PENGEMBANGAN SISTEM PENYIMPANAN DATA BERBASIS MONGODB DAN GRIDFS UNTUK MENYIMPAN DATA YANG BERAGAM DARI NODE SENSOR

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Gabreil Arganata
NIM: 135150201111272

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2017

PENGESAHAN

PENGEMBANGAN SISTEM PENYIMPANAN DATA BERBASIS MONGODB DAN GRIDFS UNTUK MENYIMPAN DATA YANG BERAGAM DARI NODE SENSOR

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Gabreil Arganata
NIM: 135150201111272

Skripsi ini telah diuji dan dinyatakan lulus pada
11 Agustus 2017

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eko Sakti P., S.Kom., M.Kom	Widhi Yahya, S.Kom., M.Sc
NIK. 201102 860805 1 001	NIK. 201607 891121 1 001

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, 11 Agustus 2017

Gabreil Arganata

NIM: 135150201111272

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa yang telah melimpahkan rahmat, taufik serta hidayah-Nya sehingga laporan skripsi yang berjudul "Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor" ini dapat terselesaikan.

Penulis menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak yang telah memberikan bantuan baik lahir maupun batin selama penulisan skripsi ini. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Bapak Eko Sakti P.,S.Kom.,M.Kom dan bapak Widhi Yahya, S.Kom., M.Sc selaku dosen pembimbing skripsi penulis yang dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D., selaku Ketua Jurusan Teknik Informatika
3. Bapak Agus Wahyu Widodo, S.T, M.Cs, selaku Ketua Program Studi Teknik Informatika
4. Kedua orang tua dan seluruh keluarga besar atas segala nasehat, kasih sayang dan kesabaran dalam membesarkan dan mendidik penulis, serta yang senantiasa tiada henti-hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini.
5. Putri Laras Rinjani yang selalu bersedia memberikan bantuan, saran, dan semangat selama proses penyelesaian skripsi ini.
6. Teman-teman terdekat Hudan Abdur Rohman, Niki Yuniar, Muhammad Fauzi, Ferdy Wahyu Rianto, Satrio Dimas Bagaskara, Edwar Budiman, Hanif Kuncahyo, Rizky Kharisma.
7. Teman, sekaligus sahabat "Tetap di Hati", Zam, Abi, Bayu, Niki, yang selalu menjadi penyemangat selama menjalani perkuliahan dan penyelesaian skripsi.
8. Teman-teman "TEAM KRS" yang selalu Bersama dan memberi dukungan selama perkuliahan.
9. Keluarga BEM Bersatu 2 dan Bersatu 3 yang sudah memberikan banyak pengalaman selama masa perkuliahan.
10. Keluarga PSDM Beratu 2, dan "FUNFAMS" yang sudah memberikan banyak pengalaman, cerita selama masa perkuliahan.
11. Teman-teman Teknik Informatika angkatan 2013 yang selalu mendukung dan berbagi ilmu dari awal perkuliahan sampai tahap akhir penyelesaian skripsi
12. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penyelesaian skripsi ini.
13. Semua pihak yang telah membantu dan berbagi ilmu dalam penyelesaian skripsi ini yang tidak dapat penulis sebutkan satu per satu.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat membawa manfaat bagi semua pihak yang menggunakannya.

Malang, 10 Agustus 2017

Gabreil Arganata
gabreilarganata13@gmail.com

ABSTRAK

Internet of things memegang peranan penting dalam perkembangan internet saat ini. Implementasi dari IoT menghasilkan berbagai data yang heterogen dari sensor, dan akan berkembang semakin besar. Contoh dari heterogen adalah data temperature, kelembapan, dan gambar yang berupa file. Hal tersebut menjadi kendala dalam pemilihan metode pada media penyimpanan. Dari permasalahan ini, solusi yang paling mungkin diterapkan adalah penerapan metode NoSQL. Oleh karena itu, pada penelitian ini diusulkan sebuah media penyimpanan berbasis MongoDB dan GridFS yang merupakan database NoSQL untuk menjawab tantangan tersebut. Penelitian ini juga mengusulkan sebuah *Internet Gateway Device* untuk menghubungkan *middleware* yang telah ada dengan pusat data. Solusi tersebut dibungkus dalam sebuah *framework* yang didalamnya terdapat sebuah *web service* untuk memudahkan proses *request* dan *response*. Pengujian kinerja sistem dilakukan dari segi fungsional, skalabilitas, serta *response time* penyimpanan dan pengambilan data. Hasil dari pengujian fungsional didapatkan bahwa sistem penyimpanan data yang dikembangkan sudah berjalan sesuai dengan fungsinya dalam menyimpan beragam data ke dalam MongoDB dan GridFS. Pada pengujian skalabilitas *web service*, menghasilkan rata-rata 173,66 *request* fungsi GET, dan 433,33 untuk *request* fungsi POST. Aspek lain yang diuji ialah *response time* dalam menampilkan data pada IoT Apps yang diambil dari *data storage*, untuk pengujian sebanyak 50 data menghasilkan *response time* 0,186 detik untuk MongoDB dan 0,185 untuk GridFS. Berdasarkan hasil tersebut, sistem ini dapat menjadi solusi dari permasalahan penyimpanan data IoT.

Kata kunci: *Internet of things*, MongoDB, GridFS, *data storage*, data sensor

ABSTRACT

Internet of things (IoT) plays an important role in the development of the Internet today. The implementation, which resulted into IoT's heterogen data from variety of sensors, and it will keep getting bigger everyday. The examples of heterogen data are temperature, humidity, and image file. Along with that, problem of choosing which method to store the data into data storage appear. Based on that problem, the solution that's possible to be applied is NoSQL method. Therefore, in this research a media storing system using MongoDB and GridFS (NoSQL database) based are proposed to answer those challenges. Furthermore, an Internet Gateway Device is proposed to connect an existing middleware with data center. The solution are wrapped inside a framework which also include a web service to ease the request and response process. The system performance test has been done in few aspect such as functional, scalability, and also response time of saving and retrieving data. The result of the functional test is that data storage system, which has been developed, is running well based on its function in storing variety of data into data storage. Upon the scalability of web service test, the results are 173,66 as average of GET request and for the POST request the average is 433,33. Another aspect that also being tested is response time in showing the data in IoT Apps which retrieved from GridFS, the test has been done for 50 data and it can be done in 0,186 second for MongoDB and 0,185 for GridFS. Based on those results, this system can become a solution for IoT data storage problem.

***Keywords**: Internet of things, MongoDB, GridFS, data storage, sensor data*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Internet of Things (IoT) telah berkembang besar dan semakin mempengaruhi berbagai bidang dalam kehidupan. IoT mengacu kepada suatu objek unik yang diidentifikasikan dan direpresentasikan secara *virtual* di internet. IoT dapat dijelaskan sebagai 1 set *things* yang saling terkoneksi melalui internet. *Things* dapat berupa *tags*, sensor, peralatan rumah tangga, dan lain sebagainya. IoT berfungsi mengumpulkan data dan informasi dari lingkungan fisik (*environment*), yang kemudian data tersebut akan diproses sesuai kepentingan dan kebutuhan di setiap bidangnya. Konsep IoT mengacu pada 3 elemen utama, yaitu barang fisik yang dilengkapi modul IoT, perangkat koneksi ke internet seperti modem dan router, dan *cloud data center* tempat untuk menyimpan aplikasi beserta *data storage*.

Penerapan dalam IoT sudah banyak dilakukan dengan berbagai macam sensor dengan kebutuhan yang berdeda-beda di setiap bidangnya. Contohnya pada penelitian sebelumnya yang berjudul "Pengembangan IoT *middleware* berbasis *event-based* dengan protokol komunikasi CoAP MQTT dan Websocket", telah dikembangkan sebuah lingkungan IoT yang terdiri dari node sensor (sensor suhu dan kelembapan) dan *middleware* dengan mekanisme *publish subscribe*. Middleware dikembangkan untuk mengatasi masalah interoperabilitas agar dapat menghubungan perangkat yang pada dasarnya menggunakan protokol berbeda yaitu protokol CoAP MQTT dan *Websocket*. Middleware tersebut mampu mendukung interoperabilitas dengan menyediakan *gateway* multi-protokol untuk CoAP, MQTT, dan *Websocket* (Anwari, 2017). Middleware yang telah dikembangkan tersebut mampu menerima data dari berbagai jenis sensor, dalam hal ini digunakan redis sebagai database dan *message broker*. Oleh karena itu, diperlukan sebuah media penyimpanan untuk menampung data dari *middleware*, karena middleware hanya berfungsi sebagai media penyimpanan sementara dan terbatas. Harapannya didapat sebuah sistem IoT yang utuh dari node sensor sampai ke pusat data untuk dilakukan tahap berikutnya yaitu analisis data.

Terdapat tantangan dalam membangun media penyimpanan data sensor, yaitu volume data yang besar, dan data yang dihasilkan beragam. Karena kebutuhan data yang semakin meningkat, IoT membutuhkan solusi penyimpanan data yang tidak hanya mampu menyimpan data besar secara efisien, namun juga mendukung skala horizontal(peningkatan kapasitas penyimpanan data). Selain itu, data IoT dapat dikumpulkan dari berbagai sumber yang terdiri dari berbagai data terstruktur dan tidak terstruktur. Melihat tantangan tersebut, dibutuhkan sebuah platform penyimpanan data dengan kemampuan menyimpan dan mengelola data IoT yang terstruktur dan tidak terstruktur secara efisien (Jiang & Xu, 2014). SQL dan NoSQL merupakan teknologi yang banyak diterapkan untuk media penyimpanan. Pada SQL saat data semakin besar, *data storage* relasional akan membutuhkan sumber daya yang sangat besar dalam mempertahankan

performanya. Hal ini berakibat langsung ke membengkaknya biaya operasional dan infrastruktur yang dibutuhkan. Semakin banyak benda fisik (sensor) yang digunakan, maka volume data yang dihasilkan akan semakin membesar, format atau struktur data yang disimpan pun semakin beragam. Selain itu data IoT yang dikumpulkan terdiri dari data terstruktur dan tidak terstruktur (Atzori, Iera, & Morabito, 2010). Karena *data storage* relasional memiliki struktur yang tetap (*fixed*), maka tidak dapat menjadi solusi yang tepat untuk mengakomodasi permasalahan tersebut.

Untuk mengatasi permasalahan volume data dan heterogenitas tipe data, Pada penelitian yang berjudul "*A Storage Solution for Massive IoT Data Based on NoSQL*", merancang menajemen penyimpanan yang disebut IOTMDB yang berdasarkan NoSQL yang menjadi solusi penyimpanan data IoT yang besar dan heterogen. Sistem IOTMDB dibagi menjadi 4 bagian utama, yaitu *master node* yang menjadi manajer dari setiap cluster, *stanby node* berfungsi menjadi pengganti saat terjadi error pada *master node*, *data reception node* berfungsi sebagai penerima data sensor, dan *slave node* untuk menyimpan seluruh data (Tingli, Yang, Ye, Shuo, & Wei, 2012). Penelitian lain yang berjudul "*An IoT-Oriented Data Storage Framework in Cloud Computing Platform",* dirancang sebuah *data storage* yang terdiri dari Hadoop Distributed File System (HDFS) untuk menyimpan data tidak terstruktur, dan *data storage* relasional yaitu SQL digabung dengan *data storage* non relasional yaitu MongoDB (Jiang & Xu, 2014). penelitian ini menyediakan berbagai subsistem yang bekerja untuk menangani masing-masing tipe data. Dari referensi tersebut untuk menjawab tantangan volume dan data pendekatan nosql lebih diunggulkan.

Di antara database NoSQL ini, MongoDB cukup populer karena memiliki banyak dukungan dari berbagai framework web. Semantik data pada MongoDB menggunakan JSON yang mudah dimengerti. MongoDB juga mempunyai model data yang fleksibel. Hal ini sesuai dengan representasi data pada penelitian sebelumnya yang menggunakan format JSON dalam pengiriman data dari node sensor ke middleware. Selain itu NoSQL sangat sesuai untuk menampung sejumlah besar data log yang tidak memiliki hubungan dan struktur beragam dan rumit. Pada MongoDB juga terdapat teknologi GridFS yang merupakan spesifikasi dari MongoDB dimana dapat mengatasi permasalahan IoT data yang besar, format dan tipe data yang berbeda. Bahkan untuk menyimpan file multimedia berukuran besar misal gambar atau video, GridFS membagi data menjadi dua koleksi. Koleksi file yang menyimpan metadata file, dan koleksi potongan (*chunks*) yang menyimpan data biner sebenarnya.

Dari pembahasan sebelumnya, maka sistem media penyimpanan akan dibangun dengan MongoDB dan GridFS. Selain media penyimpanan dengan NoSQL, pada penelitian ini menyediakan fitur pengelolaan data yang mampu menjawab permasalahan format data, tipe data yang besar dan beragam, serta mampu berkomunikasi dengan *middleware* pada penelitian sebelumnya dan aplikasi lain (IoT *Apps*).

1.2 Rumusan Masalah

1. Bagaimana membangun sistem *data storage* untuk menyimpan data sensor yang heterogen?
2. Bagaimana kinerja sistem *data storage* dari segi fungsional dan non fungsional?

1.3 Tujuan

Tujuan dari penelitian ini:

1. Dapat mengembangkan sistem *data storage* untuk menyimpan data sensor yang heterogen dengan MongoDB GridFS
2. Dapat mengetahui kinerja sistem *data storage* dari segi fungsional dan non fungsional

1.4 Manfaat

Manfaat dari penelitian ini adalah untuk membangun *media penyimpanan* yang dapat menyimpan data sensor yang heterogen dengan MongoDB GridFS dan mempunyai fungsi pengelolaan data.

1.5 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih fokus, maka penelitian tugas akhir ini dibatasi dalam hal:

1. Dalam pembangunan *data storage* ini, *data storage* yang digunakan adalah MongoDB
2. Pengujian berfokus pada penyimpanan data sensor
3. Lingkungan pengujian menggunakan jaringan LAN pada FILKOM
4. Sensor yang digunakan adalah DHT11, sensor CO2, dan sensor kamera

1.6 Sistematika Pembahasan

Penulisan sistematika pembahasan dan penyusunan laporan penelitian dapat diuraikan sebagai berikut:

BAB I PENDAHULUAN

Pada Bab I Pendahuluan dijelaskan mengenai latar belakang, rumusan masalah, tujuan, manfaat penelitian, batasan penelitian dan sistematika pembahasan dari "Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor".

BAB II LANDASAN KEPUSTAKAAN

Pada bab ini berisi beberapa dasar teori dan referensi penelitian yang pernah ada yang memiliki tujuan dan perancangan yang hampir sama dengan penelitian ini.

BAB III METODOLOGI

Pada bab ini membahas beberapa langkah kerja yang akan dilakukan dalam penelitian, dianataranya studi literatur, analisis kebutuhan sistem, implementasi dan pengujian.

BAB IV ANALISIS KEBUTUHAN DAN PERANCANGAN

Pada bab ini membahas perancangan, analisis kebutuhan dan pengujian *data storage* yang menunjukkan bahwa sistem dapat diimplementasikan dengan baik.

BAB V IMPLEMENTASI

Bab ini membahas implementasi *Internet Gateway Device*, *web service*, *data storage* MongoDB GridFS, IoT Apps, dan lingkungan sistem sehingga dapat menunjukkan bahwa sistem dapat diimplementasikan dengan baik.

