consumer* berdasarkan *username* dan *password* yang digunakan oleh *Consumer*. Setelah *Broker* bisa menerima koneksi dari *Consumer*, sistem mengharuskan *Broker* dapat mengirim pesan yang ada di dalam *queue* kepada *Consumer* yang melakukan *consume* pada *queue* tersebut.

Sistem mengharuskan *Consumer* dapat melakukan koneksi dengan *Broker* menggunakan parameter *username*, *password*, *IP* yang digunakan oleh *Broker* dan *port* yang digunakan oleh protokol *AMQP*. Selain itu, sistem membutuhkan *Consumer* untuk dapat melakukan fungsi pengambilan pesan dari *queue* yang ada di dalam *Broker* menggunakan parameter *queue* yang sudah ditentukan. Setelah pesan tersebut berhasil diterima pada sisi *Consumer*, sistem mengharuskan fungsi komunikasi *WebSocket* untuk mengirim pesan tersebut kepada semua klien yang sudah terhubung. Di dalam sistem, proses pengiriman pesan oleh *WebSocket* dilakukan setelah klien dapat mengakses server, aplikasi halaman *web* berhasil dikirim kepada klien dan aplikasi halaman *web* dapat ditampilkan pada *web browser*. Klien dapat mengakses server setelah fungsi komunikasi *WebSocket* sudah melakukan layanan pada *port* yang telah ditentukan.

Lalu pada sisi *web browser*, sistem mengharuskan perangkat lunak *web browser* mendukung protokol *WebSocket* sehingga dapat mengakses server menggunakan *IP* dan *port* yang digunakan oleh server. *Web browser* juga harus mendukung *HTML5* sehingga aplikasi halaman *web* dapat ditampilkan dan mulai menerima data yang dikirim oleh server. Semua fungsi yang telah dijelaskan sangat dibutuhkan oleh sistem dalam mengirim data dari sensor DHT11(suhu dan kelembapan udara) hingga data tersebut diterima oleh pengguna melalui perangkat lunak *web browser*.

### 4.2.3 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan kebutuhan yang tidak terhubung secara langsung dengan layanan sistem dan membahas mengenai apakah data yang mengalir pada sistem memiliki integritas, dimulai dari saat data dikirimkan oleh *Producer* hingga diterima oleh *WebSocket* dan ditampilkan pada sebuah halaman *web*. Serta mengenai *delay* yang ada saat sistem berjalan, dimulai dari *Producer* mengirim pesan hingga halaman *web* menampilkan pesan tersebut. Dan juga penggunaan memori saat *Producer* mulai mengirimkan data hingga diterima oleh *Consumer*.

### 4.2.4 Kebutuhan Perangkat Keras

Perancangan dan implementasi dari sistem mengacu pada analisis kebutuhan perangkat keras yang bertujuan untuk mengetahui perangkat keras yang dibutuhkan oleh sistem agar berjalan dengan baik. Setiap komponen yang ada pada sistem memiliki fungsi dan kebutuhan perangkat keras masing-masing.

Pada komponen *Producer*, perangkat keras yang digunakan berupa Raspberry Pi 3 yang terhubung dengan sensor suhu dan kelembapan udara DHT11. Sensor DHT11 ini dihubungkan dengan Raspberry Pi 3 menggunakan kabel. Perangkat keras ini berfungsi sebagai pengambil data yang berupa nilai suhu dan kelembapan udara. Setelah data nilai sensor di ambil maka data nilai sensor tersebut akan dikirimkan ke *Broker*. *Broker* yang berada pada perangkat keras Raspberry Pi 3 ini akan memproses dan mengirimkannya ke *Consumer*. Pada sisi perangkat keras laptop terdapat tiga komponen sistem yaitu, *Consumer*, *WebSocket* dan juga *web browser*. Perangkat keras laptop ini berperan sebagai *Consumer* yang nantinya digunakan sebagai pengambil data dari *Broker*, mengirim data menuju *web browser* dan menampilkan data hasil pemantauan tersebut melalui halaman *web*. Berikut gambaran kebutuhan perangkat keras yang dibutuhkan oleh sistem.

**Tabel 4.1 Kebutuhan Perangkat Keras**

| Komponen | Perangkat Keras |
|---|---|
| *Producer* | DHT11 dan Raspberry Pi 3 |
| *Broker* | Raspberry Pi 3 |
| *Consumer* | Laptop |
| *WebSocket* | Laptop |
| Pengguna | *PC*, laptop, atau *smartphone* |

### 4.2.5 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak bertujuan untuk mengetahui perangkat lunak yang digunakan oleh sistem. Analisis kebutuhan perangkat lunak menjadi dasar perancangan dan implementasi sistem. Seperti pada kebutuhan perangkat keras, setiap komponen sistem tersebut juga memiliki kebutuhan perangkat lunak. Perangkat lunak ini berupa program, aplikasi, atau library.

Pada sisi *Producer* dan *Broker* yang berada pada Raspberry Pi 3, dilakukan instalasi Raspbian OS Jessie pada mikrokomputer Raspberry Pi 3. Setelah instalasi Raspbian OS Jessie berhasil, pada sisi *Producer* dilakukan instalasi library Adafruit_Python_DHT dan Pika. Library Adafruit_Python_DHT yang digunakan

untuk mengambil data suhu dan kelembapan udara dari sensor DHT11 dan library Pika yang digunakan untuk melakukan proses *publish* menuju *Broker* pada penerapan protokol *AMQP*. Pada sisi *Broker* dilakukan instalasi *message broker* RabbitMQ yang digunakan sebagai *Broker* pada penerapan protokol *AMQP*. RabbitMQ berperan sebagai penerima pesan dari *Producer* dan mengirimkan data kepada *Consumer*. Selain itu, pada sisi server *WebSocket* dan *Consumer* dilakukan instalasi library Tornado dan Pika. Library Tornado yang digunakan untuk membangun server *WebSocket* dan library Pika yang digunakan untuk melakukan proses *consume* pada penerapan protokol *AMQP*. Berikut merupakan gambaran dari perangkat lunak yang dibutuhkan oleh sistem.

**Tabel 4.2 Kebutuhan Perangkat Lunak**

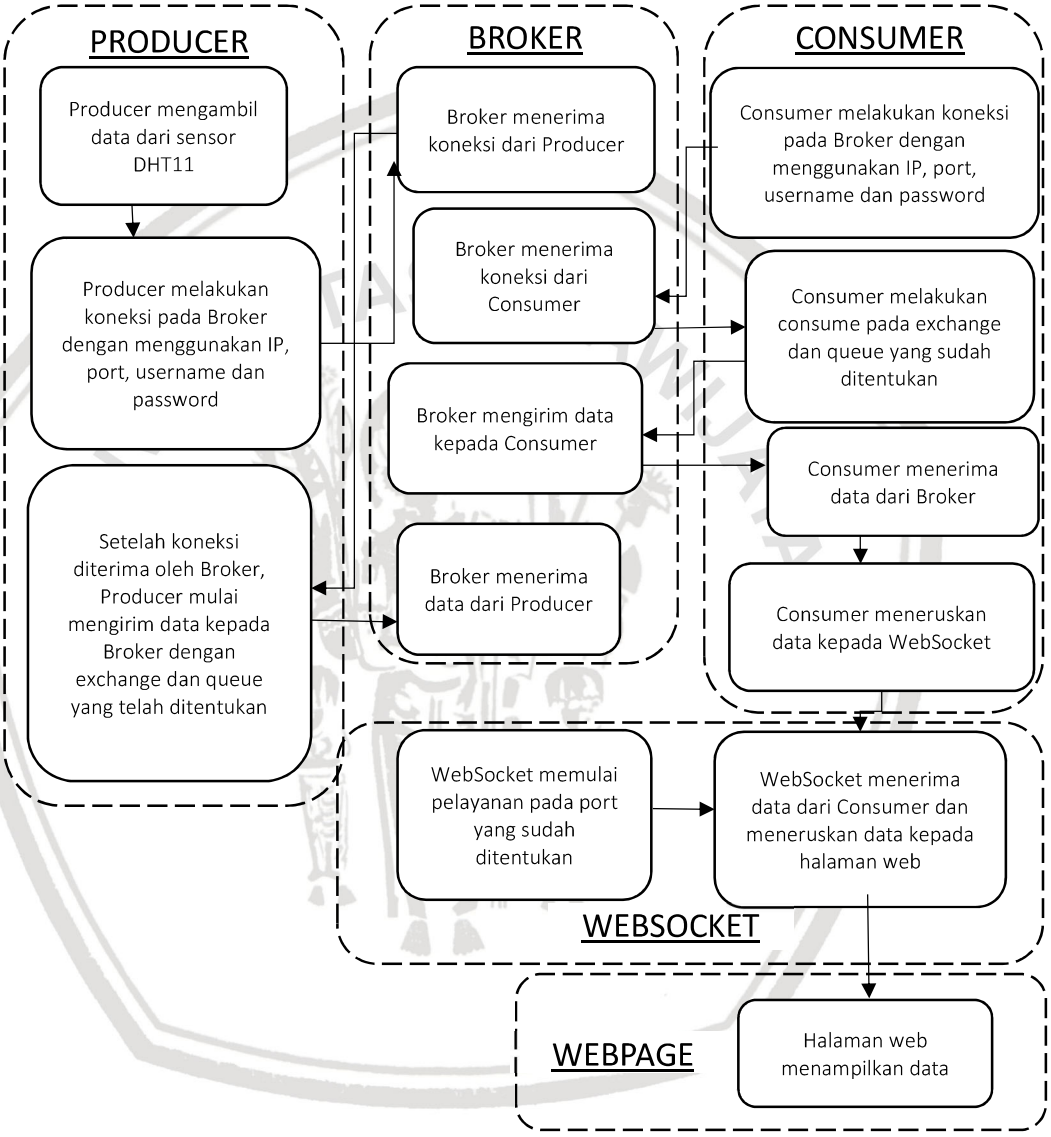
| Komponen | Perangkat Lunak |
|---|---|
| *Producer* | Raspbian OS Jessie, library Adafruit_Python_DHT dan library Pika |
| *Broker* | Raspbian OS Jessie dan *message broker* RabbitMQ |
| *Consumer* | Ubuntu OS 16.04 dan library Pika |
| *WebSocket* | Ubuntu OS 16.04 dan library Tornado |
| Pengguna | Google Chrome |

## 4.3 Alur Kerja Sistem

Sistem terdiri dari 4 komponen utama yang memiliki tugas masing-masing yaitu *Producer, Broker, Consumer,* dan *WebSocket*. Komponen *Producer*, *Broker* dan *Consumer* termasuk penerapan protokol *AMQP* pada sistem ini. Komponen *Producer* mengambil data suhu dan kelembapan udara menggunakan sensor DHT11, data tersebut didapatkan dan diolah menggunakan library Adafruit_DHT. Setelah itu komponen *Producer* melakukan koneksi dengan *Broker* menggunakan parameter *username*, *password*, *IP* dan *port* yang telah ditentukan, setelah koneksi *Broker* berhasil dilakukan, *Producer* mulai melakukan fungsi *publish* untuk mengirimkan data tersebut menuju *Broker* dengan menggunakan parameter *exchange* dan *routing key* yang telah ditentukan. Proses koneksi dan pengiriman data menuju *Broker* dilakukan dengan menggunakan library Pika. Setelah itu, pada komponen *Broker* akan menerima data yang dikirimkan oleh *Producer* dan data tersebut akan dimasukkan pada sebuah *queue* yang telah dibuat sebelumnya. Komponen *Broker* akan mengirimkan data yang ada pada *queue* tersebut menuju semua *Consumer* yang melakukan *consume* pada *queue* tersebut.