BAB VI PENGUJIAN DAN ANALISIS

Pada bab pengujian dan analisis ditunjukkan hasil pengujian dari perancangan sistem dan aplikasi serta dijelaskan pula teknik pengujian yang dilakukan sehingga dapat menunjukkan bahwa sistem dapat diimplementasikan dengan baik.

BAB VII KESIMPULAN DAN SARAN

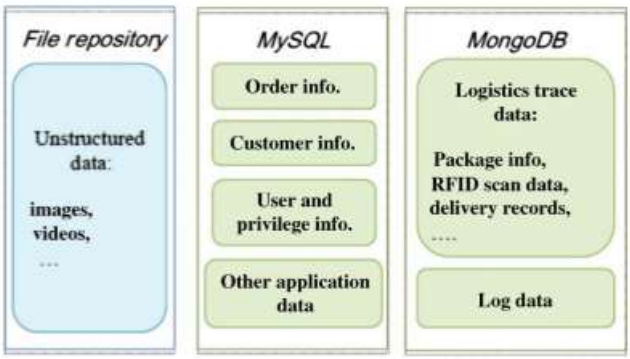
Bab ini berisi kesimpulan dari pembahasan rumusan masalah yang ada berdasarkan analisis dan pengujian serta berisi saran-saran dan pengembangan yang dapat dilakukan dari "Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor".

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Tinjauan Pustaka

Internet of Things (IOT) telah memberikan peluang yang menjanjikan untuk membangun sistem industri yang kuat dan aplikasi dengan memanfaatkan perangkat sensor nirkabel. Manfaat dari teknologi jaringan sensor, benda-benda fisik umum dapat terhubung, dan dapat dipantau dan dikelola oleh satu sistem.

Dalam mengelola benda-benda fisik yang terlibat pada sistem IOT dan perangkat yang digunakan untuk memantau objek, mengumpulkan dan data transit, kita menghadapi serangkaian tantangan. Berbagai macam sensor, pembaca RFID, dan perangkat lain yang terlibat dalam sistem IOT dapat menghasilkan data dengan cepat sehingga data harus diproses dengan throughput yang tinggi. Selanjutnya, karena volume data yang sangat besar dan dapat meningkat dengan cepat, solusi penyimpanan data untuk data IOT tidak hanya harus mampu menyimpan data yang besar secara efisien tetapi juga mendukung skala horisontal. Selain itu, data IOT dapat dikumpulkan dari berbagai sumber dan terdiri dari berbagai data terstruktur dan tidak terstruktur. Komponen penyimpanan data diharapkan memiliki kemampuan untuk menangani sumber daya terhadap data yang heterogen (Jiang & Xu, 2014).



Gambar 2.1 Distribusi data dalam framework penyimpanan data (Jiang & Xu, 2014)

Kerangka penyimpanan data terdiri dari HDFS (Hadoop Distributed File System) untuk menyimpan file repository dimana file yang tidak terstruktur, kemudian terdapat basis data NoSQL dimana dalam kasus ini menggunakan MongoDB dan basis data relasional menggunakan MySQL untuk menyimpan data repository dimana file yang terstruktur (Jiang & Xu, 2014).

Pada penelitian sebelumnya yang berjudul "Pengembangan IoT *middleware* berbasis *event-based* dengan protokol komunikasi CoAP MQTT dan *Websocket*" dikembangkan *middleware* yang mampu mengatasi masalah interoperabilitas pada IoT. Permasalahan interoperabilitas yang diangkat yakni mengacu pada perkembangan *middleware* agar dapat menghubungan perangkat yang pada dasarnya menggunakan protokol yang berbeda yaitu protokol CoAP MQTT dan *Websocket*. *Middleware* tersebut mampu mendukung interoperabilitas dengan

menyediakan gateway multi-protokol untuk CoAP, MQTT, dan *Websocket* (Anwari, 2017).

Pada penelitian yang berjudul "*A Storage Solution for Massive IoT Data Based on NoSQL*", merancang menajemen penyimpanan yang disebut IOTMDB yang berdasarkan NoSQL yang menjadi solusi penyimpanan data IoT yang besar dan heterogen. Sistem IOTMDB dibagi menjadi 4 bagian utama, yaitu *master node* yang menjadi manajer dari setiap cluster, *stanby node* berfungsi menjadi pengganti saat terjadi error pada *master node*, *data reception node* berfungsi sebagai penerima data sensor, dan *slave node* untuk menyimpan seluruh data (Tingli, Yang, Ye, Shuo, & Wei, 2012). (Kende, et al., 2015)

Pada penelitian yang dilakukan (Jiang & Xu, 2014), memakai penyimpanan data yang memiliki kemampuan menangani sumber daya terhadap data yang heterogen. Akan tetapi mereka merancang sistem *data storage* yang terdiri dari Hadoop Distributed File System (HDFS) untuk menyimpan data tidak terstruktur, dan *data storage* relasional yaitu MQTT digabung dengan *data storage* non relasional yaitu MongoDB dalam mengatasi masalah tersebut. Pada penelitian kedua yang dilakukan oleh (Anwari, 2017), melibatkan data fisik berupa sensor DHT11/22, dan middleware dengan protokol MQTT, QoAP, dan aplikasi web sebagai penyimpan data melalui protokol *websocket*, masih belum menggunakan penyimpanan data berupa *data storage* untuk menyimpan data sensor tersebut. Oleh karena itu penelitian ini membangun *data storage* yang berupa MongoDB GridFS untuk menyimpan data sensor yang beragam, kemudian juga akan dibangun sebuah *Internet Gateway Device* yang berfungsi untuk menerima data dari middleware (mikrokontroller) dan diteruskan ke data *center* sebagai penyimpanan datanya.

2.2 Dasar Teori

Berdasarkan beberapa informasi dari beberapa kajian pustaka, maka dalam penulisan penelitian ini terdapat beberapa dasar teori, antara lain:

2.2.1 *Internet of Things*

Menurut analisa McKinsey Global Institute, *Internet of Things* adalah sebuah teknologi yang memungkinkan kita untuk menghubungkan mesin . peralatan, dan benda fisik lainnya dengan sensor jaringan dan aktuator untuk memperoleh data dan mengelola kinerjanya sendiri, sehingga memungkinkan mesin untuk berkolaborasi dan bahkan bertindak berdasarkan informasi baru yang diperoleh secara independen. *Internet of Things* adalah interkoneksi yang unik antara embedded computing devices dalam infrastruktur internet yang ada. Sebuah publikasi mengenai *Internet of Things* in 2020 menjelaskan bahwa *Internet of Things* adalah suatu keadaan ketika menda memiliki identitas, bisa beroperasi secara intelijen, dan bisa berkomunikasi dengan sosial, lingkungan, dan pengguna (Atzori, Iera, & Morabito, 2010). Dengan demikian, dapat kita simpulkan bahwa *Internet of Things* membuat kita membuat suatu koneksi antara mesin dengan mesin, sehingga mesin-mesin tersebut dapat berinteraksi dan bekerja secara independen sesuai dengan data yang diperoleh dan diolahnya

secara mandiri. Tujuannya adalah untuk membuat manusia berinteraksi dengan benda dengan lebih mudah, bahkan supaya benda juga bisa berkomunikasi dengan benda lainnya.

Cara kerja dari *Internet of Things* cukup mudah. Setiap benda harus memiliki sebuah IP Address. IP Address adalah sebuah identitas dalam jaringan yang membuat benda tersebut bisa diperintahkan dari benda lain dalam jaringan yang sama. Selanjutnya, IP address dalam benda-benda tersebut akan dikoneksikan ke jaringan internet.

2.2.2 *Internet of Things Middleware*

Untuk mempermudah *things* dalam berkomunikasi dengan internet, IoT membutuhkan sebuah *middleware* yakni *software* atau *hardware* yang menyediakan *interface* bagi *things* untuk mengirimkan atau mendapatkan data. Dalam IoT, penggunaan *middleware* sangat beragam misalkan, *middleware* untuk menghubungkan sensor dengan *Cloud*, *middleware* untuk *smart-device* terhubung dengan *smart-phone* ataupun untuk komunikasi *machine-to-machine* (M2M). Pada umumnya *middleware* memiliki *interface* bagi sensor untuk mengirimkan data dan juga *interface* bagi aplikasi untuk membaca data tersebut. Hal yang membedakan antara *middleware* satu dengan *middleware* lainnya yakni bagaimana ia menerapkan *interface* tersebut, melakukan manajemen data, serta menjaga keamanan dan privasi data tersebut.

Dengan kondisi IoT saat ini, *middleware* yang dikembangkan harus dapat beradaptasi dan menyelesaikan tantangan yang ada. Beberapa penelitian telah dilakukan untuk merumuskan tantangan tersebut dan bagaimana menyelesaikannya. Penelitian ini meliputi survei dan analisis tentang *middleware* yang sudah saat ini, pola-pola perancangan serta konsep arsitektur *middleware* (Ngu & Gutierrez, 2016)

2.2.3 *Cloud Computing*

Cloud Computing. *Cloud* computing adalah sebuah model komputasi yang terkonfigurasi, dimana sumber daya seperti *processor*/*computing power*, *storage*, *network*, dan *sofware* menjadi abstrak (virtual) serta diberikan sebagai layanan di jaringan/internet menggunakan pola akses remote (Purbo, 2011). NIST (National Institute of Standarts and Technology) sebagai badan nasional standar dan teknologi Amerika Serikat memberikan definisi *cloud* computing yaitu suatu model untuk memberikan kenyamanan, on-demand akses jaringan untuk memanfaatkan bersama suatu sumber daya komputasi yang terkonfigurasi (misalnya, jaringan, server, penyimpanan, aplikasi, dan layanan) yang dapat secara cepat diberikan dan dirilis dengan upaya manajemen yang minimal atau interaksi penyedia layanan. *Cloud* computing menyediakan 3 model layanan yaitu (Mell & Grance, 2011):

a) *Software as a Service* (SaaS)
b) *Platform as a Service* (PaaS)
c) *Infrastructure as a Service* (IaaS) (Purbo, 2011)

Sedangkan model penyebaran *cloud computing* menurut NIST (National Institute of Standarts and Technology) terdiri dari empat model, yaitu (Mell & Grance, 2011):

a) *Private Cloud*

b) *Community Cloud*

c) *Public Cloud*

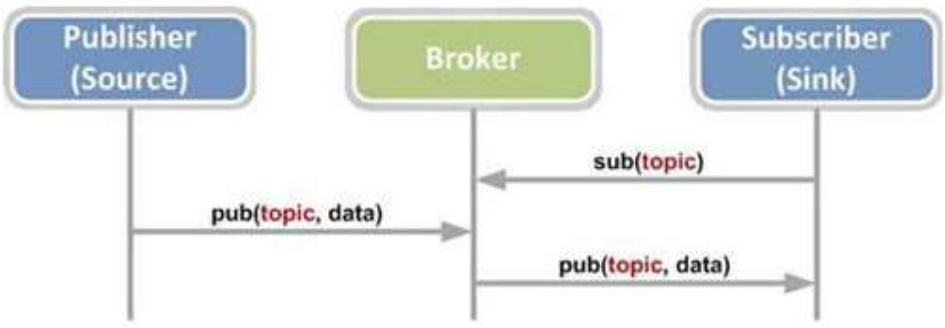
d) *Hybrid Cloud*

2.2.4 *Internet Gateway Device* (IGD)

Internet Gateway Device adalah sebuah perangkat yang dipakai untuk menghubungkan satu jaringan komputer dengan satu ataupun lebih jaringan komputer yang memakai protokol komunikasi yang berbeda sehingga informasi dari satu jaringan komputer bisa diberikan kepada jaringan komputer lain yang protokolnya tidak sama atau berbeda. Sama halnya dengan sebuah gateway jaringan, yang berfungsi sebagai sistem internetworking yang menghubungkan dua jaringan bersama-sama dan bisa dikonfigurasi dalam aplikasi perangkat lunak, perangkat keras, ataupun keduanya (Kende, et al., 2015).

2.2.5 *Message Queue Telemetry Transport* (MQTT) *Publish/Subscribe*

*Message Queue Telemetry Transport (*MQTT*)* adalah sebuah protokol komunikasi data *machine to machine* (M2M) yang berada pada layer aplikasi. *MQTT* bersifat lightweight message artinya MQTT berkomunikasi dengan mengirimkan data pesan yang memiliki header berukuran kecil yaitu hanya sebesar 2bytes untuk setiap jenis data, sehingga dapat bekerja di dalam lingkungan yang terbatas sumberdayanya seperti kecilnya bandwidth dan terbatasnya sumberdaya listrik. Protokol MQTT juga menjamin terkiriminya semua pesan walaupun koneksi terputus sementara. Protokol MQTT menggunakan metode *publish*/*subscribe* untuk metode komunikasinya, cara kerja protokol MQTT dapat dilihat seperti pada gambar 2.3 beriikut:



Gambar 2.2 Cara Kerta MQTT

[Sumber tessel.io]

Publish/*subscribe* sendiri adalah sebuah pola pertukaran pesan di dalam komunikasi jaringan dimana pengirim data disebut *publisher* dan penerima data disebut dengan *Internet Gateway Device*. Metode *publish*/*subscribe* adalah salah satu tipe komunikasi *indirect* atau tidak langsung yang memiliki beberapa kelebihan salah satunya yaitu *loose coupling* atau *decouple* dimana berarti antara produsen data dan konsumen data tidak saling terhubung secara langsung. Terdapat 3 buah *decoupling* yaitu *time decoupling*, *space decoupling* dan *synchronization decoupling*. *Time decoupling* adalah sebuah kondisi dimana *publisher* dan *Internet Gateway Device* tidak harus saling aktif pada waktu yang sama. *Space decoupling* adalah dimana *publisher* dan *Internet Gateway Device* aktif di waktu yang sama akan tetapi antara *publisher* dan *Internet Gateway Device* tidak saling mengetahui keberadaan dan identitas satu sama lain. Dan yang terakhir adalah *synchronization decoupling* kondisi dimana pengaturan event baik itu penerimaan atau pengiriman pesan di sebuah node hingga tidak saling mengganggu satu sama lain (Adi, et al., 2016). Dengan kelebihan – kelebihan tersebut jika terjadi kerusakan disalah satu node yang terhubung dalam jaringan menggunakan *MQTT* tidak akan terlalu berpengaruh pada node – node lainnya sehingga jika terdapat kerukasan pada satu node *publisher* atau *Internet Gateway Device*, node – node yang lain akan tetap berfungsi sehingga memudahkan untuk melakukan pengcekan jika terjadi kerusakan.

Pengiriman data pada *MQTT* didasari oleh topik, topik ini nantinya yang akan menentukan pesan dari *publisher* harus dikirim pada *Internet Gateway Device* yang mana. Topik ini dapat bersifat hirarki, dimana *MQTT* topik memiliki tipe data string dan untuk perbedaan hirarki atau level dari topik digunakan tanda baca "/", pada *MQTT* topic dikenal 2 buah *wildcard* karakter untuk subscription topik yaitu "+" dan "#", penggunaan *wildcard* karakter "+" adalah sebuah *single level wildcard* yang digunakan untuk mencocokkan topik yang di *Internet Gateway Device* dengan apapun, contohnya seperti rumah /+/ cahaya, adalah sama dengan rumah/taman/cahaya, atau rumah/kamar/cahaya, tetapi tidak sama dengan rumah/taman, karena karakter "+" adalah single wildcard karakter yang berati mencocokkan topik dengan semuanya pada level tertentu, sedangkan karakter "#" adalah *multi-level wildcard* karakter dimana karakter ini akan mencocokkan topik pada setiap level karakter ini ada dan setelahnya, contoh rumah/# berarti sama dengan rumah/taman, atau rumah/taman/cahaya atau rumah/cahaya (Solace, n.d.).

Broker merupakan komponen yang paling penting di dalam arsitektur *publish*/subscribe dan *MQTT*. Broker adalah sebagai perantara pertukaran pesan antara *publisher* dan *Internet Gateway Device*. Masing - masing dari *publisher* dan *Internet Gateway Device* harus terkoneksi ke broker untuk mengirimkan ataupun menerima pesan, broker akan menerima seluruh pesan yang di *publish* oleh *publisher*, untuk selanjutnya diteruskan kepada *Internet Gateway Device* berdasarkan dengan topik yang sesuai dengan pesan yang dikirim dan topik yang di *subscribe* oleh *Internet Gateway Device* , ketika *Internet Gateway Device* tidak aktif maka broker akan menyimpan pesan pada *buffer* untuk selanjutnya dikirimkan ketika *Internet Gateway Device* telah aktif kembali.

MQTT memiliki 3 level *quality of service* (QoS) dalam pengiriman pesannya yaitu 0,1,2. Pada level 0 atau disebut dengan "*at most once delivery message*" QoS level 0 ini adalah QoS paling cepat untuk mengirim pesan, QoS ini akan menjamin mencoba mengirim dengan usaha terbaiknya, pesan akan langsung dikirimkan tanpa menunggu *acknowledge* dari sisi penerima. Namun pengiriman dengan menggunakan QoS level 0 memungkinkan hilangnya pesan jika penerima terputus secara tiba – tiba. QoS level 1 atau disebut juga "*at least once*" dengan opsi level ini *MQTT* akan mengirimkan pesan minimal sekali. Pesan yang dikirimkan nantinya akan di acknowledge oleh penerima, kemudian *MQTT* akan mengirimkan kembali pesan yang menggunakan QoS level 1 jika pengirim tidak menerima acknowledge dari penerima. Pesan yang dikirimkan menggunakan QoS level 1 juga akan di simpan oleh broker pada *data storage*. Untuk QoS level 2 atau "*exactly once delivery*" QoS ini akan menjami pesan dikirim dan diterima pada sisi penerima. QoS ini adalah yang paling aman namun juga paling lambat dibandingkan dengan 2 level QoS lainnya, jaminan pesan akan sampai dikarenakan adanya komunikasi 2 arah dari pengirim dan penerima (Lampkin, et al., 2012).

2.2.6 *Web Service*

Dalam Microsoft (2000) dinyatakan bahwa *web service* merupakan tahapan ketiga dari tahapan evolusi ASP (*Application Service Provider*) dimana pada tahapan pertama ditekankan pada penyediaan aplikasi *desktop* sedangkan pada tahapan kedua ditekankan pada penyediaan aplikasi berbasis *client-server*. Pada tahapan ketiga ini, komponen-komponen atau *building blocks software* disediakan sebagai *service* dan disebarkan lewat jaringan internet untuk diintegrasikan dengan aplikasi-aplikasi lain.

Menurut Kreger (2001) *web service* diartikan sebagai sebuah antar muka (*interface*) yang menggambarkan sekumpulan operasi-operasi yang dapat diakses melalui jaringan, misalnya internet, dalam bentuk pesan XML. Sedangkan menurut Manes (2001), *web service* diartikan sebagai sepotong atau sebagian informasi atau proses yang dapat diakses oleh siapa saja, kapan saja dengan menggunakan piranti apa saja, tidak terikat dengan sistem operasi atau bahasa pemrograman yang digunakan.

Web service dapat dibangun dengan menggunakan bahasa pemrograman apa saja dan juga dapat diimplementasikan pada *platform* manapun. Kita dapat membangun *web service* pada Windows 2000 dan menjalankannya melalui Windows, Linux, Unix, Mac, PalmOS dan WinCE (Hamids, 2000). Hal ini dimungkinkan karena *web service* berkomunikasi menggunakan sebuah standar format data yang *universal* yaitu XML dan menggunakan protokol SOAP. Karena *web service* menggunakan format data XML, maka *web service* juga mewariskan sifat *multi-tier* dari XML sehingga memungkinkan terjadinya integrasi antar *web service* atau aplikasi (Microsoft, 2001).

Menurut Meiyanto (2001) pada sistem *multi-tier*, aplikasi maupun dokumen XML dapat dilewatkan ke pihak lain dan diolah oleh pihak tersebut. Dalam sistem ini dimungkinkan suatu aplikasi dapat mengambil data dari satu sumber tanpa

harus tahu bahwa sebenarnya data tersebut dihasilkan melalui proses pengolahan oleh sistem lain sehingga dapat terjadi integrasi data maupun aplikasi yang sering disebut dengan A2A (*application to application*).

Dalam Kreger (2001) dikatakan bahwa model dari sebuah *web service* didasarkan pada interaksi antara 3 komponen yang berperan dalam *web service*, yaitu: *service provider, service registry* dan *service requestor/consumer*. Interaksi yang terjadi antara ketiga komponen tersebut juga melibatkan operasi *publish*, *find* dan *bind*. *Service provider* menyediakan *service* yang dapat diakses melalui jaringan komputer, misalnya internet. Kemudian, *service provider* mendeskripsikan *service* yang dibangun dan mem-*publish*-kan *service description* tersebut ke *service registry* atau secara langsung ke *service consumer*. *Service requestor/consumer* menggunakan operasi *find* untuk mendapatkan *service description* secara local maupun melalui *service registry*. *Service description* yang diperoleh itu kemudian digunakan untuk men-*bind service provider* dan berinteraksi dengan implementasi *web service* yang akan digunakan tersebut.

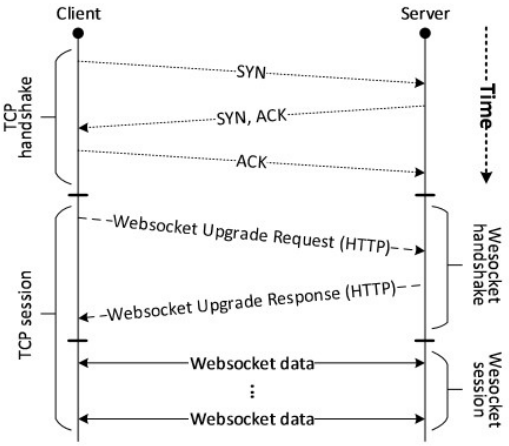
2.2.7 *WebSocket*

Websocket merupakan protokol pada layer aplikasi yang dibangun di atas koneksi TCP. Hal ini memungkinkan komunikasi secara dua arah pada web. *Websocket* merupakan alternatif dari teknologi sebelumnya yaitu HTTP polling dengan membawa peningkatan dalam kehandalan, kinerja, serta efisiensi. Dari uraian di atas maka *Websocket* sangat tepat digunakan untuk web dengan update data real-time seperti game, chat, *Internet of Things* (IoT) dan masih banyak lagi.

Meskipun memiliki fungsi yang hampir sama dan performanya tidak sebaik protokol TCP dalam komunikasi, namun *Websocket* mendukung teknologi sebelumnya (backward compatible) untuk infrastruktur web saat ini (Skvorc, Horvat, & Srbljic, 2014). *Websocket* handshake masih menggunakan standar HTTP 1.1, terlebih lagi komunikasi dengan *Websocket* mendukung fitur-fitur yang sudah ada seperti:

1. Dukungan penuh secara native pada web browser, termasuk SSL untuk keamanan.
2. Proxy dan firewall.
3. URL-based endpoint yang memungkinkan banyak atau bahkan layanan tidak terbatas pada satu TCP port.
4. Tidak ada batasan panjang data seperti yang ada pada protokol TCP.

Gambar di bawah ini menjelaskan sequence diagram dari webscoket protokol serta perbandingan dengan protokol TCP. Karena dibangun menggunakan TCP, maka *Websocket* juga membutuhkan handshake terlebih dahulu untuk membuat koneksi, bedanya pada TCP diperlukan 3-*way-handsake* sedangkan pada *Websocket* client mengirim upgrade request dan server membalasnya dengan upgrade respond. Dari sini maka terbentuklah koneksi antara server yang mendukung komunikasi secara asynchronous dan full-duplex. Gambar 2.2 menunjukkan proses komunikasi *websocket*.



Gambar 2.3 Proses komunikasi *Websocket* (Skvorc, Horvat, & Srbljic, 2014)

2.2.8 MongoDB GridFS

MongoDB adalah sebuah *database* yang bersifat *Open Source* yang memiliki *high performance*. MongoDB merupakan sebuah *database* dengan konsep manajemen *database* berorietasi dokumen yang dibuat menggunakan bahasa pemrograman C++. *Database* Berorientasi Dokumen adalah sebuah program komputer yang dirancang untuk menyimpan, mengambil dan mengelola data yang berorientasi dokumen. *database* berorientasi Dokumen adalah salah satu dari kategori *database* yang di kenal dengan istilah populer NoSQL. NoSQL singkatan dari Not Only SQL, artinya sebuah sistem basis data yang tidak harus menggunakan perintah SQL (*Structure Query Language*) untuk melakukan proses manipulasi data. MongoDB merupakan basis data yang tidak relasional, hal ini membuat MongoDB sangat cepat saat melakukan proses manipulasi data dari pada sistem basis data relasional (RDBMS), selain itu MongoDB berbasis dokumen sehingga tidak memiliki struktur yang teratur seperti tabel. Kelebihan MongoDB dibandingkan *database* yang lain adalah dapat melakukan searching lebih cepat, tidak perlu membuat struktur tabel karena MongoDB otomatis membuatkannya, jadi hanya perlu melakukan insert saja, mempercepat proses CRUD (*create, read, update, delete*), digunakan oleh banyak *website-website* besar (Chodorow & Dirolf, 2010).

Menurut Seguin (2012,p4), MongoDB adalah *data storage* dimana tiap tabel tidak memiliki relasi dengan tabel lainnya. MongoDB berisi *collection*. Setiap *collection* terdiri dari *documents*. Setiap *documents* terdiri dari *fields*. Sebuah *collections* dapat di *indexes*, yang meningkatkan kinerja pengurutan data.
Ada 6 konsep yang harus di mengerti adalah :

1. MongoDB memiliki konsep yang sama dengan *data storage* lainnya seperti MySQL atau Oracle. MongoDB dapat tidak memiliki *data storage* atau lebih dari satu *data storage*, masing-masingnya bertindak sebagai "*high level containers*".
2. Sebuah *data storage* dapat tidak memiliki *collection* atau lebih dari satu collection. Sebuah *collection* memiliki banyak kesamaan dengan tabel tradisional

pada *data storage* seperti MySQL. Sebuah *collection* dan tabel tradisional dalam hal ini dapat di anggap sama.

3. Sebuah *data storage* dapat tidak memiliki *documents* atau lebih dari satu *documents*. Sebuah *documents* dapat dianggap sama dengan sebuah *row* pada tabel tradisional.
4. Sebuah *documents* terdiri dari satu atau lebih *fields* atau *columns*.
5. *Indexes* di MongoDB seperti *indexes* di *Relational Data storage Management System*.
6. *Cursors* pada MongoDB di gunakan untuk meminta atau memanggil data.

GridFS merupakan spesifikasi MongoDB untuk menyimpan dan mengambil file besar, yang merupakan jenis sistem file untuk menyimpan file namun data yang disimpan dalam koleksi MongoDB.

BAB 3 METODOLOGI

Pada bab ini dijelaskan langkah-langkah dan metode yang akan dilakukan dalam pengerjaan tugas akhir Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor. Langkah-langkah yang dilakukan dalam penelitian ini ditunjukkan pada Gambar 3.1.

Gambar 3.1 Flowchart metode penelitian

3.1 Studi Literatur

Pada penelitian ini dibutuhkan studi literatur yang bertujuan untuk mencari dasar-dasar teori dan kajian pustaka yang digunakan untuk menunjang penulisan skripsi yang dilakukan yaitu Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor.

Teori-teori pendukung tersebut diperoleh dari buku, jurnal, e-book dan dokumentasi project pada penelitian sebelumnya.

3.2 Analisis Kebutuhan

Pada tahap ini merupakan tahap menganalisa kebutuhan sistem yang akan digunakan. Kebutuhan yang diperlukan dalam penelitian ini terbagi menjadi dua yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak. Analisis kebutuhan sistem diperlukan agar dapat mengetahui hal-hal yang diperlukan dalam "Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data yang Beragam dari Node Sensor" untuk menghindari penggunaan sumberdaya yang tidak perlu, analisis kebutuhan juga digunakan untuk acuan dalam merancang sistem, sehingga perancangan nantinya dapat terbentuk secara sistematis dan terarah. Dalam melakukan analisis kebutuhan salah satu cara yang dapat digunakan adalah melalui kepustakaan yang telah ada, kepustakaan berlandaskan pada penelitian-penelitian sejenis yang telah dilakukan untuk mengetahui perangkat apa saja yang dapat digunakan dalam pengembangan sistem yang akan dibuat selanjutnya.

3.2.1 Kebutuhan Perangkat Keras

Perangkat keras yang dibutuhkan dalam implementasi sistem dalam penelitian ini yaitu perangkat keras pada penelitian sebelumnya berupa, Raspberry Pi sebagai media untuk menjalan *middleware* yang dikembangkan serta sebagai *access point* yang menghubungkan sensor dan Raspberry Pi. Dibutuhkan juga *microSD card* 8GB sebagai tempat sistem operasi *Raspbian Jessie* dan semua perangkat lunak yang dibutuhkan. Perangkat penunjang lain yang digunakan dalam penelitian ini adalah USB Adapter yang bertindak sebagai *wireless adapter*. Kemudian mikrokontroler ESP8266, modul DHT11/DHT22 sebagai sensor data suhu dan kelembapan, modul kamera sebagai sensor gambar dan modul CO sebagai data CO. Sensor ini bekerja menggunakan *power adapter* dengan output 12V/2.0A. Perangkat berikutnya yaitu sebuah server untuk menjalankan aplikasi web dan menyimpan data di basis data. Perangkat terakhir yakni sebuah laptop untuk mengakses aplikasi web yang menampilkan data dari *middleware*.

3.2.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan untuk mengembangkan penelitian ini berupa Windows 10 Pro untuk menjalankan middleware melalui remote. Putty, MobaXterm, VNC Viewer sebagai aplikasi untuk meremote dan menlajankan middleware. Robomongo sebagai manajemen *data storage* yang digunakan untuk menyimpan dan melihat hasil data sensor yang disimpan pada MongoDB dan GridFS.

3.3 Perancangan

Perancangan sistem dalam penelitian ini terdiri dari perancangan alur sistem, perancangan *data storage*, dan perancangan arsitektur jaringan untuk pengujian. Perancangan sistem dibuat dengan menggambarkan arsitektur sistem berdasarkan hasil analisis kebutuhan perangkat keras, kebutuhan perangkat

lunak, kebutuhan fungsional dan kebutuhan non-fungsional yang telah ditentukan.