*Consumer* akan melakukan koneksi pada *Broker* dengan menggunakan parameter yang sama dengan yang digunakan oleh *Producer*. Setelah koneksi dengan *Broker* berhasil dilakukan, *Consumer* mulai melakukan fungsi *consume* untuk menerima data menggunakan parameter *queue* yang telah ditentukan, proses koneksi dan penerimaan data dari *Broker* dilakukan dengan menggunakan library Pika. Setelah data diterima pada sisi *Consumer*, data tersebut akan diteruskan menuju komponen *WebSocket*. Pada sisi komponen *WebSocket*, proses layanan yang dilakukan komponen ini dilakukan pada *port* yang telah ditentukan

dengan menggunakan library Tornado. Setelah proses layanan berhasil dilakukan, server *WebSocket* akan mulai mengirimkan data yang didapatkan dari *Consumer* menuju semua klien yang telah terhubung pada server, klien dapat terhubung kepada *WebSocket* dengan mengakses *IP* dan *port* dari server melalui *web browser*. Data tersebut akan ditampilkan pada sebuah halaman *web* yang diakses melalui *web browser* yang mendukung protokol *WebSocket*. Semua komponen yaitu, *Producer*, *Consumer*, *Broker*, *WebSocket* dan klien *web browser* harus terhubung pada satu jaringan nirkabel yang sama sehingga dapat saling berkomunikasi dan bertukar data. Gambar 4.2 merupakan alur kerja dari sistem.

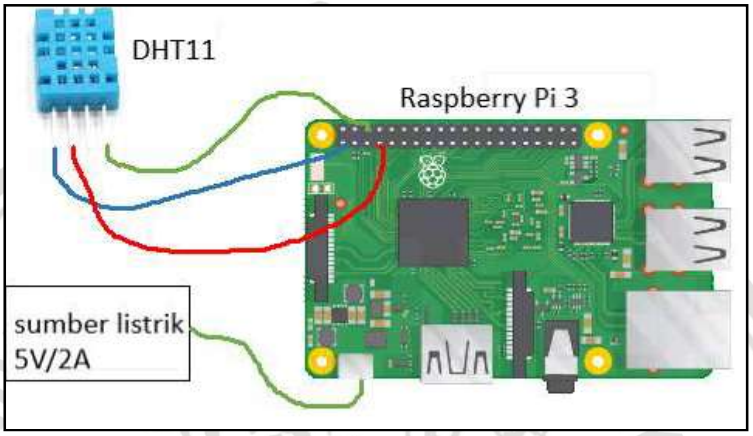



**Gambar 4.2 Diagram Alur Kerja Sistem**

## 4.4 Perancangan Perangkat Keras

*Producer* terdiri dari satu sensor dan mikrokomputer Raspberry Pi 3, sensor yang digunakan adalah sensor DHT11 yang berfungsi untuk mengambil data suhu

dan kelembapan udara. Sensor DHT11 dihubungkan dengan menggunakan kabel *jumper* kepada Raspberry Pi 3, bagian pada sensor yang dihubungkan pada Raspberry Pi 3 yaitu *VCC, DATA,* dan *GND*. *VCC* dihubungkan pada pin 1 yang memberi tegangan 3.3V, *DATA* dihubungkan pada pin 7 yaitu *GPIO* 4 yang berfungsi untuk memberikan input dan output, *GND* dihubungkan pada pin 6 yang berfungsi sebagai *ground* pada Raspberry Pi 3. Setelah proses pemasangan sensor, mikrokomputer Raspberry Pi 3 harus diberi sumber tegangan listrik sebesar 5V dan arus sebesar 2A agar dapat berfungsi dengan baik, mikrokomputer Raspberry Pi 3 yang digunakan oleh komponen *Broker* juga harus diberi sumber tegangan listrik sebesar 5V dan arus sebesar 2A agar dapat berfungsi dengan baik. Perancangan perangkat keras komponen *Producer* ditunjukkan pada gambar 4.3.



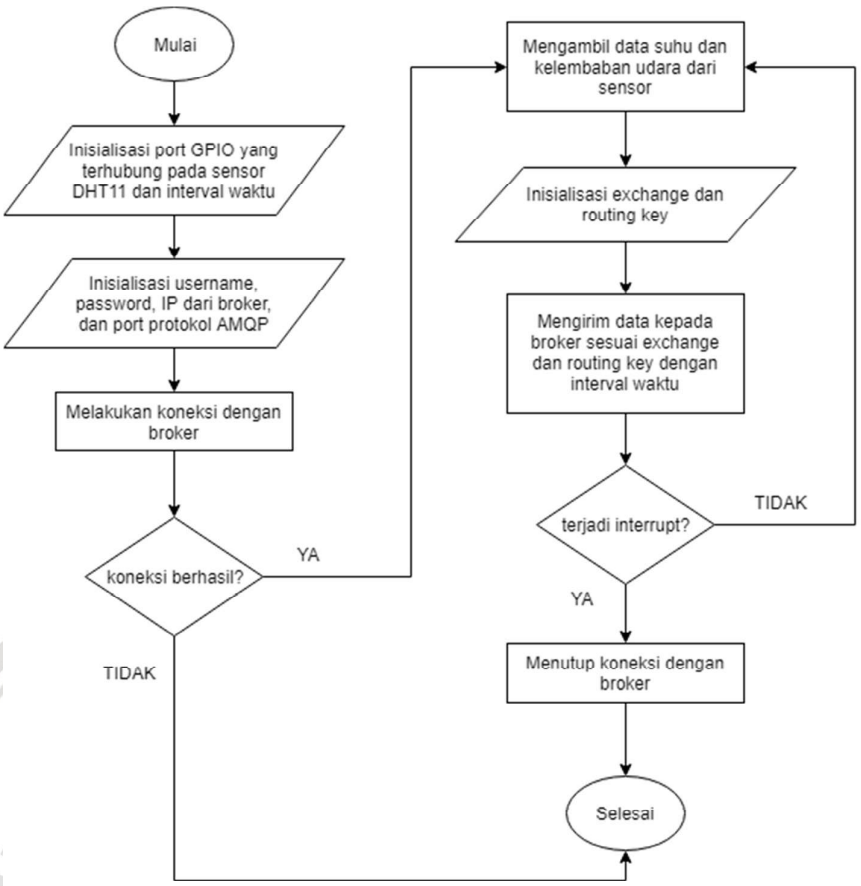

**Gambar 4.3 Perancangan Perangkat Keras AMQP Producer**

## 4.5 Perancangan Perangkat Lunak

Sesuai dengan perancangan yang telah dijelaskan pada bab sebelumnya, terdapat tiga perangkat lunak yang akan dirancang pada tahap ini yaitu, *Producer*, *Consumer* dan *WebSocket*. Masing-masing perangkat lunak akan dirancang sehingga sistem pemantauan yang akan dibangung dapat berjalan dengan baik.

### 4.5.1 Perancangan *Producer*

Pada penerapan protokol *AMQP*, *Producer* menyediakan informasi berupa data suhu dan kelembapan udara yang didapat dari sensor. Selain itu, *Producer* juga memiliki fungsi untuk mengirimkan data tersebut menuju *Broker*. Pada komponen *Producer* terdapat dua perangkat keras yang akan digunakan yaitu, mikrokomputer Raspberry Pi 3 dan sensor DHT11 yang proses perancangannya sudah dijelaskan sebelumnya. Perancangan perangkat lunak *Producer* dilakukan setelah melakukan instalasi perangkat lunak yang dibutuhkan yaitu library Adafruit_Python_DHT dan library Pika yang berjalan pada sistem operasi *Raspbian*. Selanjutnya, perancangan *Producer* akan digunakan pada bagian implementasi sehingga dapat mengambil data dari sensor DHT11, melakukan koneksi kepada *Broker* dan mengirimkan data yang didapat dari sensor DHT11 kepada *Broker* berdasarkan *exchange* dan *routing key*. *Flowchart* perancangan ditunjukkan pada gambar 4.4.

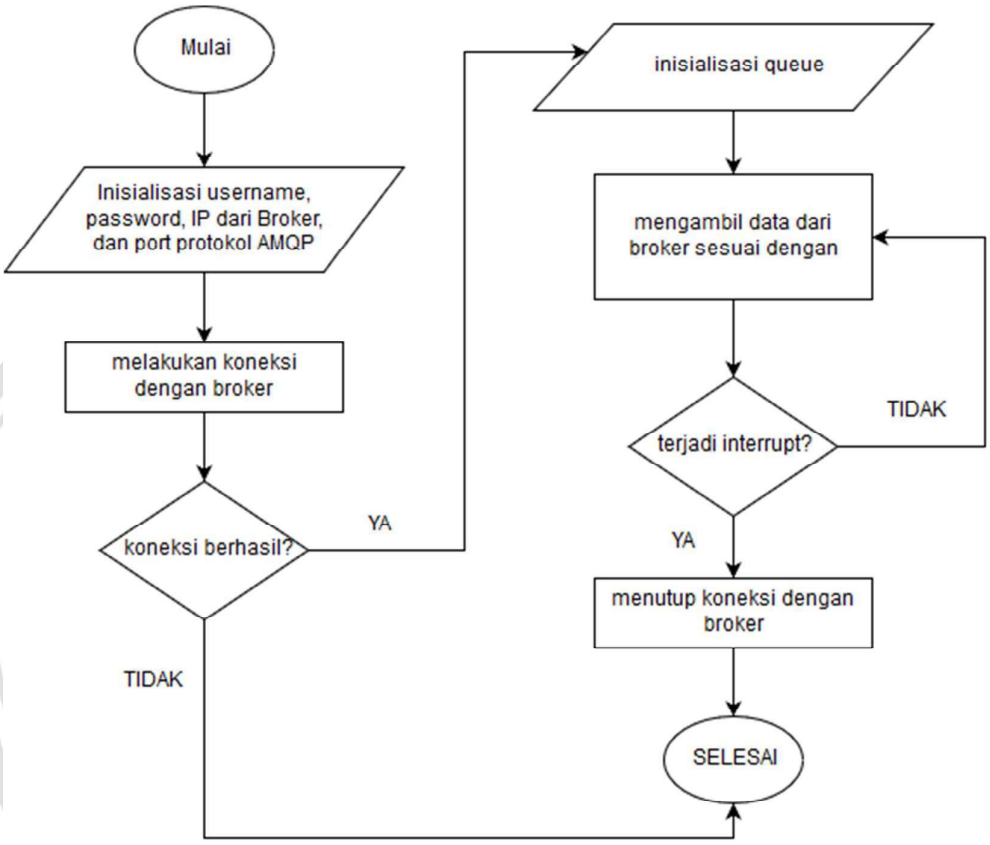



**Gambar 4.4 Perancangan Producer**

Inisialisasi *port GPIO* dan waktu *interval* merupakan langkah awal yang dilakukan pada proses *Consumer*. *Port GPIO* digunakan untuk mendapatkan input dari sensor DHT11 dan waktu *interval* yang berfungsi sebagai jeda pengiriman data. Lalu *Producer* melakukan inisialisasi *username, password*, *IP* dari *Broker* dan *port* dari protokol *AMQP* 0-9-1 yang digunakan untuk melakukan koneksi pada *Broker*. Jika proses koneksi pada *Broker* tidak berhasil, maka proses *Producer* akan berakhir. Apabila proses koneksi dengan *Broker* berhasil, *Producer* mulai mengambil data suhu dan kelembapan dari sensor DHT11 menggunakan library Adafruit_DHT. Selanjutnya, *Producer* menginisialisasi *exchange* dan *routing key* yang akan digunakan untuk melakukan proses *publish*, proses *publish* merupakan proses dimana *Producer* mengirimkan data kepada *Broker*. Selanjutnya *Producer* akan melakukan *publish* data kepada *Broker* sesuai dengan *exchange* dan *routing key* yang telah diinisialisasi sebelumnya, serta proses *publish* data dilakukan setiap *interval* (setiap 5 menit sekali) yang sudah ditentukan sebelumnya. Jika tidak ada *interrupt* yang terjadi, *Producer* akan mulai mengambil data dari sensor dan mengirim data tersebut kepada *Broker*. Apabila *interrupt* terdeteksi, maka proses akan berlanjut pada *Producer* menutup koneksi pada *Broker*.

### 4.5.2 Perancangan *Consumer*

Perangkat lunak *Consumer* berperan sebagai pengambil data dari *Broker*. Data yang didapat dari *Broker* akan digunakan oleh perangkat lunak *WebSocket* dan selanjutnya diteruskan kepada pengguna. Perancangan perangkat lunak *Consumer* dilakukan setelah melakukan instalasi perangkat lunak yang dibutuhkan yaitu library Pika yang berjalan pada sistem operasi *Ubuntu 16.04*. Selanjutnya, perancangan *Consumer* akan digunakan pada bagian implementasi sehingga dapat melakukan koneksi pada *Broker* dan mengambil data dari sebuah *queue* yang ada di dalam *Broker*. *Flowchart* perancangan ditunjukkan pada gambar 4.5.