Dalam sistem yang dikembangkan, terdapat dua data yang diterima oleh *Internet Gateway Device* yang nantinya akan disimpan ke dalam *data storage* GridFS. Data pertama adalah data dari sensor CO yang berupa data karbon monoksida. Data kedua adalah data yang berasal dari sensor kamera yang berupa data gambar. Pada bagian perancangan *data storage*, data akan disimpan pada *data storage* GridFS. Data yang berasal dari sensor DHT11/22 dan sensor kamera nantinya akan di subscribe kemudian *Internet Gateway Device* akan menyimpan data tersebut ke *data storage*.

Untuk mengetahui apakah sistem yang dibuat dapat bekerja sesuai dengan kebutuhan, maka dibutuhkan pengujian untuk membuktikannya. Dalam tahap ini terdapat dua jenis perancangan yakni perancangan topologi jaringan yang akan digunakan dan perancangan skenario pengujian. Pengujian dalam penelitian ini dibagi menjadi dua yakni *integration testing* dan *interoperability testing*. *Integration testing* digunakan untuk mengetahui apakah *middleware* yang dikembangkan sudah sesuai dengan kebutuhan fungsional sedangkan *interoperability testing* dilakukan untuk menentukan tingkat interoperabilitas *middleware* dan kinerja *middleware* dalam hal pengiriman pesan dan *delay*.

3.4 Implementasi

Implementasi dilakukan dengan menjalankan sistem pada penelitian sebelumnya yang dilakukan oleh (Anwari, 2017). Sistem tersebut akan menghasilkan data sensor suhu dan kelembapan, data karbon monoksida, dan data sensor gambar. Data sensor tersebut nantinya akan disimpan kedalam *data storage* MongoDB dan GridFS. Selanjutnya membangun sebuah *web service* untuk membaca dan mengakses basis data MongoDB dan GridFS dari luar jaringan system itu sendiri. Langkah berikutnya adalah mengimplementasikan topologi jaringan untuk proses pengujian. Implementasi ini meliputi pengaturan pada middleware, menghubungkan sensor serta menghubungkan laptop ke jaringan LAN.

3.5 Pengujian dan Analisis Hasil Pengujian

Pengujian dilakukan untuk mengetahui apakah sistem yang dikembangkan sudah sesuai dengan kebutuhan fungsional, skalabilitas sistem dan *response time*. Pada tahap ini dilakukan tiga jenis pengujian yakni pengujian fungsional, pengujian skalabilitas dan pengujian *response time* . Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang dikembangkan sudah memenuhi kebutuhan fungsional yang sudah ditentukan. Apabila semua kebutuhan fungsional terpenuhi maka dilanjutkan pengujian berikutnya. Namun apabila ada kebutuhan fungsional yang belum berjalan seusai harapan, maka dilakukan pengecekan kembali terhadap rancangan dan kode sistem kemudian melakukan pengujian sekali lagi, begitu seterusnya hingga semua kebutuhan terpenuhi. Pengujian skalabilitas dilakukan untuk mengetahui sistem yang dikembangkan dalam menangani peningkatan beban sistem dan *response time*

dilakukan untuk mengetahui waktu tanggap sistem ketika ada *request* yang masuk.

Data dari pengujian selanjutnya diolah dan dianalisis untuk mengetahui kinerja sistem dalam menyimpan data jika dihadapkan dengan beberapa kondisi jaringan. Sementara analisa hasil pengujian dilakukan untuk mengetahui apakah sistem yang dibuat telah sesuai dengan tujuan perancangan sistem tersebut. Sehingga kita bisa melihat hasil bagaimana data sensor yang disimpan.

3.6 Kesimpulan dan Saran

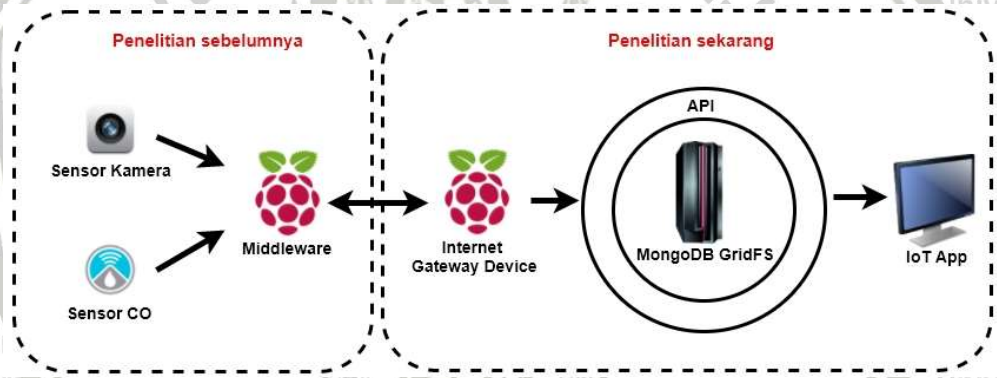
Pengambilan kesimpulan dan saran merupakan tahapan akhir dari proses penelitian, yang nantinya dapat disimpulkan mengenai bagaimana sistem yang telah dibuat, kesimpulan dapat diambil dari kelebihan serta kekurangan sistem yang telah dibuat, dan kesusaian sistem antara teori dan praktik, yang nantinya akan menjawab rumusan masalah yang telah dirumuskan sebelumnya. Saran dimaksud untuk memberikan masukan terhadap kekurangan – kekurangan yang ada di dalam sistem yang telah dibuat, yang nantinya dapat sebagai pertimbangan untuk penelitian selanjutnya.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

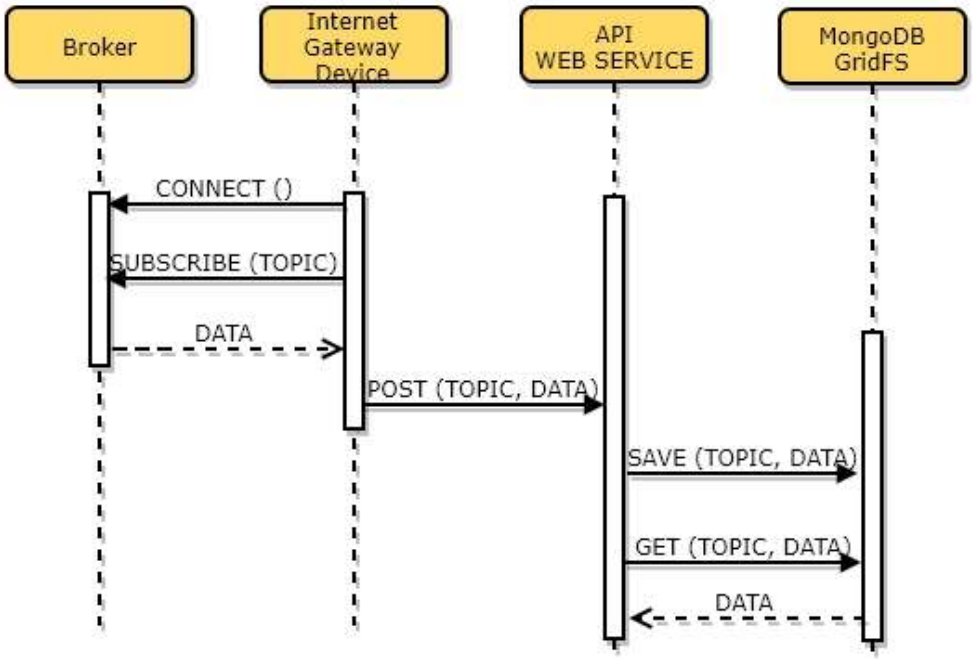
Analisis kebutuhan dan perancangan diperlukan agar dalam pengembangan sistem dapat berjalan dengan sistematis, analisis kebutuhan berisi mengenai perangkat-perangkat dan kebutuhan lainnya yang meliputi kebutuhan perangakat keras, dan perangkat lunak.

4.1 Deskripsi Umum Sistem

Pada penelitian ini, peneliti akan berfokus pada bagian penyimpanan data yang sudah dikirimkan ke *middleware*. *Middleware* menggunakan metode *publish subscribe* dalam pertukaran pesan/data. Maka untuk menerima data dari *middleware* tersebut, harus ada *subscriber* dimana pada penelitian ini disebut *Internet Gateway Device* (IGD) dan data selanjutnya disimpan pada *data storage* MongoDB dan GridFS. Untuk memudahkan proses *get* dan *post* data, dibangun API *web service*. Untuk melihat data yang disimpan dalam *data storage*, peneliti membangun sebuah IoT App menggunakan *websocket*. Pada gambar 4.1 menjelaskan tentang gambaran umum sistem.



Gambar 4.1 Gambaran umum sistem



Gambar 4.2 Sequence diagram sistem

Sequence pada gambar 4.2, setiap kali sensor mengirimkan data ke broker, *Internet Gateway Device* (IGD) *subscribe* ke broker berdasarkan topik yang ada pada *middleware*. Setiap data diterima dari broker, IGD mengirimkannya ke API, dan *web service* akan melakukan *post* dan *get* data ke GridFS.

4.2 Kebutuhan Sistem

Kebutuhan sistem bertujuan untuk mengetahui kebutuhan yang diperlukan dalam membangun sistem dan mengimplementasikan *data storage*. Kebutuhan ini dapat mempermudah proses perancangan dan implementasi. Kebutuhan dalam penelitian ini terbagi menjadi dua yakni kebutuhan fungsional dan kebutuhan nonfungsional.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan-kebutuhan yang harus dipenuhi dan proses-proses yang dapat dilakukan oleh sistem. Untuk menjawab peramasalahan dalam penelitian disusulkan ada 4 sub sistem, yaitu IGD sebagai subscriber, web service, data storage dan web apps. Kebutuhan fungsional subs sistem tersebut dijelaskan pada Tabel 4.1 berikut ini :

Tabel 4.1 Tabel Kebutuhan Fungsional

No	Kebutuhan Fungsional
1	*Internet Gateway Device* dapat men-subscribe topik dari *middleware*, yaitu topik /Gambar, home/CO
2	*Internet Gateway Device* dapat mengambil data berdasarkan topik di *middleware,* yaitu topik /Gambar, home/CO
3	*Internet Gateway Device* dapat mengirim data berdasarkan topik ke API, yaitu topik /Gambar, home/CO
4	API web service dapat menerima data dari *Internet Gateway Device*
5	API web service dapat mengirim data dari *Internet Gateway Device* ke MongoDB GridFS
6	API web service dapat menyimpan data dari *Internet Gateway Device* ke MongoDB GridFS
7	MongoDB GridFS dapat menyimpan data sensor
8	MongoDB GridFS dapat menghapus data sensor
9	IOT App dapat menerima permintaan untuk menampilkan data berdasarkan topik, topik /Gambar, home/CO melalui protokol WebSocket
10	IOT App dapat menampilkan data berdasarkan topik, topik /Gambar, home/CO melalui protokol WebSocket

4.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional adalah kebutuhan yang menitikberatkan pada perilaku dan batasan fungsi pada sistem. Dalam penelitian ini kebutuhan nonfungsional terdiri dari kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.2.2.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras yang dibutuhkan untuk mengembangkan *Internet Gateway Device,* API *web service, data storage,* dan *IoT App* dapat dilihat pada Tabel 4.2.

Tabel 4.2 Tabel Kebutuhan Perangkat Keras

Perangkat	Keterangan
Raspberry Pi	Raspberry Pi digunakan sebagai perangkat untuk menjalankan middleware
Virtual Private Server (VPS)	Sistem ini digunakan untuk menjalankan dan menyimpan *data storage* MongoDB GridFS
NodeMCU	Perangkat ini digunakan sebagai mikrokontroler bagi

	sensor karbon monoksida yang dibuat. Mikrokontroler ini nantinya dirangkai dengan modul CO dan diprogram sehingga dapat mengirimkan data karbon monoksida.
Sensor Kamera	Modul sensor untuk mengambil gambar.
Sensor CO	Modul sensor untuk membaca karbon monoksida.
Asus X450J Core i7	Perangkat ini digunakan untuk mengakses aplikasi web untuk menampilkan data dari middleware

4.2.2.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak dibutuhkan untuk mengembangkan *Internet Gateway Device,* API *web service, data storage,* dan *IoT App* dapat dilihat pada Tabel 4.3.

Tabel 4.3 Tabel Kebutuhan Perangkat Lunak

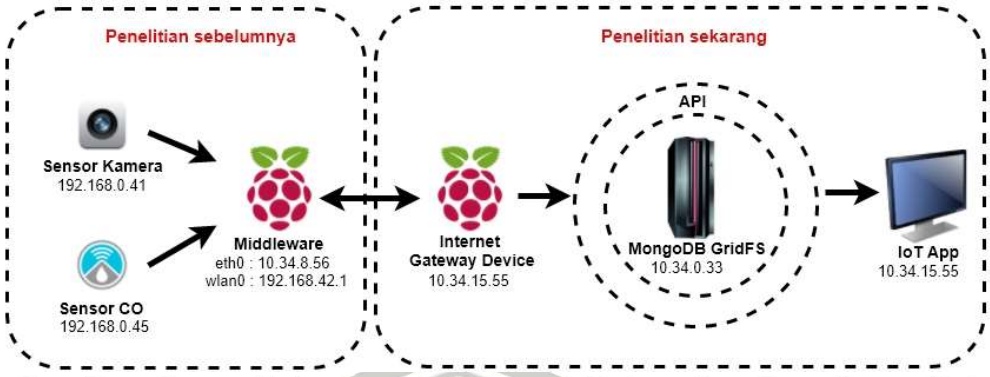
Perangkat	Keterangan
Raspbian Jessie	Operating system untuk Raspberry Pi
GridFS	Basis data No-SQL untuk menyimpan data dari sensor
Robomongo	Aplikasi manajemen *data storage* MongoDB untuk melihat data yang diterima GridFS
Putty	Aplikasi remote console/ terminal yang digunakan untuk meremote komputer dengan terhubungnya menggunakan port ssh
MobaXterm	Aplikasi remote console/ terminal yang digunakan untuk meremote komputer dengan terhubungnya menggunakan port ssh
Apache JMeter	Aplikasi *open source* berbasis Java yang dapat dipergunakan untuk *performance* tes

4.3 Perancangan

Perancangan digunakan sebagai acuan dari implementasi sehingga memudahkan proses implementasi dapat terarah dan sesuai, perancangan terdiri dari perancangan lingkungan IoT yang meliputi pengalamatan setiap sistem yang terhubung dan berkomunikasi, perancangan *Internet Gateway Device* sebagai penghubung sistem *middleware* dan juga sistem yang dikembangkan saat ini, perancangan API *restfull web service*, perancangan *data storage* MongoDB GridFS untuk penyimpanan data sensor, dan perancangan IoT App dimana akan menampilkan data sensor berdasarkan topik yang diminta.

4.3.1 Perancangan Lingkungan Sistem

Agar sistem dapat berjalan sesuai dengan yang diharapkan, perlu dibuat rancangan lingkungan sistem. Rancangan ini mencerminkan bagaimana *Internet Gateway Device* mengirim data ke *web service* dan *oleh web service* data diteruskan dan disimpan ke MongoDB GridFS. Rancangan lingkungan ini dapat dilihat pada Gambar 4.3.