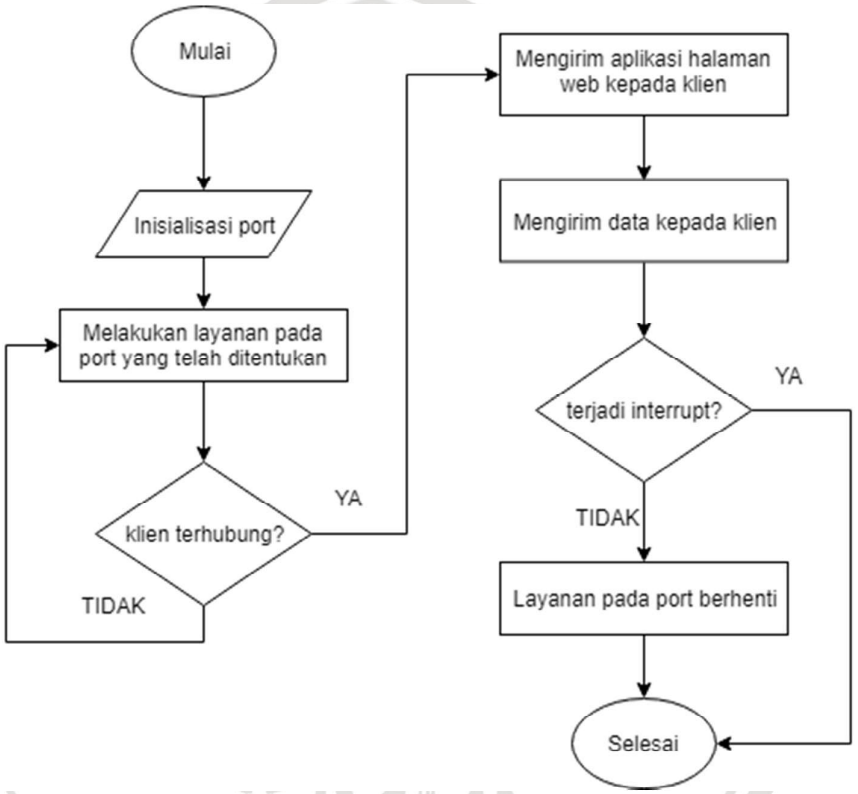



**Gambar 4.5 Perancangan Consumer**

Proses *Consumer* dimulai dari melakukan inisialisasi *username, password*, *IP* dari *Broker* dan *port* dari protokol *AMQP* 0-9-1 yang digunakan untuk melakukan koneksi pada *Broker*. Jika proses koneksi pada *Broker* tidak berhasil, maka proses *Consumer* akan berakhir dan aplikasi ditutup. Jika koneksi diterima oleh *Broker* maka proses akan berlanjut pada inisialisasi parameter *queue* yang digunakan untuk melakukan *consume* pada *Broker*, proses *consume* merupakan proses dimana aplikasi *Consumer* mengambil data dari *queue* yang ada pada *Broker*. Jika tidak terdapat *interrupt* maka proses akan kembali pada *Consumer* melakukan *consume* dengan menggunakan parameter *queue*, sehingga dapat menerima data dari *Broker* lagi. Jika terdapat *interrupt* maka proses akan berlanjut pada *Consumer* menutup koneksi pada *Broker* dan proses selesai.

### 4.5.3 Perancangan *WebSocket*

Di dalam sistem yang akan dibangun, perangkat lunak *WebSocket* berfungsi sebagai pemberi layanan kepada klien atau server. Server ini dapat diakses dengan menggunakan *IP* dan *port* yang digunakan oleh server melalui *web browser*. Selain itu, *WebSocket* juga memiliki fungsi untuk mengirimkan data yang didapatkan dari *Consumer* menuju semua klien yang telah terhubung kepadanya. Perancangan perangkat lunak *WebSocket* dilakukan setelah melakukan instalasi perangkat lunak yang dibutuhkan yaitu library Tornado yang berjalan pada sistem operasi *Ubuntu 16.04*. Selanjutnya, perancangan *WebSocket* akan digunakan pada bagian implementasi sehingga server dapat memberikan layanan pada *port* yang telah ditentukan, mengirim aplikasi halaman *web* dan data dari perangkat lunak *Consumer* kepada klien. *Flowchart* perancangan ditunjukkan pada gambar 4.6.




**Gambar 4.6 Perancangan WebSocket**

Pada perangkat lunak *WebSocket*, proses dimulai dengan melakukan inisialisasi *port*. Lalu dilanjutkan dengan melakukan layanan pada *port* yang sudah ditentukan, sehingga klien dapat melakukan koneksi kepada server dengan mengakses *IP* dan *port* yang digunakan oleh server. Jika tidak terdapat klien yang terhubung dengan *WebSocket*, *WebSocket* akan tetap melakukan layanan pada *port* yang sudah ditentukan. Apabila terdapat klien yang terhubung, aplikasi halaman *web* akan dikirim kepada klien dan ditampilkan pada *web browser*. Setelah halaman *web* berhasil ditampilkan, server akan mulai mengirimkan data kepada klien dan menampilkan data tersebut pada halaman *web*. Selanjutnya, jika tidak terdapat *interrupt* pada perangkat lunak *WebSocket* maka proses akan tetap

berlanjut yaitu server akan tetap mengirimkan data kepada klien yang terhubung. Jika terdapat *interrupt* maka perangkat lunak *WebSocket* akan menghentikan layanan dan proses selesai. Aplikasi *WebSocket* dapat melayani banyak klien dan dapat menerima koneksi saat mengirimkan data kepada klien yang telah terhubung.

## 4.6 Perancangan Pengujian

Perancangan pengujian dibutuhkan untuk mengetahui batasan minimal yang harus dipenuhi dalam pengujian. Dalam perancangan pengujian ini juga akan dibuat skenario dalam pengujian. Pada penelitian ini, akan dilakukan pengujian integritas sistem, pengujian integritas data dan pengujian keandalan sistem.

### 4.6.1 Perancangan Pengujian Integritas Sistem

Pengujian integritas sistem dilakukan untuk mengetahui interaksi antara dua atau lebih komponen untuk menjalankan satu fungsi tertentu. Rancangan pengujian ini akan dilakukan sebelum sistem digunakan untuk mengetahui apakah semua fungsi sistem yang ada pada kebutuhan fungsional sudah terpenuhi. Keberhasilan pengujian ini ditentukan oleh kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Informasi mengenai skenario dalam melakukan pengujian integritas sistem tersebut dijelaskan pada tabel 4.3 dibawah ini,

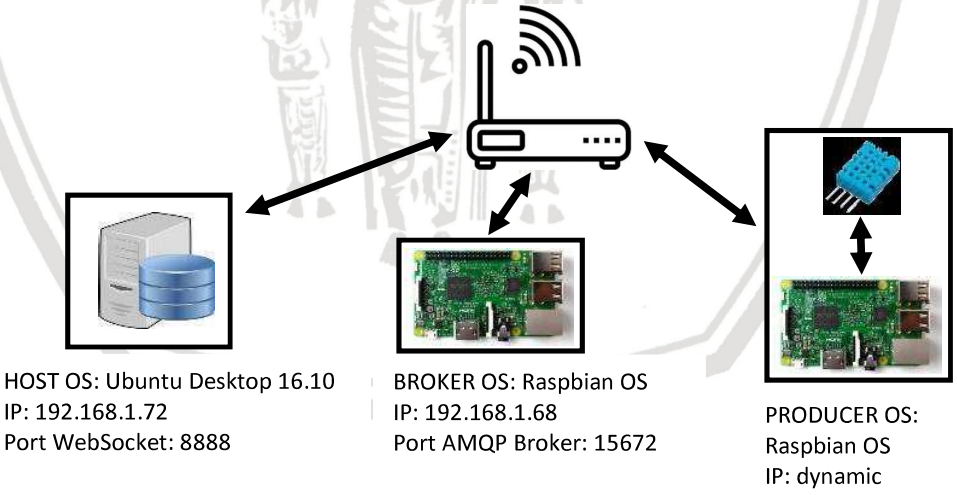
**Tabel 4.3 Skenario Pengujian Integritas Sistem**

| Kode | Kebutuhan Fungsional | Skenario |
|---|---|---|
| I_01 | *Broker* dapat menerima koneksi dari *Producer* dan *Consumer*. | • Ketika *Broker* sudah terhubung dengan *Access Point/wifi* dan *Producer*, *Consumer* memiliki hak akses yang valid.<br>• Maka *Broker* dapat menerima koneksi dari *Producer* dan *Consumer*. |
| I_02 | *Producer* dapat membaca data dari sensor DHT11 | • Ketika sensor DHT11 sudah terhubung dengan *Raspberri Pi*.<br>• Maka *Producer* dapat menampilkan data sensor. |
| I_03 | *Producer* dapat mengirim data kepada *Broker*. | • Ketika *Producer* sudah terhubung dengan *Broker* dan telah membuat sebuah *queue*.<br>• Maka *Broker* dapat menerima data yang dimasukkan kedalam *queue*. |
| I_04 | *Consumer* dapat menerima data dari *Broker*. | • Ketika *Consumer* sudah terhubung dengan *Broker* dan melakukan *consume* pada *queue*.<br>• Maka *Consumer* dapat menerima data dari *Broker*. |

| | | |
|---|---|---|
| I_05 | *WebSocket* dapat melakukan layanan dan halaman *web* dapat ditampilkan. | • Ketika *WebSocket* sudah terhubung pada *Access Point/wifi* dan *port* sudah ditentukan.<br>• Maka *WebSocket* dapat melakukan layanan pada *port* tersebut.<br>• Ketika *WebSocket* telah melakukan layanan pada *port* tertentu.<br>• Maka Halaman *web* dapat ditampilkan melalui *Web Browser*. |
| I_06 | *WebSocket* dapat melakukan *broadcast* data dan dapat ditampilkan pada halaman *web*. | • Ketika Halaman *web* dapat ditampilkan.<br>• Maka data dapat ditampilkan melalui halaman *web*. |

### 4.6.2 Perancangan Pengujian Integritas Data

Integritas data mengacu pada keandalan dan apakah data dapat dipercaya selama siklus hidupnya. Integritas data dapat menggambarkan keadaan data, misalnya, valid atau tidak valid atau proses memastikan dan menjaga keabsahan dan keakuratan data (Ng, 2018). Perancangan pengujian integritas data merupakan rancangan pengujian yang dilakukan untuk mengetahui apakah data yang mengalir pada sistem memiliki integritas. Pengujian akan dilakukan dengan mengirimkan 50 data yang didapatkan dari sensor DHT11 hingga ditampilkan pada halaman *web*. Dan akan dilakukan pengecekan apakah data yang ditampilkan pada halaman *web* sesuai dengan data yang dikirimkan dari *Producer*.




**Gambar 4.7 Rancangan Pengujian Integritas Data**

Pada gambar 4.7 *Producer* berjalan pada *Raspbian OS* yang bertugas sebagai pengambil data sensor DHT11 dan mengirim data tersebut kepada *Broker*. *Broker* berjalan pada *Raspbian OS* yang bertugas sebagai penerima data yang dikirim oleh *Producer* dan meneruskannya kepada *Consumer*, *Broker* berada pada alamat *IP* 192.168.1.68 dan *port* 5672. Sedangkan *host* yang bertugas sebagai *Consumer* yang menerima data dari *Broker* dan server *WebSocket* dijalankan pada *Ubuntu Dekstop* 16.10, server *WebSocket* dapat diakses melalui *IP* 192.168.1.72 dengan *port* 8888 melalui *web browser*. Tabel 4.4 merupakan tabel skenario pengujian yang akan dilakukan pada pengujian integritas data.