Gambar 4.3 Gambar perancangan lingkungan sistem

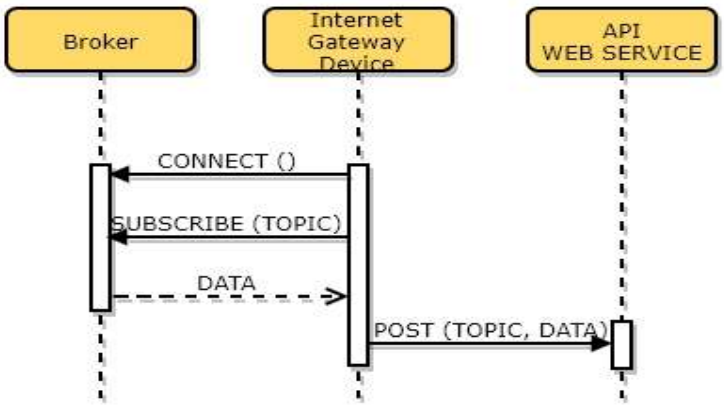
Internet Gateway Device dijalankan dan terhubung dengan Raspberrry Pi melalui jaringan LAN dengan IP 10.34.15.55. *Internet Gateway Device* ini akan berkomunikasi dengan *middleware* melalui *interface* eth0 pada Raspberrry Pi.

Untuk penerimaan data, *Internet Gateway Device* menerima data dari *middleware*. Kemudian data yang sudah berada di *Internet Gateway Device* akan dikirimkan ke web service dan web service meneruskan data untuk disimpan ke MongoDB GridFS.

Saat ingin melihat data yang sudah tersimpan dalam *data storage* GridFS, kita gunakan IoT App yang sudah dirancang. Kemudian untuk mengambil data yang sudah ditampilkan tadi, kita bisa langsung klik pada data yang ingin diambil di IoT App dengan menggunakan *web service*.

4.3.2 Perancangan *Internet Gateway Device* (IGD)

Internet Gateway Device berfungsi untuk *subscribe* topik yang ada di *middleware*, dalam hal ini berhubungan dengan broker. *Middleware* yang telah dikembangkan oleh peneliti sebelumnya menggunakan metode *publish subscribe*, sehingga membutuhkan penghubung antara *middleware* dengan *data storage* untuk menyimpan data sensor. Gambar 4.4 menjelaskan proses dari *Internet Gateway Device.*



Gambar 4.4 Sequence proses *Internet Gateway Device*

Internet Gateway Device berfungsi sebagai subscriber ke middleware pada penelitian sebelumnya. *Internet Gateway Device* akan *subscribe* berdasarkan topik, yaitu home/CO, dan /Gambar untuk menerima data yang *publish* oleh middleware. Kemudian data dari *Internet Gateway Device* akan dikirimkan ke API

4.3.3 Perancangan API *RESTfull Web Service*

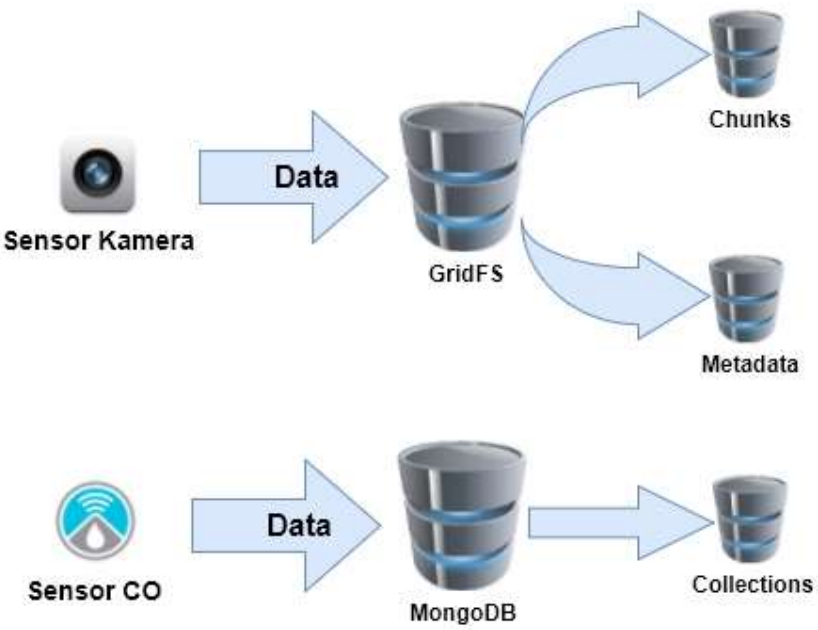
Data dari *Internet Gateway Device* akan dikirimkan dan disimpan dalam *data storage* MongoDB GridFS. Seperti pada penjelasan sebelumnya, agar data dapat dikirim dan diambil dengan mudah, maka dibangun sebuah API *RESTfull web service*. Dimana API ini akan menjadi penghubung antara *Internet Gateway Device* dan *data storage* MongoDB GridFS. Gambar 4.5 menunjukkan cara kerja API *RESTfull web service*.

Gambar 4.5 Cara kerja API RESTfull *web service*

Untuk request get, pengguna bisa mengaksesnya melalui alamat http://127.0.0.1:5000/api/getdata/“nama data yang akan diambil”.

4.3.4 Perancangan *Data Storage* MongoDB GridFS

Pada bagian perancangan *data storage*, data yang berasal dari sensor akan disimpan pada *data storage* MongoDB GridFS. Data yang berasal dari sensor DHT11/22, sensor CO, dan sensor kamera nantinya akan di subscribe oleh *Internet Gateway Device* ke middleware, kemudian *Internet Gateway Device* akan menyimpan data tersebut ke *data storage*. Untuk data dari sensor DHT11/22 dan sensor CO akan disimpan dalam MongoDB, dan untuk data dari sensor kamera akan disimpan dalam GridFS. Gambar 4.6 menunjukkan alur komunikasi sistem.



Gambar 4.6 Alur Komunikasi *data storage*

Data yang tersimpan ke dalam GridFS akan dibagi menjadi 2 bagian penyimpanan, yang pertama yaitu chunks file dimana file yg besar tadi akan dibagi menjadi beberapa chunk yang berukuran 255 kB tiap chunknya. Kemudian bagian kedua terdapat metadata. Gambar 4.7 menunjukkan representasi file chunk dan gambar 4.8 menunjukkan representasi file metadata.

```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

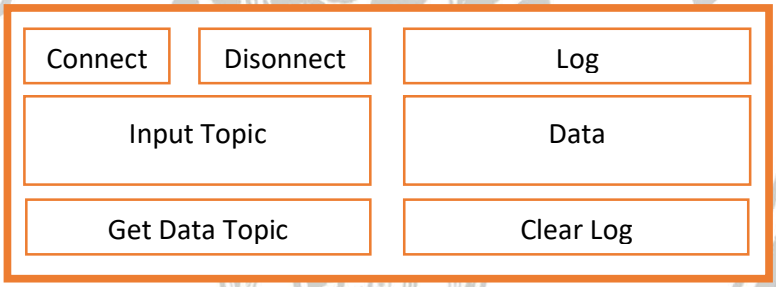
Gambar 4.7 Representasi file chunk

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <dataObject>,
}
```

Gambar 4.8 Representasi file metadata

4.3.5 Perancangan *IoT Application*

Untuk membaca dan menampilkan data yang disimpan dalam *data storage*, dibuatlah sebuah IoT App sederhana menggunakan bahasa pemrograman web seperti PHP, CSS dan Javascript. IoT App ini dapat diakses oleh browser pada smartphone ataupun laptop. Perancangan IoT App dijelaskan pada Gambar 4.9 di bawah ini.

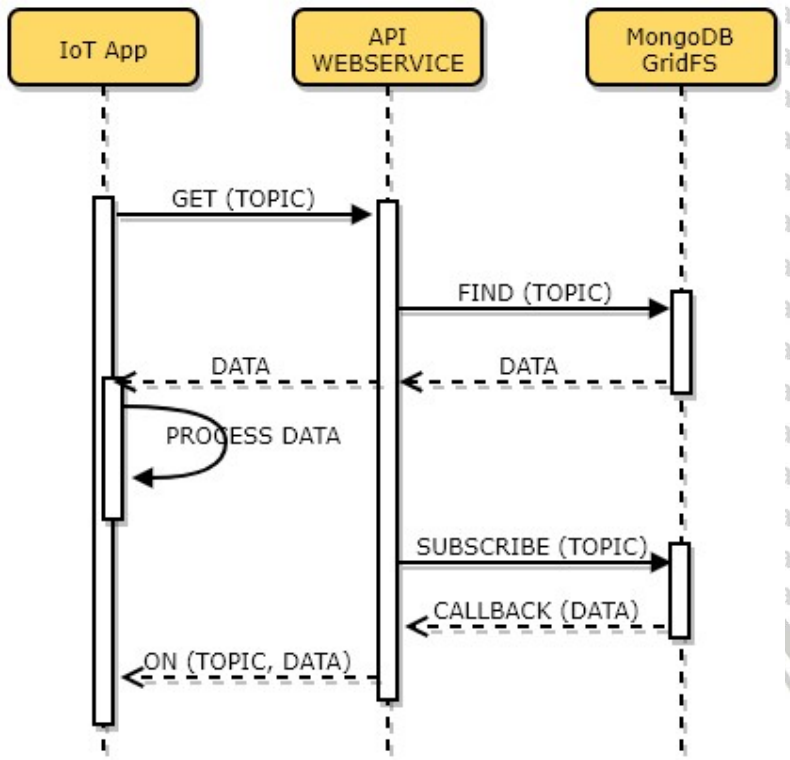


Gambar 4.9 Rancangan IoT App

Seperti yang ditunjukkan Gambar 4.9 di atas, IoT App ini memiliki beberapa komponen bagi pengguna untuk berinteraksi. Untuk menampilkan data dari *data storage*, pengguna mengisi nama topik di kolom *input topic* kemudian meng-klik tombol *get data topic*. Selanjutnya setiap kali data bertambah didalam GridFS, data tersebut akan ditampilkan secara real-time pada kolom data. User dapat mengulangi proses yang sama untuk membaca data dari topik lainnya.

Dari segi teknis, ketika pengguna mengakses dari browser, aplikasi ini menginisialisasi koneksi dengan MongoDB GridFS melalui protokol *Websocket*. Dengan data yang sudah tersimpan, ketika pengguna membuka IoT App ini dilain waktu, data yang sudah disimpan sebelumnya dapat ditampilkan.

Kemudian dalam IoT App ini, kita bisa langsung mengambil data yang sudah ditampilkan sebelumnya karena sudah disambungkan dengan *web service*. Jadi pengguna hanya meng-klik data mana yang sudah ditampilkan yang akan diambil.



Gambar 4.10 Sequence diagram IoT App

Pada Gambar 4.10 proses dalam IoT Apps terjadi saat masukan dari IoT Apps yang berupa topik. Dari topik yang diminta, data akan diambil dari penyimpanan data MongoDB atau GridFS sesuai topik tadi.

4.3.6 Perancangan Pengujian

Perancangan pengujian dibutuhkan untuk menentukan batasan minimal yang harus dipenuhi dan skenario pengujian dari data *center* yang dikembangkan.

4.3.6.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang dikembangkan sudah memenuhi kebutuhan fungsional yang sudah ditentukan. Apabila semua kebutuhan fungsional terpenuhi maka dilanjutkan pengujian berikutnya. Namun apabila ada kebutuhan fungsional yang belum berjalan seusai harapan, maka dilakukan pengecekan kembali terhadap rancangan dan kode sistem kemudian melakukan pengujian sekali lagi, begitu seterusnya hingga semua kebutuhan terpenuhi. Skenario pengujian yang dilakukan dalam penelitian kali ini dapat dilihat pada

Tabel 4.4.

Tabel 4.4 Skenario pengujian fungsional

Kode	Fungsi	Skenario
DC_001	*Internet Gateway Device* dapat men-subscribe topik dari *middleware*, yaitu topik /Gambar, home/CO	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device*
DC_002	*Internet Gateway Device* dapat mengambil data berdasarkan topik di *middleware,* yaitu topik /Gambar, home/CO	4. Middleware sudah berjalan 5. Pengguna menjalankan *Internet Gateway Device*
DC_003	*Internet Gateway Device* dapat mengirim data berdasarkan topik ke API, yaitu topik /Gambar, home/CO	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
DC_004	API web service dapat menerima data dari *Internet Gateway Device*	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
DC_005	API web service dapat mengirim data dari *Internet Gateway Device* ke GridFS	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
DC_006	API web service dapat menyimpan data dari *Internet Gateway Device* ke GridFS	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
DC_007	GridFS dapat menyimpan data sensor	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device* 3. Pengguna menjalankan *websocket* 4. Pengguna membuka Robomongo untuk melihat data yang disimpan pada GridFS 5. Pengguna melihat data pada *collection* penyimpanan data

DC_008	GridFS dapat menghapus data sensor	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device* 3. Pengguna menjalankan *websocket* 4. Pengguna membuka Robomongo untuk melihat data yang disimpan pada GridFS 5. Pengguna menghapus data pada GridFS
DC_009	IOT App dapat menerima permintaan menampilkan data berdasarkan topik, topik /Gambar, home/CO	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device* 4. Pengguna membuka dan menjalankan IoT App 5. Pengguna menginputkan topik yang akan ditampilkan pada IoT App
DC_010	IOT App dapat menampilkan data berdasarkan topik, topik /Gambar, home/CO melalui protokol WebSocket	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device* 4. Pengguna membuka dan menjalankan IoT App 5. Pengguna menginputkan topik yang akan ditampilkan pada IoT App 6. IoT App menampilkan data

4.3.6.2 Pengujian Skalabilitas API *Web Service*

Pengujian skalabilitas dilakukan untuk mengetahui sistem yang dikembangkan dalam menangani peningkatan beban sistem.

4.3.6.3 Pengujian *Response Time*

Pengujian *response time* dilakukan untuk mengetahui waktu tanggap sistem ketika ada *request* yang masuk.

BAB 5 IMPLEMENTASI

Implementasi adalah bagian terpenting dalam sebuah pengembangan sistem berhasil atau tidaknya sistem yang dibuat ditentukan oleh keberhasilan dari implementasinya, implementasi juga bertujuan untuk memenuhi tujuan dari penelitian yang dilakukan, implementasi dilakukan berdasarkan dengan perancangan yang telah dibuat pada bab sebelumnya, yaitu implementasi lingkungan sistem yang meliputi bagaimana data dikirim dan diterima, implementasi *Internet Gateway Device* sebagai penghubung sistem *middleware* dan juga sistem yang dikembangkan saat ini, implementasi API *restfull web service*, implementasi *data storage* GridFS untuk penyimpanan data sensor, dan implementasi IoT app dimana akan menjelaskan dan menggambarkan data dalam web.