**Tabel 4.4 Skenario Pengujian Integritas Data**

| Kode | Pengujian | Keterangan | Hasil yang diharapkan |
|---|---|---|---|
| S_01 | Data yang dikirim oleh *Producer* akan sama dengan data yang ditampilkan pada halaman *web*. | Pengujian akan dilakukan dengan membandingkan 50 data yang dikirim oleh *Producer* dengan 50 data yang ditampilkan pada halaman *web*. | Sistem memiliki integritas data. |

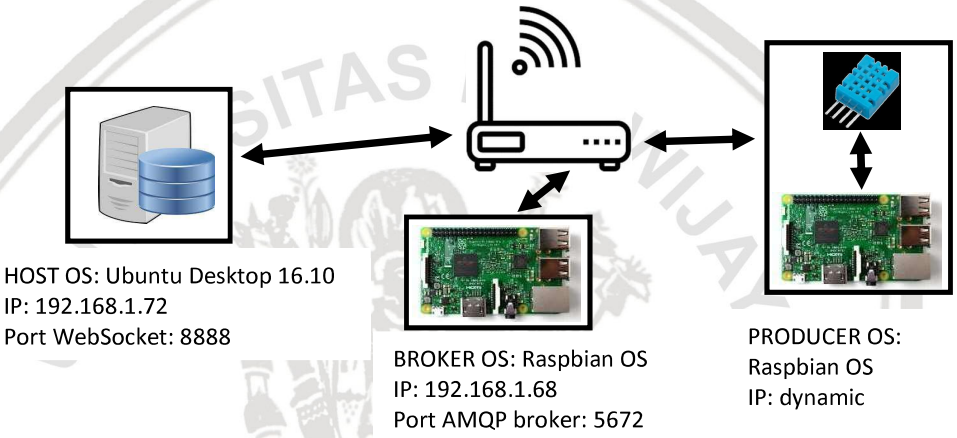
### 4.6.3 Perancangan Pengujian Keandalan Sistem

Di dalam jaringan, *delay* mengukur waktu yang dibutuhkan beberapa data untuk mencapai tujuannya. *Delay* adalah waktu tunda yang terjadi dalam proses pengiriman suatu paket dari titik awal ke titik tujuan. Dalam sebuah jaringan *delay* dapat menjadi sebuah acuan untuk menilai kualitas jaringan. Semakin kecil nilai *delay* yang dihasilkan maka semakin baik jaringan tersebut. Cara untuk menghitung *delay* yaitu, waktu paket diterima dikurangi waktu paket dikirimkan (KCRAZY666, 2015).

Pengujian keandalan sistem terbagi menjadi empat bagian yaitu pengujian dengan menggunakan parameter *delay* yang memiliki skenario dimana dilakukan dengan memantau paket berisi data dari mulai *Producer* mengirim data dengan jeda pengirim paket per-5 detik hingga ditampilkan pada halaman *web*. Pada saat *Producer* mulai mengirimkan data sensor akan ditangkap *timestamp* dan dikirim bersamaan dengan data sensor menuju *Broker*. Dan saat data yang dikirimkan oleh *Producer* ditampilkan pada halaman *web* akan ditangkap juga *timestamp* pada saat tersebut. Skenario ini akan dilakukan dengan 5 variasi yaitu, pengiriman 25, 50, 100, 150 dan 200 data. Setiap variasi akan diambil rata-rata *delay* dan setiap variasi akan diulang sebanyak 10 kali, setelah setiap variasi diulang sebanyak 10 kali akan diambil rata-rata *delay* dari hasil tiap variasi yang telah diulang sebanyak 10 kali.

Bagian kedua memiliki langkah-langkah yang sama dengan skenario sebelumnya, hanya saja terdapat perbedaan dimana terdapat variasi yaitu beberapa jumlah *thread* yang dianggap sebagai *Producer* menjalankan fungsi untuk mengirim 1 pesan. Variasi yang terdapat pada skenario ini yaitu, 25, 50, 75, 100 dan 125 *Producer*. Setiap variasi tersebut akan diulang sebanyak 10 kali. Pengujian ketiga dilakukan dengan melakukan pemantauan pada paket yang berisi

data pada saat *WebSocket* mulai melakukan *broadcast* data tersebut kepada seluruh klien yang terkoneksi. Saat *WebSocket* melakukan *broadcast* akan ditangkap *timestamp* dan ditampilkan pada halaman *web*, begitu juga saat halaman *web* menampilkan data tersebut. *Producer* akan mengirimkan 25 data dengan variasi jeda yaitu, 1, 0,5, 0,1, 0,05 dan 0,01 detik dengan 5 klien yang terkoneksi pada *WebSocket*. Dan diambil rata-rata *delay* pada tiap variasi yang diulang sebanyak 10 kali. Sedangkan pada pengujian keempat dilakukan dengan menangkap *log* penggunaan memori pada *Broker* dan *Consumer* dengan menggunakan perangkat lunak *psrecord* saat beberapa *Producer* melakukan pengiriman data hingga diterima oleh *Consumer* yang digunakan untuk mengetahui rata-rata penggunaan memori. Variasi jumlah *Producer* pada skenario ini yaitu, 90, 120, 150, 180, dan 210 *Producer*. Pengujian ini dilakukan dengan menjalankan beberapa *thread* yang melakukan proses pengambilan data sensor dan *publish* menuju *Broker*. Setiap variasi akan diulang sebanyak 10 kali untuk mendapatkan hasil yang *reliable*.




**Gambar 4.8 Rancangan Pengujian Keandalan Sistem**

Pada gambar 4.8 *Producer* berjalan pada *Raspbian OS* yang bertugas sebagai pengambil data sensor DHT11 dan mengirim data tersebut kepada *Broker*. *Broker* berjalan pada *Raspbian OS* yang bertugas sebagai penerima data yang dikirim oleh *Producer* dan meneruskannya kepada *Consumer*, *Broker* berada pada alamat *IP* 192.168.1.68 dan *port* 5672. Sedangkan *host* yang bertugas sebagai *Consumer* yang menerima data dari *Broker* dan server *WebSocket* dijalankan pada *Ubuntu Dekstop* 16.10, server *WebSocket* dapat diakses melalui *IP* 192.168.1.72 dengan *port* 8888 melalui *web browser*. Skenario pengujian keandalan sistem akan dijelaskan pada tabel 4.5.

**Tabel 4.5 Skenario Pengujian Keandalan Sistem**

| Kode | Parameter | Keterangan |
|---|---|---|
| K_01 | *Delay* | Menangkap *timestamp* saat *Producer* melakukan pengiriman data dan saat data ditampilkan pada halaman *web* dengan variasi jumlah data 25, 50, 100, 150 dan 200 data. Data dikirim tiap 5 detik. *Timestamp* digunakan untuk menghitung *average delay*. |

| | | |
|---|---|---|
| K_02 | *Delay* | Menangkap *timestamp* saat *Producer* melakukan pengiriman data dan saat data ditampilkan pada halaman *web* dengan variasi jumlah *Producer* 25, 50, 75, 100 dan 125 *Producer*. *Timestamp* digunakan untuk menghitung *average delay*. |
| K_03 | *Delay* | Menangkap *timestamp* saat *Producer* melakukan pengiriman 25 data dan saat data ditampilkan pada halaman *web* dengan variasi jeda pengiriman data 1, 0,5, 0,1, 0,05 dan 0,01 detik. Terdapat 5 klien yang terkoneksi pada *WebSocket*. *Timestamp* digunakan untuk menghitung *average delay*. |
| K_04 | Memori | Menangkap *log* penggunaan memori pada *Broker* dan *Consumer* dengan menggunakan perangkat lunak *psrecord* saat beberapa *Producer* mengirimkan data. Variasi jumlah *Producer* yaitu 90, 120, 150, 180, dan 210. |

# BAB 5 IMPLEMENTASI SISTEM

Pada bab ini menjelaskan tentang proses implementasi sistem yang akan digunakan dalam proses penelitian. Proses implementasi dimulai dari implementasi perangkat keras dan perangkat lunak yang mengacu pada bab perancangan sistem sebelumnya.

## 5.1 Implementasi Perangkat Keras

Implementasi perangkat keras dilakukan sesuai dengan perancangan yang telah dilakukan pada tahap sebelumnya. Implementasi perangkat keras dimulai dengan memasang perangkat sensor DHT11 pada mikrokomputer Raspberry Pi 3 dengan menggunakan kabel. Ada tiga bagian dari perangkat sensor yang harus tersambung pada mikrokomputer Raspberry Pi 3 yaitu, *vcc*, data dan *ground*. Bagian *vcc* dihubungkan pada pin *GPIO* 02 yang merupakan sumber tegangan listrik sebesar 5V *DC*, bagian data dihubungkan pada pin *GPIO* 07 yang berfungsi untuk menerima masukan dan bagian *ground* dihubungkan pada pin *GPIO* 06 yang berfungsi sebagai *ground*. Hasil implementasi perangkat keras dapat dilihat pada gambar 5.1 dibawah ini,

**Gambar 5.1 Hasil Implementasi Perangkat Keras**

## 5.2 Implementasi Perangkat Lunak

Immplementasi perangkat lunak, dibagi menjadi tiga bagian besar yang harus dilakukan. Bagian-bagian tersebut meliputi implementasi perangkat lunak *Producer*, *Consumer* dan *WebSocket*.

### 5.2.1 Implementasi *Producer*

Implementasi perangkat lunak *Producer* dimulai dengan melakukan instalasi dari beberapa library yaitu, library Pika dan Adafruit_DHT. Serta library time dan datetime yang digunakan untuk mengambil informasi waktu. Proses penulisan aplikasi *Producer* dilakukan menggunakan bahasa pemgrograman Python. Aplikasi *Producer* dijalankan pada perangkat keras Raspberry Pi 3 dengan sistem operasi Raspbian Jessie. Tabel 5.1 merupakan *pseudocode* dari perangkat lunak *Producer*.

**Tabel 5.1 *Pseudocode Producer***

```
1   START
2        IMPORT pika, Adafruit_DHT as dht, time, datetime
3        SET gpio <- 4
4        SET interval <- 300
5        SET sensor_data <- {'temperature': 0, 'humidity': 0}
6        SET ip <- 192.168.1.68
7        SET credentials <- pika.credentials.PlainCredentials
8                           ATTRIBUTE 'edgar','edgar'
9        SET param <- pika.ConnectionParameters ATTRIBUTE ip,
10                  5672, '/', credentials
11       SET connection <- pika.BlockingConnection ATTRIBUTE
12                         param
13       SET channel <- connection.channel
14       STATEMENT try DO
15            LOOP while STATEMENT True DO
16                 SET humidity,temperature <- dht.read_retry
17                                             ATTRIBUTE
18                                             dht.DHT11, gpio
19                 SET humidity <- round ATTRIBUTE humidity, 2
20                 SET temperature <- round ATTRIBUTE
21                                    temperature, 2
22                 SET s <- u"Temperature: {:g}\u00b0C,
23                           Humidity: {:g}%".format ATTRIBUTE
24                           temperature, humidity
25                 CALL FUNCTION basic_publish ATTRIBUTE
26                                        SET exchange <-'edgar',
27                                    SET routing_key <-'edgar',
28                                 SET body <- s + ' | dikirim :
29                           '+datetime.datetime.now().strftime
30                                              ('%H:%M:%S.$f')[:-3]
31                 DELAY interval
32       EXCEPTION KeyboardInterrupt DO
33            pass
34       connection.close()
35  FINISH
```

Penjelasan secara runtut dari *pseudocode* hasil implementasi perangkat lunak *Producer* ditampilkan pada tabel 5.2.