5.1 Implementasi Lingkungan Sistem

Implementasi topologi jaringan akan membahas konfigurasi yang dibutuhkan oleh *Internet Gateway Device* dan laptop dalam membangun sistem untuk pengujian sistem. Konfigurasi yang akan dilakukan adalah mengatur alamat IP pada *interface* eth0 menjadi *static*. Selain itu diperlukan juga konfigurasi IP pada *Internet Gateway Device* untuk berkomunikasi dengan Raspberry Pi, dan *data storage.*

5.1.1 Konfigurasi Middleware

Sesuai perancangan pada Gambar 4.3, sudah dikembangkan sebuah middleware yang dapat menerima data sensor menggunakan multiprotocol. Konfigurasi pada Raspberry Pi menggunakan IP static pada interface eth0 yaitu 10.34.8.56. Kemudian Raspberry Pi dapat digunakan sebagai *access point*. Selanjutnya konfigurasi sensor dilakukan dengan menghubungkan sensor ke access point dari Raspberry Pi untuk mendapatkan IP.

5.1.2 Konfigurasi Pada *Internet Gateway Device*

Internet Gateway Device ialah sistem yang terhubung secara langsung dengan *middleware*, oleh karena itu IGD harus sudah terkoneksi dengan *middleware* dengan IP 10.34.15.56, proses *subscribe* data bisa berjalan. Gambar 5.1 menunjukkan pengaturan koneksi pada *Internet Gateway Device*.

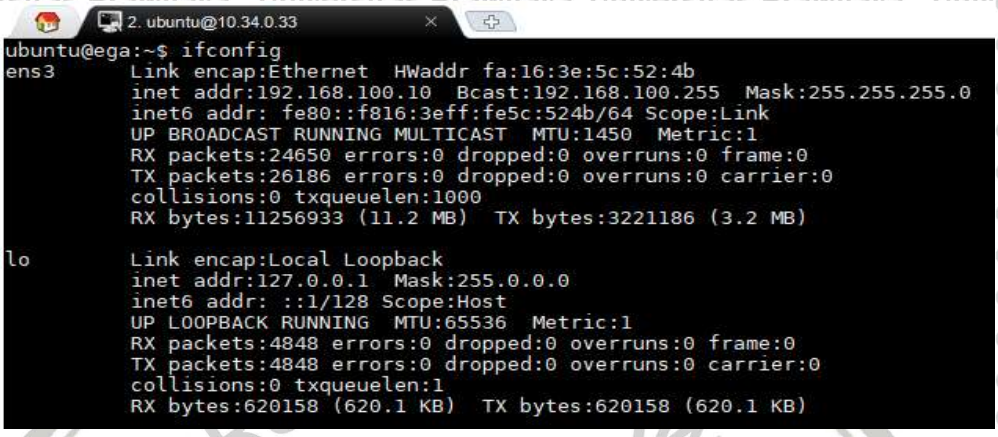
```
mqttc = mqtt.Client("server", clean_session=False)
mqttc.connect("10.34.15.56", 1883)
```

Gambar 5.1 Pengaturan koneksi pada *Internet Gateway Device*

Middleware yang telah dikembangkan pada penelitian sebelumnya menggunakan metode publish/subscribe dalam pertukaran pesan, sehingga pada penelitian saat ini, dibangunlah *Internet Gateway Device* untuk subscribe ke middleware.

5.1.3 Konfigurasi Pada Data Storage

Data storage disini dikembangkan di sebuah *Virtual Private Server* (VPS) dengan IP eksternal 10.34.0.33. Gambar 5.2 menunjukkan konfigurasi IP pada *data storage*.



Gambar 5.2 Konfigurasi pada VPS

5.2 Implementasi *Internet Gateway Device* (IGD)

Implementasi *Internet Gateway Device* dilakukan untuk membuat sebuah sistem yang dapat melakukan komunikasi yaitu subscribe ke *middleware* untuk menerima data sensor. Implementasi dilakukan pada setiap data yang disubscribe berdasarkan topik yang ada, yaitu /Gambar dan home/CO. Gambar 5.3 kode implementasi *Internet Gateway Device* untuk *subscribe* ke topik yang ada pada *middleware*.

```
mqttc.on_message = on_message

mqttc.subscribe("/Gambar")
mqttc.subscribe("home/CO")
```

Gambar 5.3 Kode program untuk *subscribe*

Gambar 5.3 menjelaskan *Internet Gateway Device* melakukan *subscribe* terhadap topik /Gambar dan home/CO.

```
def on_message(mqttc,obj,msg):
    if msg.topic=='/Gambar':
        headers = {"Content-type": "application/json"}
        params = msg.payload
        conn.request("POST", "/api/postdata", params, headers)
        response = conn.getresponse()
        print response.read()

    elif msg.topic=='home/CO' :
        headers = {"Content-type": "application/json"}
        params = msg.payload
        conn.request("POST", "/api/postdataco", params, headers)
        response = conn.getresponse()
        print response.read()
```

Gambar 5.4 Kode program subscribe topik pada middleware

Pada Gambar 5.4 diatas menjelaskan bagaimana *Internet Gateway Device* akan menerima pesan setiap middleware mengirimkan pesan pada setiap topik yang di *subscribe* ke *middleware* (mqtt.subscribe), IGD kemudian melakukan POST data yang diterima dari *middleware* ke API *web service*, kode program pada Gambar 5.4.

5.3 Implementasi API *RESTfull Web Service*

Pembuatan *web service* ditulis dengan bahasa pemrograman python yang dilakukan ketika menjalankan *Internet Gateway Device*.

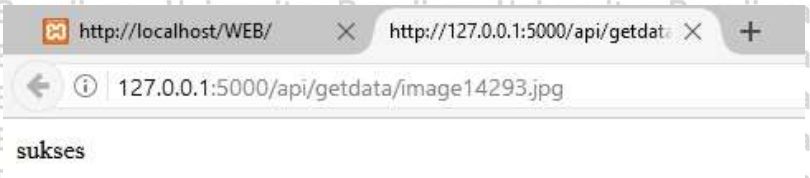
5.3.1 *Methods* GET

Implementasi *web service* GET yang dibuat dapat dilihat pada Gambar 5.5. Data yang diambil berdasarkan data yang sudah tersimpan dalam penyimpanan data. Oleh karena itu, pertama harus melakukan koneksi ke database dengan nama dataGambar, dan melakukan inisialisasi ke GridFS karena pada data gambar disimpan dalam GridFS. Setelah proses koneksi selesai, data akan dicari sesuai permintaan pada *get* tadi, dan akan dicari menggunakan *find()* pada GridFS.

```
#GET DATA GAMBAR
@app2.route('/api/getdata/<string:name>', methods=['GET'])
def getdata(name):
    db = client.dataGambar
    fs = gridfs.GridFS(db)
    with open(name, "wb") as fInput:
            getdata=fs.find_one({"filename": name})._id
            x=fs.get(getdata).read()
            y=json.loads(x)
            z=pickle.loads(y["Data"])
            fInput.write(z)
            return "DATA GAMBAR BERHASIL DIAMBIL"
```

Gambar 5.5 Kode program *web service* GET untuk data gambar

Pada saat websevice sudah dijalankan melalui *Internet Gateway Device*, maka selanjutnya kita bisa menjalankannya dengan masuk ke alamat http://127.0.0.1:5000/api/getdata/"nama data yang akan diambil". Gambar 5.6 menunjukkan hasil dari *web service* yang dijalankan.



Gambar 5.6 Hasil dari *web service* yang dijalankan

5.3.2 *Methods* POST

Implementasi *web service* POST yang dibuat dapat dilihat pada Gambar 5.7 dan Gambar 5.8. Untuk method post dilakukan request data pada *Internet Gateway Device* yang mengirimkan datanya. Kemudian data yang diterima akan disimpan ke dataGambar pada GridFS, karena berupa gambar. Disini peneliti menambahkan metadata berupa filename untuk disimpan pada GridfFS, sehingga memudahkan untuk membaca disetiap file yang disimpan.

```
#POST DATA GAMBAR
@app2.route('/api/postdata', methods=['POST'])
def postdata():
        data = request.get_json()
        db = client.dataGambar
        fs = gridfs.GridFS(db)
        #print type(data)
        print data["Name"]
        message = json.dumps(data)
        fs.put(message,filename=data["Name"])
        baru={"data":"dataGAMBARbaru","filename":data["Name"]}
        jsonbaru = json.dumps(baru)
        s.send(jsonbaru)
        print jsonbaru
        return "DATA GAMBAR TERKIRIM"
```

Gambar 5.7 Kode program *web service* POST untuk data gambar

Untuk data karbon monoksida yang berasal dari topik home/CO, dilakukan request data pada *Internet Gateway Device* yang mengirimkan datanya. Kemudian data yang diterima akan disimpan ke dataCO pada MongoDB. Disini peneliti menyimpan metadata berupa *protocol, temperature, timestamp, humidity,* dan *topic,* untuk disimpan pada MongoDB.

```
#POST DATA CO
@app2.route('/api/postdataco', methods=['POST'])
def postdataco():
        data = request.get_json()
        db = client.dataCO
        print data
        timestamp = str(datetime.datetime.now())
        message = json.dumps(data)
        db.dataCO.insert_one({
            'protocol':data['protocol'],
            'temperature':data['temperature'],
            'timestamp':data['timestamp'],
            'humidity':data['humidity'],
            'topic':data['topic']})
        baru={"data":"dataCObaru", "temperature":data["temperature"],"timestamp":timestamp}
        jsonbaru = json.dumps(baru)
        s.send(jsonbaru)
        print jsonbaru
        return "DATA CO TERKIRIM"
```

Gambar 5.8 Kode program *web service* POST untuk data CO

Keluaran yang dihasilkan dari kode program post untuk data CO dapat dilihat pada Gambar 5.9.

```
ubuntu@ega:~/skripsi$ sudo python subscriber.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 288-572-407
{u'protocol': u'mqtt', u'temperature': {u'unit': u'celcius', u'value': 0}, u'timest
, u'topic': u'home/CO', u'sensor': {u'index': 1021055, u'tipe': u'esp8266', u'modul
{"timestamp": "2017-08-01 11:35:54.165665", "data": "dataCObaru", "temperature": {"
127.0.0.1 - - [01/Aug/2017 11:35:54] "POST /api/postdataco HTTP/1.1" 200 -
DATA CO TERKIRIM
```

Gambar 5.9 Keluaran kode program *web service* POST data CO

5.4 Implementasi *Data Storage*

Implementasi *data storage* dilakukan untuk membuat sebuah *data storage* untuk menyimpan data sensor yang diterima, dalam pembuatan sistem ini yang digunakan adalah *data storage* MongoDB GridFS.

5.4.1 Instalasi MongoDB

Perintah dalam menginstall *data storage* MongoDB adalah:
Gambar 5.10 menunjukkan instalasi MongoDB.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6

echo "deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu
precise/mongodb-org/3.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.4.list

sudo apt-get update

sudo apt-get install -y mongodb-org
```

Gambar 5.10 Instalasi MongoDB

Setelah *data storage* MongoDB telah terinstall, sekarang saatnya menjalankan *data storage* tersebut dengan perintah:
Gambar 5.11 menunjukkan perintah menjalankan MongoDB.

```
sudo service mongod start
sudo service mongo start
```

Gambar 5.11 Perintah menjalankan MongoDB

5.4.2 Implementasi MongoDB GridFS

Pembuatan *data storage* GridFS ditulis dengan bahasa pemrograman python yang dilakukan oleh *Internet Gateway Device*, untuk implementasi *data storage* yang digunakan dapat dilihat pada berikut:
Gambar 5.12 menunjukkan implementasi *data storage* MongoDB GridFS data gambar.

```
1  db = client.dataGambar
2  fs = gridfs.GridFS(db)
```

Gambar 5.12 Implementasi *data storage* GridFS data gambar

Gambar 5.13 menunjukkan implementasi *data storage* MongoDB data CO.

```
1  db = client.dataCO
```

Gambar 5.13 Implementasi *data storage* GridFS data CO

Untuk membuat *data storage* pada GridFS diperlukan untuk terkoneksi terlebih dahulu dengan MongoDB serta nama collection yang akan dibuat untuk menyimpan data, penjelasan dari adalah sebagai berikut:

1. Baris 1 pada progam adalah perintah untuk membuat koneksi dengan *data storage* MongoDB GridFS dan membuat collection untuk menyimpan data.
2. Baris 2 adalah perintah untuk membuat *data storage* GridFS dengan menggunakan parameter db dimana sudah didefinisikan pada kode sebelumnya.

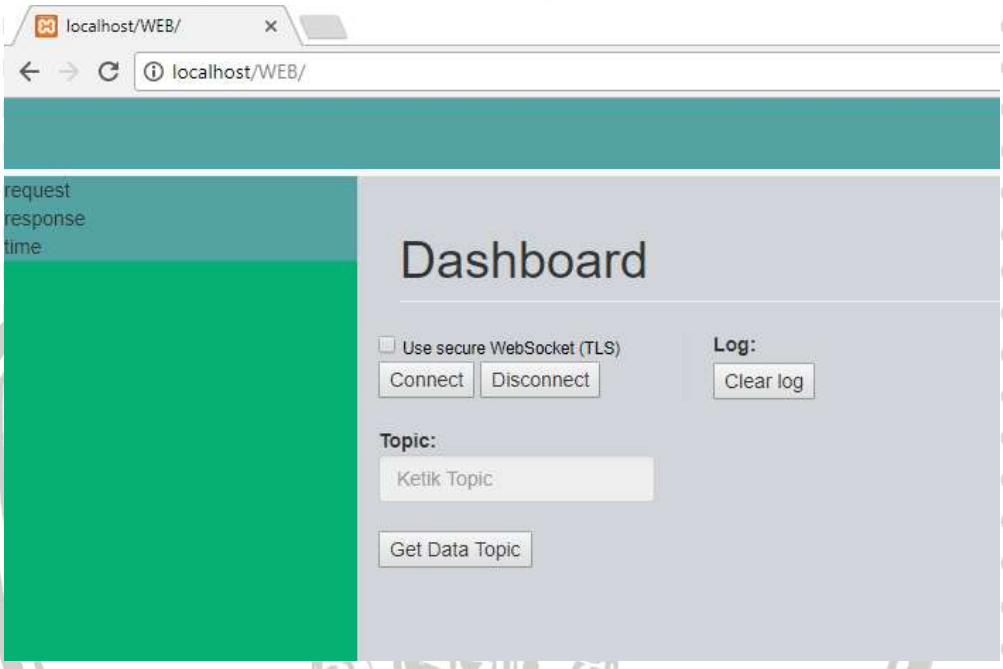
```
client = MongoClient("10.34.0.33", 27017)
```

Gambar 5.14 Pengalamatan penyimpanan pada MongoDB GridFS

Pada Gambar 5.14 menjelaskan penyimpanan data pada *data storage* yang berapa pada VPS di alamat 10.34.0.33.

5.5 Implementasi IoT Application

Web dikembangkan menggunakan PHP untuk web yang ditampilkan. Implementasi web ini dimulai dengan membuat kode program dapat terhubung dan membaca data dari *data storage* melalui protokol *Websocket* sesuai dengan bagian perancangan. Selanjutnya membuat web yang ditampilkan pada browser menggunakan PHP, CSS dan Javascript. Hasil akhir implementasi dari aplikasi web ini dapat dilihat pada Gambar 5.15. Untuk penjelasan komponen-komponen pada IoT Application sudah dijelaskan pada bab perancangan.



Gambar 5.15 Tampilan Web

BAB 6 PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

6.1 Pengujian Fungsionalitas

Pengujian fungsional dilakukan untuk untuk melihat kesesuaian fungsi – fungsi hasil implementasi dengan perancangan. Dari hasil analisis pengujian fungsionalitas dapat dilihat apakah fungsi yang ada pada sistem *data storage* telah berjalan dengan benar.

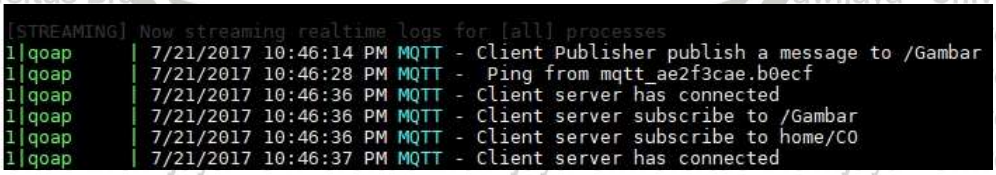
6.1.1 Subscribe Topik Gambar dan CO

Kasus uji proses subscribe topik gambar dapat ditunjukkan pada Tabel 6.1

Tabel 6.1 Subscribe topik gambar

Kode	[DC_001]
Nama Kasus Uji	Subscribe topik gambar
Tujuan Pengujian	Menguji proses subscribe topik /Gambar dari middleware
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat mensubscribe topik /Gambar dari middleware
Hasil Pengujian	*Internet Gateway Device* berhasil mensubscribe topik /Gambar dari middleware

Gambar 6.1 menunjukkan client dengan nama server (*Internet Gateway Device*) berhasil subcribe topik gambar dari middleware.



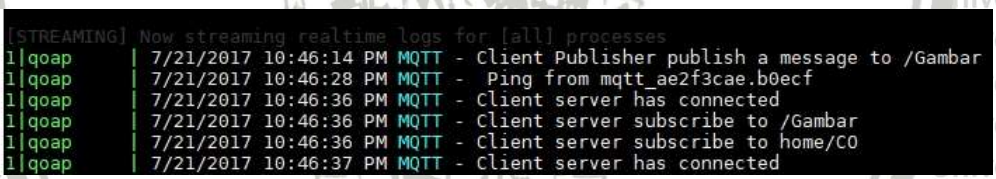
Gambar 6.1 Subscribe topik gambar

Kasus uji proses subscribe topik CO dapat ditunjukkan pada Tabel 6.2

Tabel 6.2 Subscribe topik CO

Nama Kasus Uji	Subscribe topik gambar
Tujuan Pengujian	Menguji proses subscribe topik home/CO dari middleware
Prosedur Pengujian	4. Middleware sudah berjalan 5. Pengguna memonitoring log pada middleware 6. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat mensubscribe topik home/CO dari middleware
Hasil Pengujian	*Internet Gateway Device* berhasil mensubscribe topik home/CO dari middleware

Gambar 6.2 menunjukkan client dengan nama server (*Internet Gateway Device*) berhasil subcribe topik karbon monoksida dari middleware.



Gambar 6.2 Subscribe topik CO

6.1.2 Mengambil Data Gambar dan CO

Kasus uji proses mengambil data gambar dapat ditunjukkan pada Tabel 6.3

Tabel 6.3 Mengambil data gambar

Kode	[DC_002]
Nama Kasus Uji	Mengambil data gambar
Tujuan Pengujian	Menguji proses mengambil data gambar dari topik /Gambar di middleware
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat

	mengambil data gambar dari topik /Gambar di middleware
Hasil Pengujian	*Internet Gateway Device* berhasil mengambil data gambar dari topik /Gambar di middleware

Gambar 6.3 menunjukkan *Internet Gateway Device* berhasil mengambil data gambar melalui topik /Gambar dari middleware.

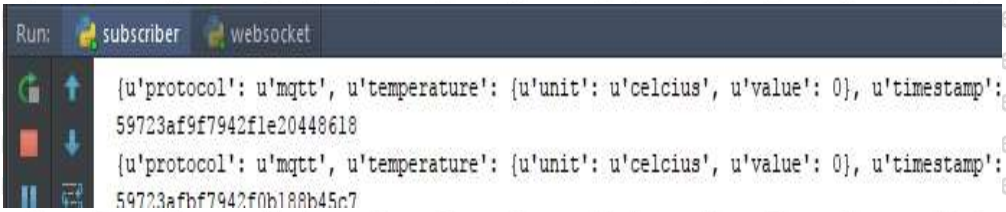
Gambar 6.3 Mengambil data gambar

Kasus uji proses mengambil data karbon monoksida dapat ditunjukkan pada Tabel 6.4

Tabel 6.4 Mengambil data CO

Nama Kasus Uji	Mengambil data CO
Tujuan Pengujian	Menguji proses mengambil data karbon monoksida dari topik home/CO di middleware
Prosedur Pengujian	3. Middleware sudah berjalan 4. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat mengambil data karbon monoksida dari topik home/CO di middleware
Hasil Pengujian	*Internet Gateway Device* berhasil mengambil data karbon monoksida dari topik home/CO di middleware

Gambar 6.4 menunjukkan *Internet Gateway Device* berhasil mengambil data karbon monoksida melalui topik home/CO di middleware.



Gambar 6.4 Mengambil data CO

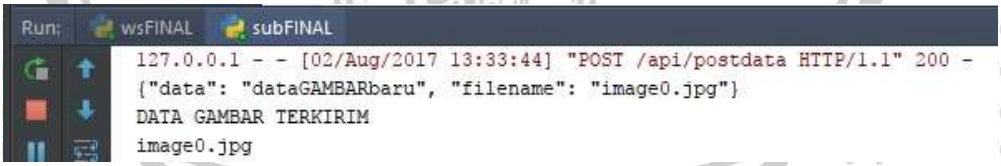
6.1.3 Mengirim Data Gambar dan CO ke API

Kasus uji proses mengirim data gambar ke API dapat ditunjukkan pada Tabel 6.5

Tabel 6.5 Mengirim data gambar ke API

Kode	[DC_003]
Nama Kasus Uji	Mengirim data gambar ke API
Tujuan Pengujian	Menguji proses mengirim data gambar dari *Internet Gateway Device* ke API
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat data gambar ke API
Hasil Pengujian	*Internet Gateway Device* berhasil mengirimkan data gambar ke API

Gambar 6.5 menunjukkan data gambar berhasil dikirim ke API.



Gambar 6.5 Mengirim data gambar ke API

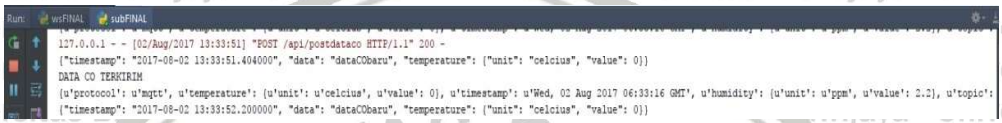
Kasus uji proses mengirim data karbon monoksida ke API dapat ditunjukkan pada Tabel 6.6

Tabel 6.6 Mengirim data CO ke API

Nama Kasus Uji	Mengirim data CO ke API
Tujuan Pengujian	Menguji proses mengirim data CO dari *Internet Gateway Device* ke API

Prosedur Pengujian	3. Middleware sudah berjalan 4. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Internet Gateway Device* dapat data CO ke API
Hasil Pengujian	*Internet Gateway Device* berhasil mengirimkan data CO ke API

Gambar 6.6 menunjukkan data karbon monoksida berhasil dikirim ke API.



Gambar 6.6 Mengirim data CO ke API

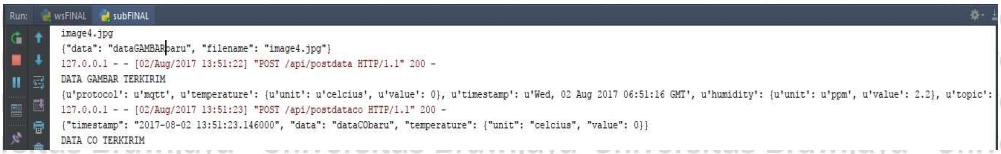
6.1.4 Menerima Data dari *Internet Gateway Device*

Kasus uji proses *web service* menerima data dari *Internet Gateway Device* dapat dilihat pada Tabel 6.7

Tabel 6.7 Menerima data dari *Internet Gateway Device*

Kode	[DC_004]
Nama Kasus Uji	Menerima data dari *Internet Gateway Device*
Tujuan Pengujian	Menguji proses *web service* menerima data dari *Internet Gateway Device*
Prosedur Pengujian	1. *Middleware* sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Web service* dapat menerima data dari *Internet Gateway Device*