**Tabel 5.2 Penjelasan *Pseudocode Producer***

| | |
|---|---|
| Baris 2 | Melakukan *import* beberapa library yang dibutuhkan oleh perangkat lunak *Producer*. Library pika digunakan untuk menerapkan protokol *AMQP* sehingga dapat berkomunikasi dengan *Broker*. Untuk membaca data dari sensor digunakan library Adafruit_DHT. Sedangkan library time dan datetime berfungsi untuk mengambil data waktu. |
| Baris 3-6 | Deklarasi pin *GPIO* yang digunakan oleh sensor, penetapan jeda dari pengambil data sensor, deklarasi variabel untuk memasukkan data dari sensor dan menentukan *IP* yang digunakan oleh *Broker*. |
| Baris 7-13 | Menetapkan *username* dan *password* yang digunakan untuk autentikasi pada *Broker*. |

| | |
|---|---|
| | Deklarasi parameter untuk melakukan koneksi yang berisi variabel ip, *port* protokol *AMQP*, *virtual host* yang diakses dan variabel credentials.<br>Melakukan koneksi dengan *Broker* menggunakan parameter yang sudah ditetapkan pada variabel param.<br>Membuka *channel* baru pada *Broker* untuk berkomunikasi dan mengirimkan data. |
| Baris 14-15 | Awal mula perulangan. |
| Baris 16-18 | Membaca data dari sensor DHT11 lalu memasukkan data tersebut kedalam variabel humidity dan temperature. |
| Baris 19-21 | Membulatkan data yang ada pada variabel humidity dan temperature sebanyak 2-digit angka setelah koma. |
| Baris 22-24 | Memasukkan data dari variabel humidity dan temperature kedalam variabel s dengan format yang telah ditentukan. |
| Baris 25-30 | Memanggil fungsi basic_publish dari library pika yang diisi dengan parameter *exchange* = edgar dan *queue* = edgar untuk mengirim pesan kedalam *queue* dengan nama edgar pada *Broker*. Sedangkan parameter *body* atau pesan yang dikirim berisi data dari variabel s dan penanda waktu saat pesan tersebut dikirim. |
| Baris 31 | Digunakan untuk memberi jeda pembacaan data sensor dan pengiriman pesan. |
| Baris 32-34 | Jika terdapat *interrupt* yang dilakukan menggunakan *keyboard*, koneksi dengan *Broker* akan diputus dan aplikasi *Producer* berhenti. |

### 5.2.2 Implementasi *Consumer*

Pembangunan perangkat lunak *Consumer* dilakukan menggunakan bahasa pemrograman Python yang membutuhkan library pika yang telah terpasang sebelumnya. Serta fungsi *Thread* dari library threading dan logging. Dibutuhkan juga semua variabel dan fungsi yang ada pada aplikasi *WebSocket*. Beberapa hal yang sudah disebutkan dibutuhkan supaya *Consumer* dapat menjalankan peran dalam mengambil data dari *Broker*. *Pseudocode* dari hasil pembangunan aplikasi *Consumer* dapat dilihat pada Tabel 5.3.

**Tabel 5.3 *Pseudocode Consumer***

```
1   START
2       IMPORT pika, logging
3       FROM threading IMPORT Thread
4       FROM websocket IMPORT ALL
5       SET ip <- 192.168.1.68
6       SET credentials <- pika.credentials.PlainCredentials
7                          ATTRIBUTE 'edgar','edgar'
8       SET param <- pika.ConnectionParameters ATTRIBUTE ip,
9                                          5672, '/', credentials
10      SET connection <- pika.BlockingConnection ATTRIBUTE
11                                                  param
12      SET channel <- connection.channel
13      FUNCTION threaded_rmq()
```

```
14          CALL FUNCTION basic_consume ATTRIBUTE
15                                          consumer_callback,
16                                    queue="edgar", no_ack=True
17          CALL FUNCTION start_consuming
18      END FUNCTION
19      FUNCTION disconnect_to_rabbitmq
20          CALL FUNCTION stop_consuming
21          CALL FUNCTION close
22          DISPLAY Disconnected from + ip
23      END FUNCTION
24      FUNCTION consumer_callback ATTRIBUTE ch, method,
25                                 properties, body
26          LOOP itm <-- clients DO
27              SEND body TO itm
28          END LOOP
29      END FUNCTION
30      IF __name__ <- __main__
31          DISPLAY Starting thread RabbitMQ
32          SET threadRMQ <- CALL FUNCTION Thread ATTRIBUTE
33                                                  threaded_rmq
34          RUN threadRMQ
35          DISPLAY Starting thread Tornado
36          SET threadTornado <- CALL FUNCTION Thread
37                                       ATTRIBUTE startTornado
38          RUN threadTornado
39          STATEMENT try DO
40              DISPLAY Server ready. Press enter to stop
41          EXCEPT SyntaxError DO
42              Pass
43          STATEMENT try DO
44              DISPLAY Disconnecting from RabbitMQ..
45              CALL FUNCTION disconnect_to_rabbitmq
46          EXCEPT Exception, e DO
47              Pass
48          CALL FUNCTION stopTornado
49  FINISH
```

Tabel 5.4 merupakan penjelasan dari *pseudocode* hasil pembangunan perangkat lunak *Consumer*.

**Tabel 5.4 Penjelasan *Pseudocode Consumer***

| | |
|---|---|
| Baris 2-4 | Melakukan *import* beberapa library yang dibutuhkan oleh perangkat lunak *Producer*. Library pika digunakan untuk menerapkan protokol *AMQP* sehingga dapat berkomunikasi dengan *Broker*. Selain itu, logging digunakan untuk menampilkan tulisan pada aplikasi digunakan library logging.<br>Melakukan *import* fungsi Thread dari library threading yang digunakan untuk menjalankan aplikasi *Consumer*.<br>Memanggil semua objek, variabel dan fungsi yang ada pada file websocket.py untuk menjalankan server *WebSocket*. |
| Baris 5 | Menentukan *IP* yang digunakan oleh *Broker*. |
| Baris 6-12 | Menetapkan *username* dan *password* yang digunakan untuk autentikasi pada *Broker*. |

| | |
|---|---|
| | Deklarasi parameter untuk melakukan koneksi yang berisi variabel ip, *port* protokol *AMQP*, *virtual host* yang diakses dan variabel credentials.<br>Melakukan koneksi dengan *Broker* menggunakan parameter yang sudah ditetapkan pada variabel param.<br>Membuka *channel* baru pada *Broker* untuk berkomunikasi dan mengirimkan data. |
| Baris 13-18 | Method ini akan dijalankan menggunakan *thread* yang berfungsi untuk mengambil data dari *queue* dengan nama edgar yang ada pada *Broker*.<br>Memanggil fungsi *start_consume* untuk memulai mengambil data dari *Broker*. |
| Baris 19-23 | Fungsi ini digunakan untuk menghentikan pengambilan data dan memutus koneksi dari *Broker*. Fungsi ini juga menampilkan teks yang menunjukkan bahwa proses pengambilan data dan koneksi dari *Broker* telah berhenti. |
| Baris 24-29 | Digunakan untuk menerima pesan dari *Broker* dan memasukkan pesan tersebut kedalam variabel pesan.<br>Perulangan yang ada fungsi ini digunakan untuk mengirim pesan yang didapat dari *Broker* menuju semua klien yang terhubung dengan server *WebSocket*. |
| Baris 30 | Semua proses yang ada pada aplikasi *Consumer* dijalankan dari dalam *statement if*. |
| Baris 31-34 | Menampilkan teks yang menunjukkan bahwa *thread* RabbitMQ mulai dijalankan.<br>Inisialisasi target dari *thread* sehingga dapat menjalankan semua proses yang dimiliki oleh *Consumer* dan mengeksekusi *thread* tersebut. |
| Baris 35-38 | Menampilkan teks yang menunjukkan bahwa *thread* Tornado mulai dijalankan.<br>Inisialisasi target dari *thread* sehingga dapat menjalankan semua proses yang dimiliki oleh *WebSocket* dan mengeksekusi *thread* tersebut. |
| Baris 39-48 | Digunakan untuk menampilkan beberapa teks dengan input yang sudah ditentukan.<br>Menghentikan semua proses dari aplikasi *Consumer* dengan memanggil *method* yang memiliki fungsi untuk menutup koneksi dari *Broker*.<br>Menghentikan semua proses dari aplikasi *WebSocket* dengan memanggil *method* yang memiliki fungsi untuk menghentikan layanan dari server *WebSocket*. |

### 5.2.3 Implementasi *WebSocket*

*Pseudocode* yang ditampilkan pada Tabel 5.5 merupakan hasil dari implementasi perangkat lunak *WebSocket* yang menggunakan library

tornado.ioloop, tornado.web, tornado.websocket dan logging. Beberapa library yang telah disebutkan supaya *WebSocket* dapat menjalankan layanan dan mengirimkan data yang didapat dari *Consumer* kepada semua klien yang telah terhubung. Aplikasi *WebSocket* ditulis menggunakan Bahasa pemrograman Python.

**Tabel 5.5 *Pseudocode WebSocket***

```
1   START
2       IMPORT tornado.ioloop, tornado.web, tornado.websocket
3       SET clients <- []
4       CLASS SocketHandler ATTRIBUTE
5                           tornado.websocket.WebSocketHandler
6           FUNCTION open ATTRIBUTE self
7               DISPLAY WebSocket opened
8               INPUT self TO clients
9           END FUNCTION
10          FUNCTION on_close ATTRIBUTE self
11              DISPLAY WebSocket closed
12              DELETE self FROM clients
13          END FUNCTION
14      END CLASS
15      CLASS MainHandler ATTRIBUTE tornado.web.RequestHandler
16          FUNCTION get ATTRIBUTE self
17              SEND aplikasi.html TO self
18          END FUNCTION
19      END CLASS
20      SET application <- CLASS tornado.web.Application
21                         ATTRIBUTE
22                         [(r'/ws', SocketHandler),
23                         (r"/", MainHandler),
24                         ]) END CLASS
25      FUNCTION startTornado
26          SET port <- 8888
27          START tornado.ioloop.IOLoop.instance()
28      END FUNCTION
29      FUNCTION stopTornado
30          STOP tornado.ioloop.IOLoop.instance()
31      END FUNCTION
32  FINISH
```

Penjabaran mengenai *pseudocode* hasil pembangunan perangkat lunak *WebSocket* ditampilkan pada Tabel 5.6.

**Tabel 5.6 Penjelasan *Pseudocode WebSocket***

| | |
|---|---|
| Baris 2 | Melakukan *import* beberapa library yang dibutuhkan oleh perangkat lunak *Producer*. Library tornado.ioloop, tornado.web dan tornado.websocket digunakan untuk membangun sebuah server yang menerapkan protokol *WebSocket* sehingga dapat memberikan layanan pada semua klien yang terhubung. |
| Baris 3 | Deklarasi *list* dengan nama clients. |
| Baris 4-14 | *Class* ini memiliki beberapa fungsi yang digunakan untuk menangani setiap klien yang melakukan koneksi dengan server. |

| | |
|---|---|
| | Fungsi open yang digunakan untuk menunjukkan bahwa terdapat klien yang terhubung dengan server dan memasukkan setiap klien yang terhubung kedalam list clients. Sedangkan fungsi on_close berjalan ketika terdapat klien yang memutuskan koneksi dengan server dan menghapus klien tersebut dari list clients. |
| Baris 15-19 | Fungsi dari *class* ini untuk menangani setiap *request* yang dilakukan oleh klien yang telah terhubung dengan server. Dan server akan membalas *request* tersebut dengan mengirimkan sebuah halaman *web* yang digunakan untuk menampilkan data hasil pemantauan. |
| Baris 20-24 | Digunakan untuk mengelompokkan *request handler* sehingga menjadi sebuah aplikasi *web*. |
| Baris 25-28 | Menentukan *port* yang digunakan oleh server dalam melayani setiap klien yang terhubung.<br>Menjalankan IOLoop yang digunakan untuk menjalankan layanan dari server. Selain itu, fungsi ini juga menangani setiap event pada koneksi antara klien dan server. |
| Baris 29-31 | Perulangan ini berfungsi untuk mengirimkan setiap data yang diambil dari perangkat lunak *Consumer* kepada setiap klien yang terhubung dengan server *Websocket*. |
| Baris 32-34 | Fungsi ini berfungsi untuk menghentikan layanan dari server. |

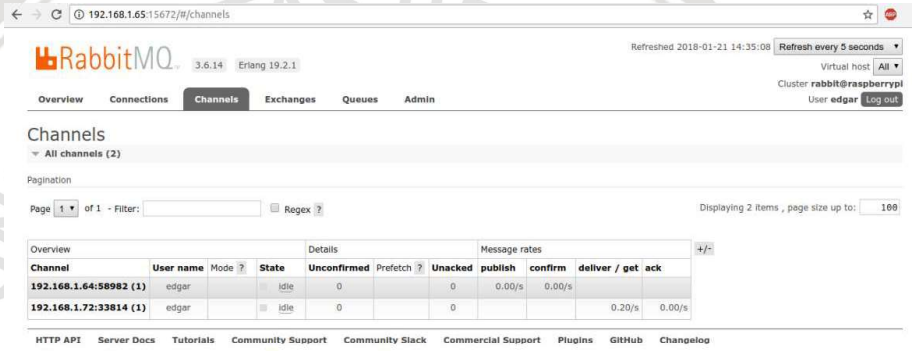
# BAB 6 PENGUJIAN DAN ANALISIS SISTEM

Pada bab ini akan dijelaskan mengenai pengujian dari implementasi sistem yang sudah dibuat. Pengujian ini dibutuhkan untuk mengetahui apakah kinerja dari sistem sudah sesuai dengan kebutuhan. Pengujian akan dilakukan berdasarkan pengujian yang telah dijelaskan pada bab sebelumnya.

## 6.1 Hasil dan Analisis Pengujian Integritas Sistem

Pengujian integritas dilakukan untuk mengetahui apakah aplikasi sesuai dengan kebutuhan fungsional. Pengujian integritas merupakan *black box* testing yang dilakukan untuk menguji dua atau lebih komponen dari sistem untuk menjalankan satu fungsi tertentu. Pengujian dikatakan berhasil jika tidak ada hasil pengujian yang menunjukkan kesalahan. Penjelasan mengenai kesesuaian hasil dengan kebutuhan sistem dapat dilihat pada poin di bawah ini.

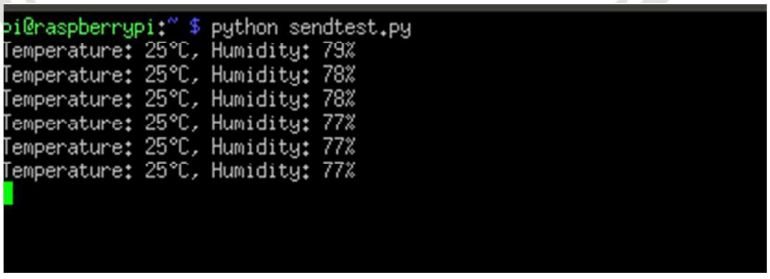
a. Skenario [I_01] *Broker* dapat menerima koneksi dari *Producer* dan *Consumer*




**Gambar 6.1 Hasil Pengujian Skenario [I_01]**

Gambar 6.1 merupakan tampilan halaman *web* dari *Broker*, pada gambar tersebut terdapat 2 *device* yang terkoneksi pada *Broker*. Dimana *device* yang memiliki *IP* 192.168.1.64 bertindak sebagai *Producer* dan *IP* 192.168.1.72 bertindak sebagai *Consumer*. Berdasarkan hasil pengujian integritas sistem dengan skenario [I_01], dapat disimpulkan bahwa *Broker* dapat menerima koneksi dari *Producer* dan *Consumer*. Sehingga dapat melakukan aktivitas *publish* dan *consume* data dari *queue*.

b. Skenario [I_02] *Producer* dapat membaca data dari sensor DHT11



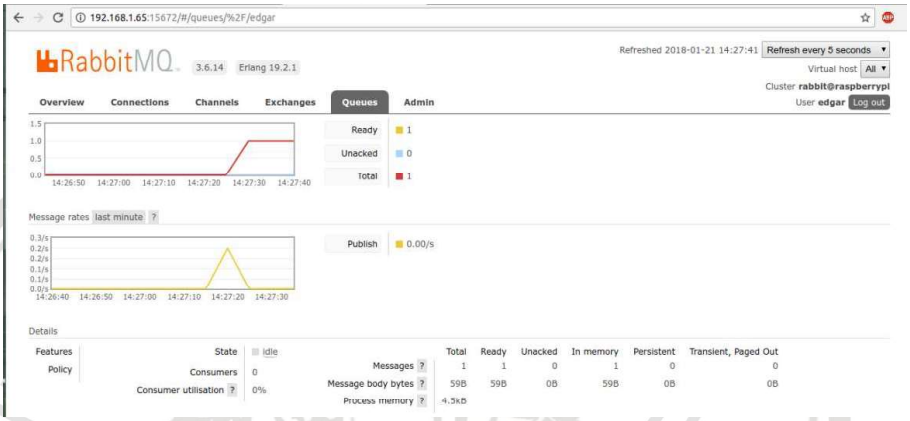

**Gambar 6.2 Hasil Pengujian Skenario [I_02]**

Tampilan dari *terminal* yang ditunjukkan pada gambar 6.2 merupakan proses yang dilakukan *Producer* untuk membaca data dari sensor dan menampilkannya. Sehingga dapat disimpulkan bahwa implementasi *Producer* yang telah dijelaskan pada bab sebelumnya berhasil dilakukan.

c. Skenario [I_03] *Producer* dapat mengirim data kepada *Broker*


**Gambar 6.3 Hasil Pengujian Skenario [I_03]**

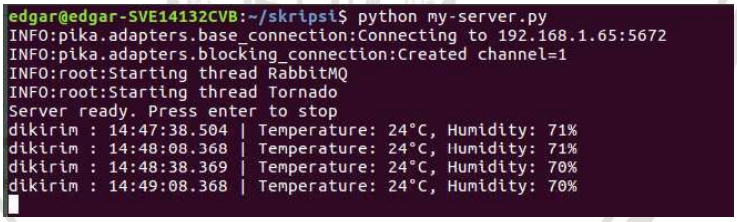
Gambar 6.3 merupakan tampilan pada *Producer* yang menunjukkan bahwa data berhasil dikirim menuju *Broker*.




**Gambar 6.4 Hasil Pengujian Skenario [I_03]**

Gambar 6.4 merupakan tampilan pada *Broker* yang menunjukkan adanya aktifitas data pada *queue* yang digunakan untuk pertukaran data antara *Producer* dan *Consumer*. Dengan hasil yang ditunjukkan pada gambar 6.3 dan 6.4, dapat disimpulkan bahwa pembangunan perangkat lunak *Producer* berhasil dilakukan.

d. Skenario [I_04] *Consumer* dapat melakukan koneksi dengan *Broker*.




**Gambar 6.5 Hasil Pengujian Skenario [I_04]**

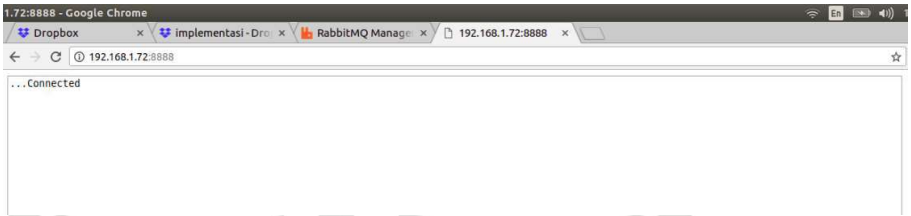
Tampilan *terminal* pada gambar 6.5 menunjukkan bahwa *Consumer* berhasil menerima data dari *Producer*. Tampilan terminal tersebut membuktikan implementasi perangkat lunak *Consumer* yang telah dijelaskan pada bab sebelumnya berhasil dilakukan.

e. Skenario [I_05] *WebSocket* dapat melakukan layanan dan halaman *web* dapat ditampilkan

```
edgar@edgar-SVE14132CVB:~/skripsi$ python my-server.py
INFO:pika.adapters.base_connection:Connecting to 192.168.1.65:5672
INFO:pika.adapters.blocking_connection:Created channel=1
INFO:root:Starting thread RabbitMQ
INFO:root:Starting thread Tornado
Server ready. Press enter to stop
INFO:tornado.access:200 GET / (192.168.1.72) 2.27ms
INFO:tornado.access:101 GET /ws (192.168.1.72) 0.78ms
INFO:root:WebSocket opened
WARNING:tornado.access:404 GET /favicon.ico (192.168.1.72) 0.51ms
```

**Gambar 6.6 Hasil Pengujian Skenario [I_05]**

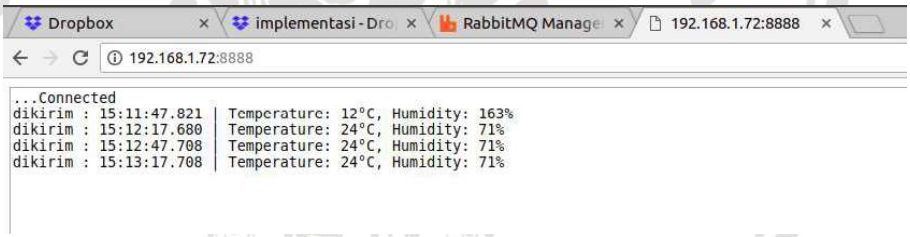
Gambar 6.6 menunjukkan server *WebSocket* sudah berjalan dan mampu melakukan layanan pada *port* yang telah ditentukan.




**Gambar 6.7 Hasil Pengujian Skenario [I_05]**

Gambar 6.7 merupakan tampilan pada *Web Browser* yang menunjukkan bahwa halaman *web* bisa diakses pada *IP* dan *port* dari server *WebSocket*. Berdasarkan gambar 6.6 dan 6.7, pembangunan server *WebSocket* yang telah dijelaskan pada bab sebelumnya berhasil dilakukan.

f. Skenario [I_06] *WebSocket* dapat melakukan *broadcast* data dan dapat ditampilkan pada halaman *web*.




**Gambar 6.8 Hasil Pengujian Skenario [I_06]**

Implementasi server *WebSocket* berhasil dilakukan, hal tersebut dibuktikan dengan hasil pengujian skenario [I_06]. Gambar 6.8 membuktikan bahwa halaman *web* dapat menampilkan data yang dikirimkan oleh server *WebSocket*.

Kesimpulan dari hasil pengujian ini adalah pengujian integritas sistem pada semua skenario memiliki kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Hal tersebut dapat memberikan kesimpulan lain bahwa pengujian integritas sistem berhasil dilakukan dan sistem dapat menjalankan fungsi-fungsi yang dibutuhkan sehingga sistem dapat berjalan secara utuh. Selain itu, keberhasilan fungsi juga menggambarkan bahwa integrasi setiap perangkat yang ada dalam sistem dapat terhubung dan saling berkomunikasi dengan baik melalui protokol *AMQP* dan *WebSocket*.

## 6.2 Hasil dan Analisis Pengujian Integritas Data

Pengujian integritas data dilakukan untuk mengetahui apakah data yang dikirim oleh *Producer* hingga ditampilkan pada halaman *web* sesuai dengan urutan dan memiliki nilai yang sama. Sesuai dengan penjelasan skenario [S_01] pada bab sebelumnya, berikut hasil dari pengujian yang telah dilakukan pada tabel 6.1.

**Tabel 6.1 Hasil Pengujian Integritas Data**

| Data ke- | Producer | Webpage | Data ke- | Producer | Webpage | Data ke- | Producer | Webpage | Data ke- | Producer | Webpage | Data ke- | Producer | Webpage |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 27°C, 57% | 27°C, 57% | 11 | 30°C, 57% | 30°C, 57% | 21 | 27°C, 54% | 27°C, 54% | 31 | 30°C, 57% | 30°C, 57% | 41 | 15°C, 156% | 15°C, 156% |
| 2 | 30°C, 58% | 30°C, 58% | 12 | 30°C, 57% | 30°C, 57% | 22 | 13°C, 155% | 13°C, 155% | 32 | 27°C, 54% | 27°C, 54% | 42 | 27°C, 54% | 27°C, 54% |
| 3 | 15°C, 156% | 15°C, 156% | 13 | 27°C, 54% | 27°C, 54% | 23 | 30°C, 57% | 30°C, 57% | 33 | 30°C, 57% | 30°C, 57% | 43 | 30°C, 57% | 30°C, 57% |
| 4 | 27°C, 55% | 27°C, 55% | 14 | 27°C, 54% | 27°C, 54% | 24 | 27°C, 54% | 27°C, 54% | 34 | 30°C, 56% | 30°C, 56% | 44 | 27°C, 54% | 27°C, 54% |
| 5 | 15°C, 156% | 15°C, 156% | 15 | 30°C, 57% | 30°C, 57% | 25 | 27°C, 54% | 27°C, 54% | 35 | 27°C, 54% | 27°C, 54% | 45 | 30°C, 56% | 30°C, 56% |
| 6 | 30°C, 57% | 30°C, 57% | 16 | 30°C, 57% | 30°C, 57% | 26 | 30°C, 57% | 30°C, 57% | 36 | 15°C, 156% | 15°C, 156% | 46 | 27°C, 54% | 27°C, 54% |
| 7 | 30°C, 57% | 30°C, 57% | 17 | 30°C, 57% | 30°C, 57% | 27 | 30°C, 57% | 30°C, 57% | 37 | 27°C, 54% | 27°C, 54% | 47 | 27°C, 54% | 27°C, 54% |
| 8 | 27°C, 55% | 27°C, 55% | 18 | 27°C, 54% | 27°C, 54% | 28 | 30°C, 57% | 30°C, 57% | 38 | 13°C, 155% | 13°C, 155% | 48 | 27°C, 54% | 27°C, 54% |
| 9 | 27°C, 55% | 27°C, 55% | 19 | 13°C, 155% | 13°C, 155% | 29 | 27°C, 54% | 27°C, 54% | 39 | 27°C, 54% | 27°C, 54% | 49 | 27°C, 54% | 27°C, 54% |
| 10 | 27°C, 54% | 27°C, 54% | 20 | 13°C, 155% | 13°C, 155% | 30 | 27°C, 54% | 27°C, 54% | 40 | 27°C, 54% | 27°C, 54% | 50 | 27°C, 54% | 27°C, 54% |

Tabel 6.1 berisi perbandingan antara data yang dikirimkan oleh *Producer* dan yang ditampilkan pada halaman *web*. Berdasarkan tabel 6.1, hasil pengujian integritas data dapat disimpulkan bahwa data yang dikirim oleh *Producer* dan ditampilkan pada halaman *web* sama dan sesuai dengan urutan.

Kesimpulan dari hasil pengujian ini adalah pengujian integritas data pada semua skenario memiliki kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Hal tersebut dapat memberikan kesimpulan lain bahwa pengujian integritas data berhasil dilakukan dan setiap data yang mengalir pada sistem memiliki integritas. Selain itu, keberhasilan pengujian integritas data juga menggambarkan bahwa komunikasi data melalui protokol *AMQP* dapat berjalan dengan baik.

## 6.3 Hasil dan Analisis Pengujian Keandalan Sistem

Pengujian keandalan sistem dilakukan berdasarkan skenario yang telah dijelaskan pada bab sebelumnya. Pengujian ini dilakukan untuk mengetahui nilai *delay* dan penggunaan memori dari protokol yang digunakan di dalam sistem.

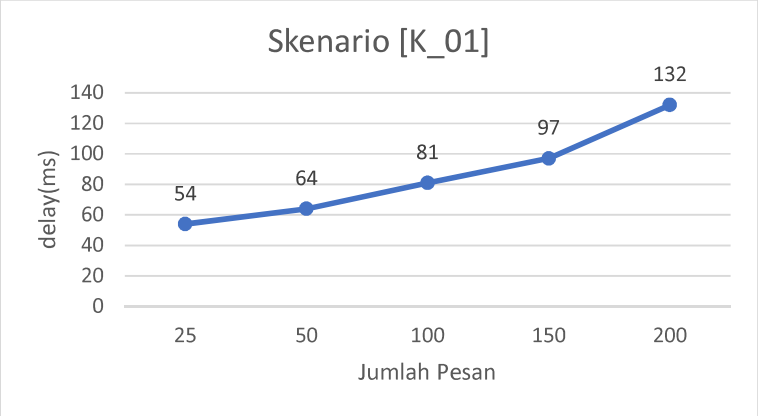
### 6.3.1 Pengujian Skenario [K_01]

Pengujian dilakukan sesuai dengan skenario [K_01] yang telah dijelaskan pada bab sebelumnya. Pengujian ini dilakukan untuk mengetahui rata-rata *delay* dari sistem yang berjalan. Berikut akan ditampilkan tabel hasil pengujian skenario [K_01] pada tabel 6.2.

**Tabel 6.2 Hasil Pengujian Skenario [K_01]**

| Jumlah Pesan | *Delay(ms)* |
|---|---|
| 25 | 54 |
| 50 | 64 |
| 100 | 81 |
| 150 | 97 |
| 200 | 132 |

Berdasarkan data pada tabel 6.2, maka dapat dibuat grafik hasil pengujian skenario [K_01] pada gambar 6.9 berikut.




**Gambar 6.9 Grafik Hasil Pengujian Skenario [K_01]**

Pada pengujian ini, jeda pengiriman pesan pada tiap variasi yaitu 5 detik. Berdasarkan data pada tabel 6.2, didapati bahwa ketika jumlah pesan sebanyak 25 dikirim oleh *Producer*, didapat rata-rata *delay* 54*ms*. Ketika jumlah pesan sebanyak 50 dikirim oleh *Producer*, rata-rata *delay* meningkat menjadi 64*ms*. Rata-rata *delay* 81*ms* didapat saat jumlah pesan sebanyak 100 dikirim oleh *Producer*. Disaat jumlah pesan sebanyak 150 dikirim oleh *Producer*, didapat rata-rata *delay* 97*ms*. Sedangkan untuk jumlah pesan sebanyak 200, rata-rata *delay* meningkat menjadi 132*ms*.

Kesimpulan dari hasil pengujian pada semua variasi memiliki kesesuaian antara hasil yang didapat dengan hasil yang diharapkan, dimana rata-rata *delay* yang didapatkan dari hasil pengujian tidak lebih dari 1000*ms*. Selain itu, dapat dilihat pada gambar 6.9 bahwa semakin banyak pesan yang dikirim oleh *Producer* semakin tinggi pula rata-rata *delay* yang didapatkan, hal ini menunjukkan bahwa semakin lama koneksi pada *Broker* terbuka dapat mempengaruhi performa dari sistem yang telah dibuat.

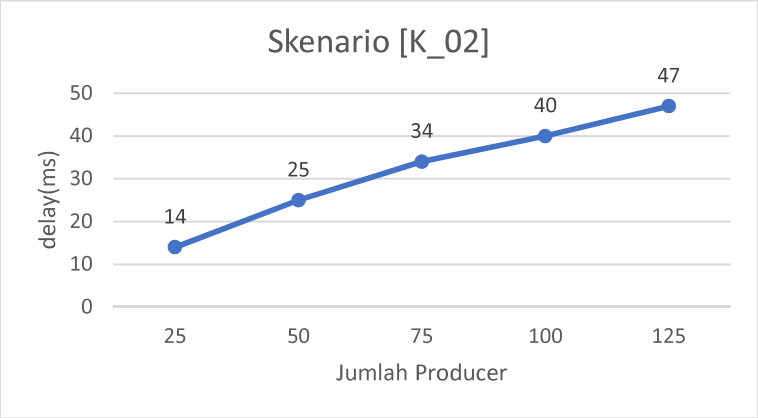
### 6.3.2 Pengujian Skenario [K_02]

Pengujian skenario [K_02] yang telah dijelaskan pada bab sebelumnya dilakukan untuk mengetahui rata-rata *delay* dari sistem yang berjalan. Berikut merupakan tabel hasil pengujian skenario [K_02].

**Tabel 6.3 Hasil Pengujian Skenario [K_02]**

| Jumlah *Producer* | *Delay(ms)* |
|---|---|
| 25 | 14 |
| 50 | 25 |
| 75 | 34 |
| 100 | 40 |
| 125 | 47 |

Berdasarkan data pada tabel 6.3, maka dapat dibuat grafik hasil pengujian skenario [K_02] pada gambar 6.10 berikut.




**Gambar 6.10 Grafik Hasil Pengujian Skenario [K_02]**

Pada pengujian ini, setiap *Producer* mengirim 1 buah pesan kepada *Broker*. Berdasarkan data pada tabel 6.3, didapati bahwa ketika jumlah *Producer* sebanyak 25 mengirim pesan, didapat rata-rata *delay* 14*ms*. Disaat jumlah *Producer* sebanyak 50 mengirim pesan, rata-rata *delay* meningkat menjadi 25*ms*. Ketika jumlah *Producer* sebanyak 75 mengirim pesan, didapat rata-rata *delay* 34*ms*. Rata-rata *delay* 40*ms* didapat ketika 100 *Producer* mengirim pesan. Dan untuk jumlah *Producer* sebanyak 125, didapat rata-rata *delay* 47*ms*.

Kesimpulan dari hasil pengujian pada semua variasi memiliki kesesuaian antara hasil yang didapat dengan hasil yang diharapkan, dimana rata-rata *delay* yang didapatkan dari hasil pengujian tidak lebih dari 1000*ms*. Selain itu, dapat dilihat pada gambar 6.10 bahwa semakin banyak *Producer* yang melakukan *publish* data kepada *Broker* semakin tinggi pula rata-rata *delay* yang didapatkan, hal ini menunjukkan bahwa semakin banyak *Producer* yang melakukan *publish* kepada *Broker* dalam waktu yang hampir bersamaan mempengaruhi performa dari sistem yang telah dibuat.

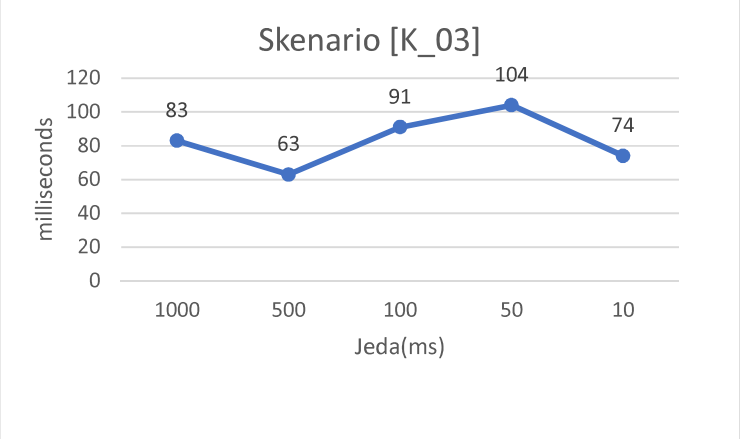
### 6.3.3 Pengujian Skenario [K_03]

Untuk mengetahui rata-rata *delay* dari sistem yang berjalan, pengujian dengan skenario [K_03] perlu dilakukan. Langkah-langkah pengujian ini telah dijelaskan pada bab sebelumnya. Berikut akan ditampilkan tabel hasil pengujian skenario [K_03] pada tabel 6.4.

**Tabel 6.4 Hasil Pengujian Skenario [K_03]**

| Jeda(*ms*) | *Delay(ms)* | | | | | Rata-rata *delay(ms)* |
|---|---|---|---|---|---|---|
| | *User* 1 | *User* 2 | *User* 3 | *User* 4 | *User* 5 | |
| 1000 | 83 | 79 | 82 | 83 | 89 | 83 |
| 500 | 66 | 59 | 69 | 57 | 65 | 63 |
| 100 | 82 | 83 | 125 | 79 | 84 | 91 |
| 50 | 96 | 102 | 111 | 101 | 109 | 104 |
| 10 | 69 | 77 | 73 | 75 | 78 | 74 |

Berdasarkan data pada tabel 6.4, maka dapat dibuat grafik hasil pengujian skenario [K_03] pada gambar 6.11 berikut.




**Gambar 6.11 Grafik Hasil Pengujian Skenario [K_03]**

Pada pengujian ini, terdapat 5 klien yang terkoneksi pada server *WebSocket* dan 1 *Producer* yang melakukan *publish* data. Berdasarkan data pada tabel 6.4, didapati bahwa ketika jeda pengiriman data 1 detik, rata-rata *delay* adalah 83*ms*. Ketika jeda pengiriman data 0,5 detik, didapat rata-rata *delay* 63*ms*. Rata-rata *delay* 91*ms* didapat saat jeda pengiriman data 0,1 detik. Disaat jeda pengiriman data 0,05 detik, rata-rata *delay* meningkat menjadi 104*ms*. Sedangkan pada jeda pengiriman data 0,01 detik, didapat rata-rata *delay* 74*ms*. Sehingga dapat disimpulkan bahwa jeda pengiriman data oleh *Producer* dapat mempengaruhi nilai *delay* dan juga mempengaruhi performa dari sistem.

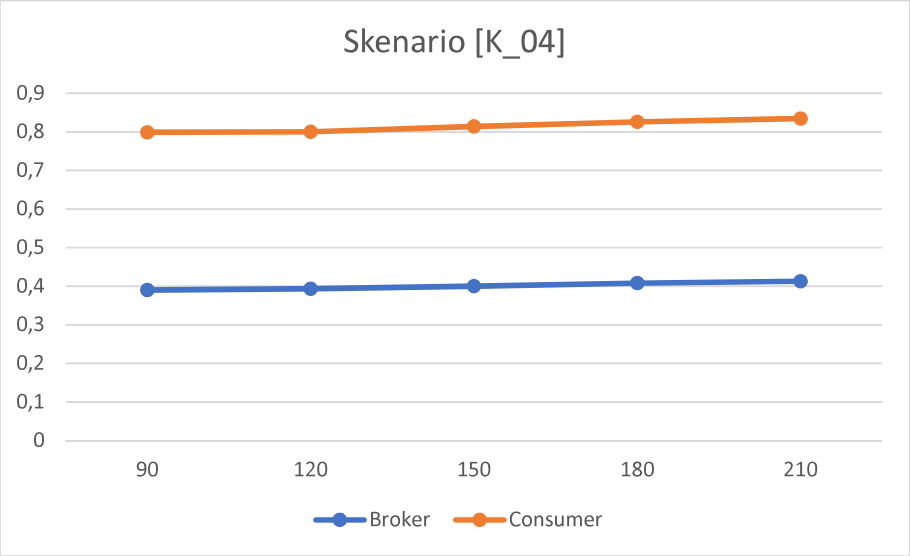
### 6.3.4 Pengujian Skenario [K_04]

Untuk mengetahui rata-rata penggunaan memori oleh *Broker* dan *Consumer* saat menerima koneksi dan data dari beberapa *Producer*, pengujian dengan skenario [K_04] perlu dilakukan. Langkah-langkah pengujian ini telah dijelaskan pada bab sebelumnya. Berikut akan ditampilkan tabel hasil pengujian skenario [K_04] pada tabel 6.5.

**Tabel 6.5 Hasil Pengujian Skenario [K_04]**

| Jumlah *Producer* | Penggunaan Memori | |
|---|---|---|
| | *Broker* | *Consumer* |
| 90 | 0.3901 | 0.7988 |
| 120 | 0.393 | 0.8004 |
| 150 | 0.4007 | 0.8143 |
| 180 | 0.408 | 0.826 |
| 210 | 0.4132 | 0.8345 |

Berdasarkan data pada tabel 6.5, maka dapat dibuat grafik hasil pengujian skenario [K_04] pada gambar 6.12 berikut.




**Gambar 6.12 Grafik Hasil Pengujian Skenario [K_04]**

Pada pengujian ini, setiap *Producer* mengirim 1 buah pesan kepada *Broker*. Berdasarkan data pada tabel 6.5, didapati bahwa ketika jumlah *Producer* sebanyak 90 mengirim pesan, penggunaan memori pada *Broker* dan *Consumer* adalah 0.3901 *MB* dan 0.7988 *MB*. Disaat 120 *Producer* mengirim pesan, rata-rata penggunaan memori pada *Broker* dan *Consumer* meningkat menjadi 0.3930 *MB* dan 0.8004 *MB*. Ketika jumlah *Producer* sebanyak 150 mengirim pesan, didapat rata-rata penggunaan memori pada *Broker* dan *Consumer* yaitu 0.4007 *MB* dan 0.8143 *MB*. Rata-rata penggunaan memori pada *Broker* yaitu 0.4080 *MB* dan pada *Consumer* yaitu 0.8260 *MB* didapat ketika 180 *Producer* mengirim pesan. Dan untuk jumlah *Producer* sebanyak 210, didapat rata-rata penggunaan memori pada *Broker* dan *Consumer* yaitu 0.4132 *MB* dan 0.8345 *MB*.

Kesimpulan dari hasil pengujian pada semua variasi yaitu semakin banyak *Producer* yang melakukan koneksi dan pengiriman data menyebabkan meningkatnya penggunaan memori pada perangkat keras *Raspberry Pi 3* yang digunakan oleh *Broker* dan *Consumer*. Selain itu, jumlah maksimum dari *Producer* hanya 210 dikarenakan perangkat mikrokomputer *Raspberry Pi 3* yang digunakan oleh *Producer* hanya mampu menjalankan 210 *thread* secara bersamaan dalam pengujian ini.

# BAB 7 PENUTUP

Bab ini akan menjelaskan kesimpulan berdasarkan dengan tahap-tahap yang telah dilakukan sebelumnya yaitu, kebutuhan, perancangan, implementasi dan pengujian. Terdapat juga saran pengembangan yang berkaitan dengan penelitian yang berhubungan.

## 7.1 Kesimpulan

Setelah tahapan-tahapan pada penelitian ini selesai dilakukan, maka kesimpulan yang dapat ditarik adalah sebagai berikut:

1. Dalam penelitian ini protokol *AMQP* dapat diterapkan pada sistem pemantauan suhu dan kelembapan udara dengan menggunakan *message broker* RabbitMQ. Untuk menjalankan aplikasi *Producer* dan *Consumer* yang ditulis dengan menggunakan bahasa pemrograman Python dibutuhkan library Pika sehingga dapat berkomunikasi dengan *Broker* RabbitMQ.
2. Dari hasil pengujian yang telah dilakukan pada bab sebelumnya, dapat dilihat bahwa sistem dapat melakukan semua fungsi dengan baik dan sesuai dengan yang diharapkan serta data yang mengalir pada sistem memiliki integritas. Sedangkan, berdasarkan hasil pengujian keandalan sistem menunjukkan bahwa semakin lama koneksi pada *Broker* terbuka, semakin banyaknya *Producer*, dan variasi jeda yang diterapkan pada pengujian keandalan sistem terbukti mempengaruhi performa dari sistem. Selain itu, penggunaan memori pada *Broker* dan *Consumer* dipengaruhi oleh banyaknya *Producer* yang melakukan pengiriman data.

## 7.2 Saran

Beberapa saran yang dapat dilakukan dalam pengembangan terkait dengan penelitian ini, dijelaskan sebagai berikut:

1. Dapat dikembangkan lebih lanjut seperti menambah sebuah *database* untuk menyimpan data yang dikirim oleh *Producer*, sehingga diperoleh data historis yang bisa diakses kapan saja.
2. Dapat menggunakan *Broker* yang membutuhkan lebih sedikit *resource* sehingga tidak membebani Raspberry Pi 3 dan sistem dapat memiliki tingkat skalabilitas yang lebih tinggi.

# DAFTAR REFERENSI


ajie, 2016. *MENGUKUR SUHU DAN KELEMBABAN UDARA DENGAN SENSOR DHT11 DAN ARDUINO.* [Online] Available at: <http://saptaji.com/2016/08/10/mengukur-suhu-dan-kelembaban-udara-dengan-sensor-dht11-dan-arduino/> [Diakses 29 January 2018].

Alex, X. A. S., 2015. *WebSocket.* [Online] Available at: <https://bertzzie.com/knowledge/javascript-lanjut/WebSocket.html> [Diakses 2 February 2018].

Alfian, S., 2016. *PENGERTIAN RASPBERRY DAN CARA INSTALL WINDOWS 10 IOT Pi2 DENGAN NOOBS.* [Online] Available at: <http://tikus-installasi.blogspot.co.id/2016/3/pengertian-raspberry-dan-cara-install.html> [Diakses 3 February 2018].

Anon., 2007. [Online] Available at: <https://www.rabbitmq.com/> [Diakses 29 January 2018].

Anon., 2007. *WebSocket.* [Online] Available at: websocket.org [Diakses 23 February 2018].

Anon., 2010. [Online] Available at: https://www.amazon.com/DHT11-Digital-Temperature-Humidity-Sensor/dp/B00V2DWL2E

Anon., 2010. *Understanding AMQP.* [Online] Available at: <https://spring.io/understanding/AMQP> [Diakses 27 January 2018].

Anon., 2013. *Overview of the Internet of things,* jenewa: International Telecommunication Union.

Authors, T. T., 2009. *Tornado Web Server.* [Online] Available at: <http://www.tornadoweb.org/en/stable/> [Diakses 5 February 2018].

B. Babovic, Z., Protic, J. & Milutinovic, V., 2016. Web Performance Evaluation for Internet of Things Applications. *IEEE Access,* pp. 6974-6991.

Darsiwan, 2016. *Apa itu WebSocket.* [Online] Available at: <https://www.codepolitan.com/menegtahui-apa-itu-websocket> [Diakses 2 February 2018].

E. Luzuriaga, J. et al., 2015. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. *2015 IEEE 12th Consumer Communications and Networking Conference (CCNC): CCNC 2015 Workshops - VENITS,* pp. 931-936.

Fette, I. & Melnikov, A., 2011. *The WebSocket Protocol.* [Online] Available at: <https://tools.ietf.org/html/rfc6455> [Diakses 11 February 2018].

Fitzpatrick, J., 2016. *Everything You Need to Know About Getting Started with the Raspberry Pi.* [Online] Available at:

<https://www.howtogeek.com/138281/the-htg-guide-to-getting-started-with-raspberry-pi/> [Diakses 21 February 2018].

Guoqiang, S., Yanming, C., Chao, Z. & Yanxu, Z., 2013. Design and Implementation of a Smart IoT Gateway. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing.*

Hu, Y. & Cheng, W., 2017. Research and Implementation of Campus Information Push System Based on WebSocket. *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE).*

Jayanti, L. S. E., 2016. Kesehatan Lingkungan Udara Ruang Rawat Inap Rumah Sakit Syekh Yusuf Kabupaten Gowa. *Higiene,* pp. 34-40.

KCRAZY666, 2015. *PENGERTIAN BANDWIDTH THROUGPUT DELAY JITTER DAN OSI LAYER.* [Online] Available at: <https://jarkomtelkom.wordpress.com/2015/09/02/pengertian-bandwidth-througput-delay-jitter-dan-osi-layer/> [Diakses 14 February 2018].

Labib, I. H., Adhitya, B. & Reza, A. S., 2018. Implementasi Protokol Websocket Pada Perangkat Non IP Berbasis (Studi Kasus: Sistem Monitoring Suhu dan Kontroling Lampu LED). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer Vol. 2, No. 6, Juni 2018, hlm. 2058-2066,* pp. 2058-2066.

lady, a., 2017. *Overview.* [Online] Available at: <https://learn.adafruit.com/dht/overview> [Diakses 17 May 2018].

Marsh, G. A. S. D., 2009. Scaling Advanced Message Queuing Protocol (AMQP) Architecture with Broker Federation and InfiniBand.

Ng, C., 2018. *What is Data Integrity and How Can You Maintain it?.* [Online] Available at: <https://blog.varonis.com/data-integrity/> [Diakses 5 June 2018].

Pieter, H., 2017. *Understanding When to use RabbitMQ or Apache Kafka.* [Online] Available at: <https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka> [Diakses 21 January 2018].

Pope, D., 2012. *Synchronous/Asynchronous Model.* [Online] Available at: <http://pythonhosted.org/nucleon.amqp/asyncmodel.html> [Diakses 2 March 2018].

Rouse, M., 2018. *Advanced Message Queuing Protocol (AMQP).* [Online] Available at: <https://whatis.techtarget.com/definition/Advanced-Message-Queuing-Protocol-AMQP> [Diakses 4 March 2018].

Rukshani, A., 2017. *Blocking I/O and non-blocking I/O.* [Online] Available at: <https://medium.com/coderscorner/tale-of-client-server-and-socket-a6ef54a74763> [Diakses 1 March 2018].

S, A., 2014. *Tornado: A Python Web Framework That's Simple and Quick.* [Online] Available at: <https://opensourceforu.com/2014/11/tornado-a-python-web-framework-simple-and-quick/> [Diakses 10 March 2018].

Singh, M., 2008. *RABBITMQ: UNDERSTANDING MESSAGE BROKER.* [Online] Available at: <https://www.3pillarglobal.com/insights/rabbitmq-understanding-message-broker> [Diakses 29 January 2018].