Hasil Pengujian	*Web service* berhasil menerima data dari *Internet Gateway Device*

Gambar 6.7 menunjukkan proses *web service* menerima data dari *Internet Gateway Device.*



Gambar 6.7 Menerima data dari *Internet Gateway Device*

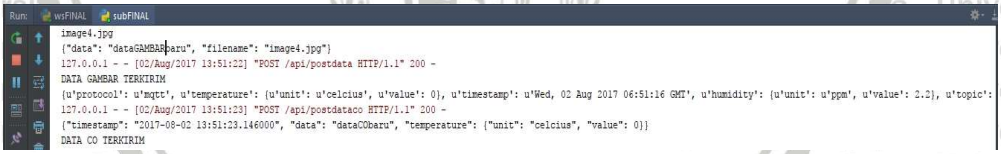
6.1.5 Mengirim Data dari *Internet Gateway Device*

Kasus uji proses *web service* mengirim data dari *Internet Gateway Device* dapat dilihat pada Tabel 6.8

Tabel 6.8 Mengirim data dari *Internet Gateway Device*

Kode	[DC_005]
Nama Kasus Uji	Mengirim data dari *Internet Gateway Device*
Tujuan Pengujian	Menguji proses *web service* mengirim data dari *Internet Gateway Device*
Prosedur Pengujian	1. *Middleware* sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	*Web service* dapat mengirim data dari *Internet Gateway Device*
Hasil Pengujian	*Web service* berhasil mengirim data dari *Internet Gateway Device*

Gambar 6.8 menunjukkan proses *web service* mengirim data dari *Internet Gateway Device.*



Gambar 6.8 Mengirim data dari *Internet Gateway Device*

6.1.6 Menyimpan Data ke GridFS

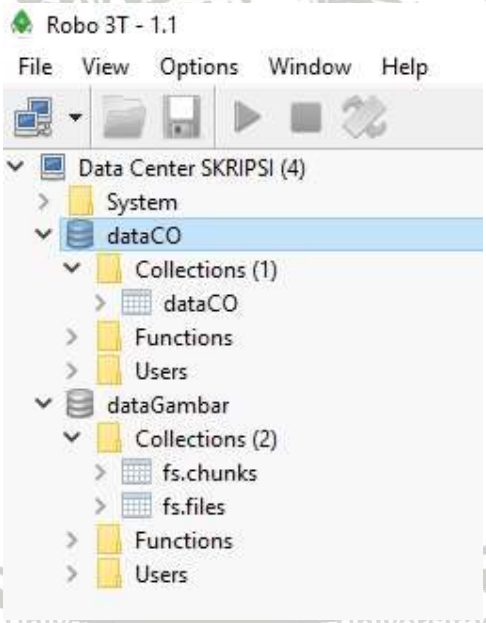
Kasus uji proses *web service* menyimpan data dari *Internet Gateway Device* ke GridFS dapat dilihat pada Tabel 6.9

Tabel 6.9 Menyimpan data dari *Internet Gateway Device* ke GridFS

Kode	[DC_006]
Nama Kasus Uji	Menyimpan data dari *Internet Gateway Device* ke GridFS

Tujuan Pengujian	Menguji proses *web service* menyimpan data dari *Internet Gateway Device* ke GridFS
Prosedur Pengujian	1. *Middleware* sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device*
Hasil yang Diharapkan	W*eb service* dapat menyimpan data dari *Internet Gateway Device* ke GridFS
Hasil Pengujian	W*eb service* berhasil menyimpan data dari *Internet Gateway Device* ke GridFS

Gambar 6.9 menunjukkan proses *web service* menyimpan data dari *Internet Gateway Device* ke GridFS. Koleksi yang disimpan berupa dataCO dan dataGambar yang merupakan koleksi dari data sensor yang dikirim dari *web service*.



Gambar 6.9 Menyimpan data dari *Internet Gateway Device* ke GridFS

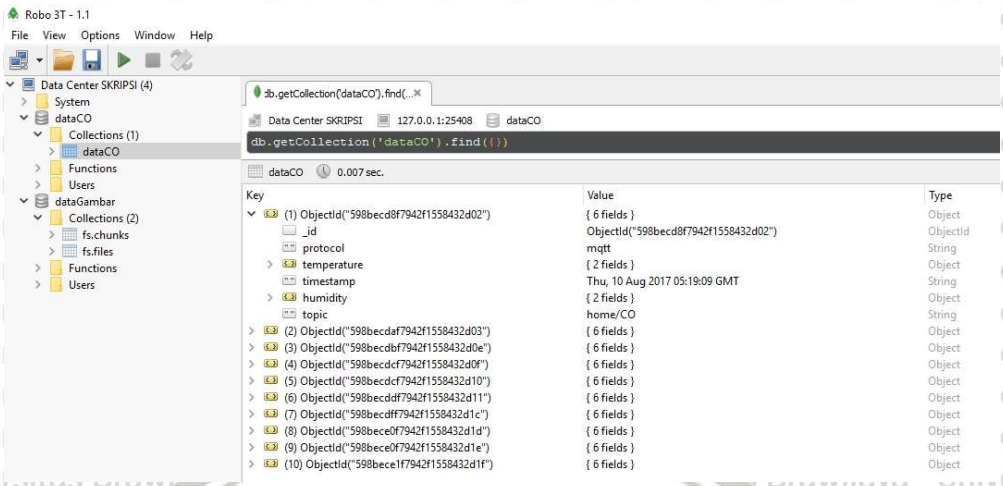
6.1.7 Menyimpan Data pada GridFS

Kasus uji proses menyimpan data sensor yang disimpan oleh *Internet Gateway Device* ke GridFS dapat ditunjukkan pada Tabel 6.10.

Tabel 6.10 Menyimpan data sensor

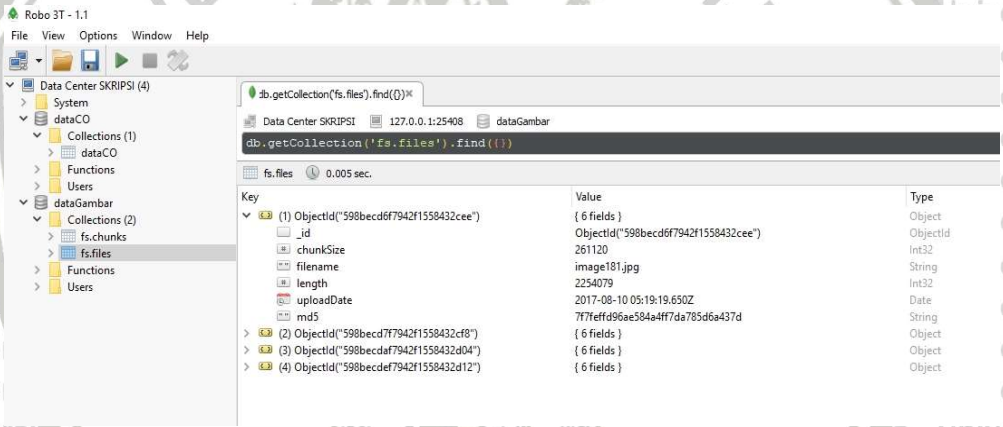
Kode	[DC_007]
Nama Kasus Uji	Menyimpan data sensor ke GridFS
Tujuan Pengujian	Menguji proses menyimpan data sensor yang disimpan oleh *Internet Gateway Device* ke GridFS
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device* 3. Pengguna menjalankan *websocket* 4. Pengguna membuka Robomongo untuk melihat data yang disimpan pada GridFS 5. Pengguna melihat data pada *collection* penyimpanan data
Hasil yang Diharapkan	GridFS dapat menyimpan data sensor yang disimpan oleh *Internet Gateway Device*
Hasil Pengujian	GridFS berhasil menyimpan data sensor yang disimpan oleh *Internet Gateway Device*

Gambar 6.10 menunjukkan data CO yang berhasil disimpan pada GridFS. Data yang disimpan pada GridFS dibagi menjadi file chunks dan file metadata. Untuk melihat isi dari data sensor, lihat pada file metadata.



Gambar 6.10 Data CO yang disimpan pada GridFS

Gambar 6.11 menunjukkan data CO yang berhasil disimpan pada GridFS. Data yang disimpan pada GridFS dibagi menjadi file chunks dan file metadata. Untuk melihat isi dari data sensor, lihat pada file metadata.



Gambar 6.11 Data gambar yang disimpan pada GridFS

6.1.8 Menghapus Data pada GridFS

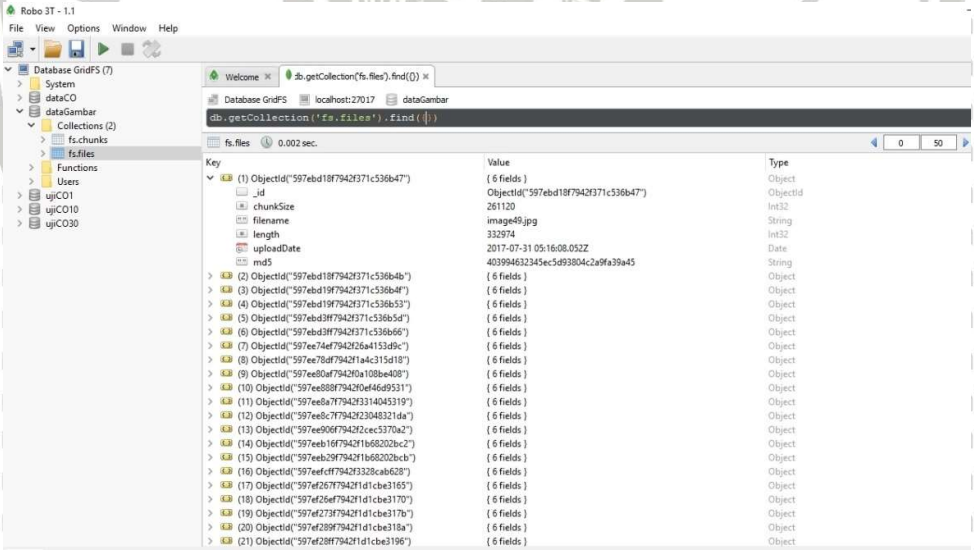
Kasus uji proses menghapus data sensor yang disimpan oleh *Internet Gateway Device* ke GridFS dapat ditunjukkan pada Tabel 6.11

Tabel 6.11 Menyimpan data sensor

Kode	[DC_008]
Nama Kasus Uji	Mnyimpan data sensor ke GridFS
Tujuan Pengujian	Menguji proses menghapus data sensor yang disimpan oleh *Internet Gateway Device* ke GridFS

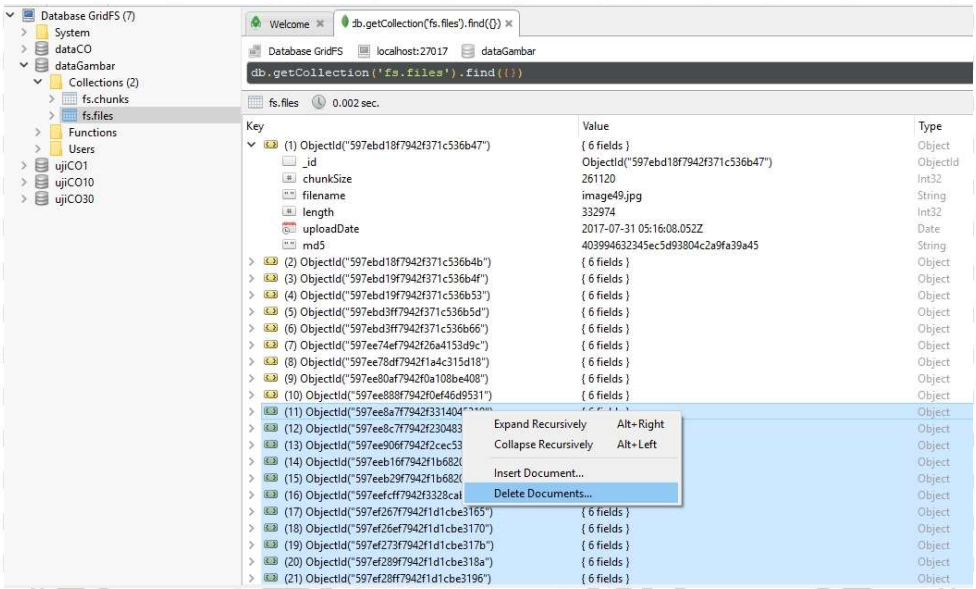
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna menjalankan *Internet Gateway Device* 3. Pengguna menjalankan *websocket* 4. Pengguna membuka Robomongo untuk melihat data yang disimpan pada GridFS 5. Pengguna menghapus data pada GridFS
Hasil yang Diharapkan	GridFS dapat menghapus data sensor yang disimpan oleh *Internet Gateway Device*
Hasil Pengujian	GridFS berhasil menghapus data sensor yang disimpan oleh *Internet Gateway Device*

Gambar 6.12 menunjukkan data gambar sebelum dihapus pada GridFS. Data berada pada file metadata di koleksi dataGambar. Data yang disimpan berjumlah 21 data gambar sebelum dihapus.



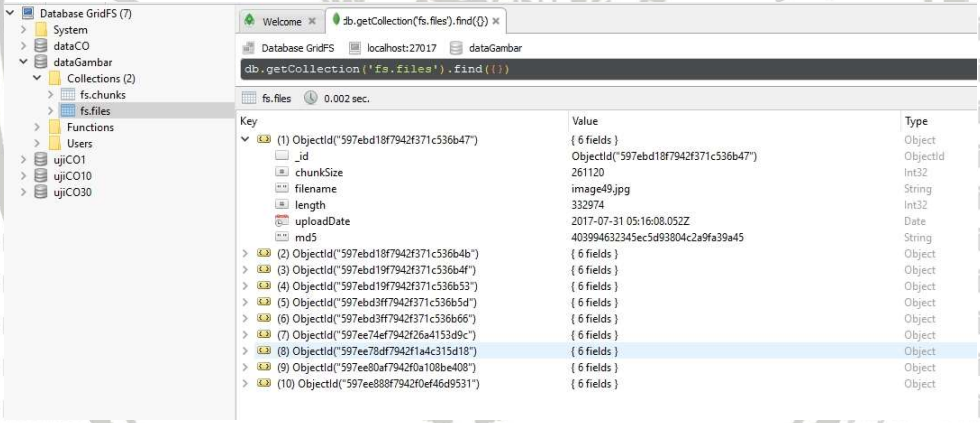
Gambar 6.12 Data gambar sebelum dihapus

Pada Gambar 6.13, dilakukan *delete documents* pada koleksi dataGambar sebanyak 11 data.



Gambar 6.13 Data gambar sebelum dihapus

Gambar 6.14 menunjukkan data gambar sesudah dihapus pada GridFS. Tersisa 10 data gambar yang masih tersimpan setelah dilakukan *delete documents*.



Gambar 6.14 Data gambar sesudah dihapus

6.1.9 Menerima Data pada IoT App

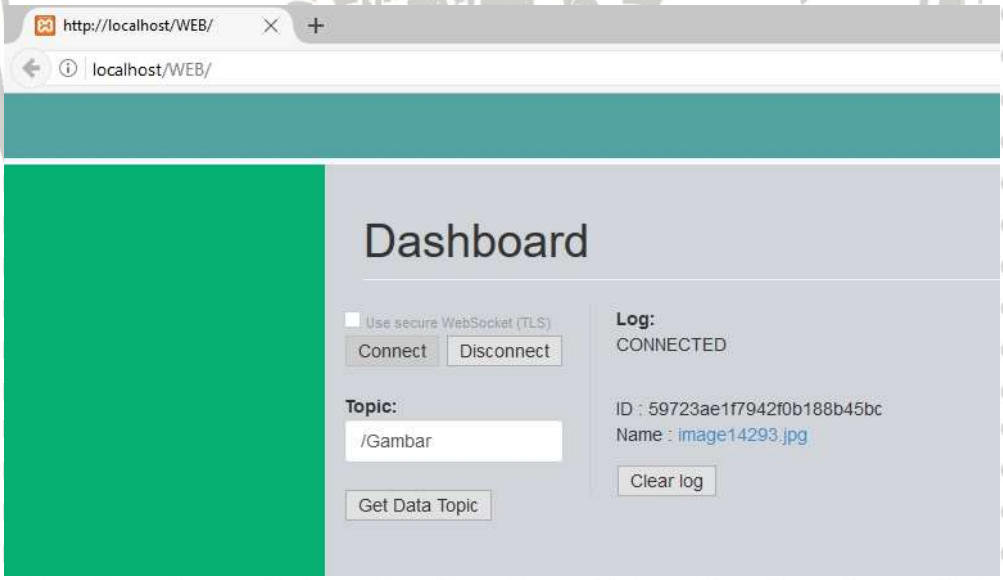
Kasus uji proses subscribe topik gambar dapat ditunjukkan pada Tabel 6.12.

Tabel 6.12 Menampilkan Data pada IoT App

Kode	[DC_009]
Nama Kasus Uji	Menerima Data pada IoT App
Tujuan Pengujian	Menguji proses permintaan menampilkan data berdasarkan topik, topik /Gambar, home/CO pada IoT

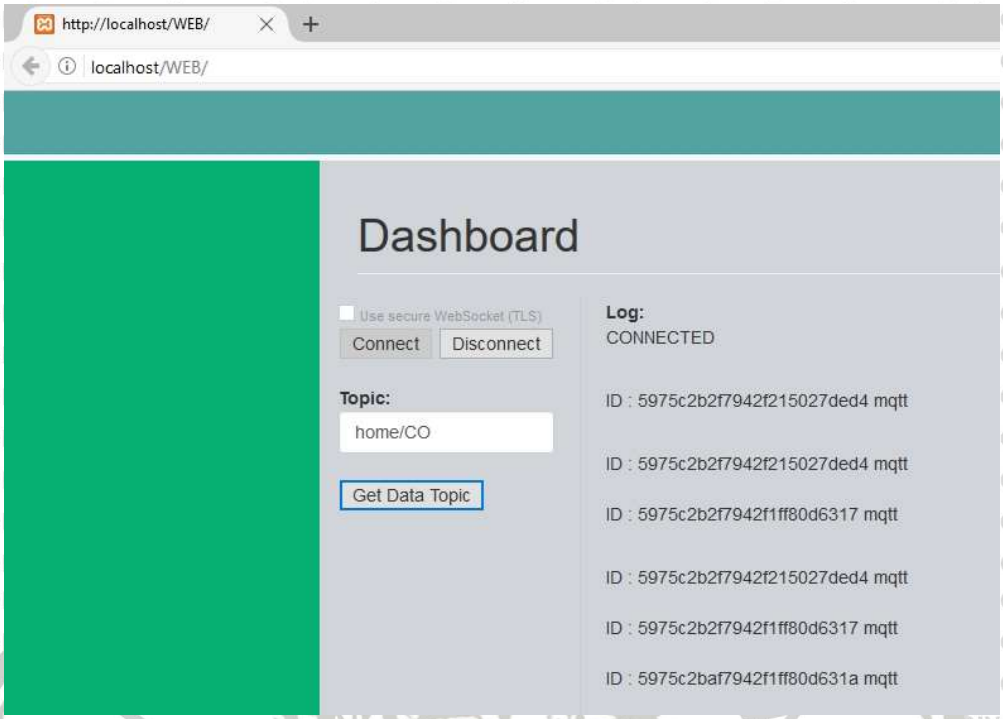
	App
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device* 4. Pengguna membuka dan menjalankan IoT App 5. Pengguna menginputkan topik yang akan ditampilkan pada IoT App
Hasil yang Diharapkan	IOT App dapat menerima permintaan menampilkan data berdasarkan topik, topik /Gambar, home/CO
Hasil Pengujian	IOT App berhasil menerima permintaan menampilkan data berdasarkan topik, topik /Gambar, home/CO

Gambar 6.15 menunjukkan IOT App dapat menerima permintaan menampilkan data gambar melalui topik /Gambar.



Gambar 6.15 Menampilkan Data Gambar pada IoT App

Gambar 6.16 menunjukkan IOT App dapat menerima permintaan menampilkan data karbon monoksida melalui topik home/CO.



Gambar 6.16 Menampilkan Data CO pada IoT App

6.1.10 Menampilkan Data pada IoT App

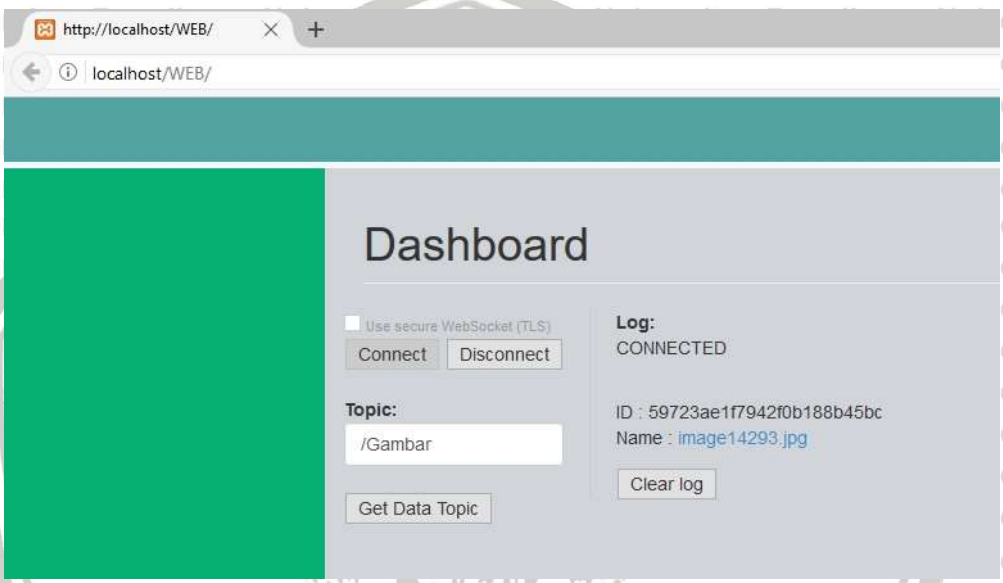
Kasus uji proses subscribe topik gambar dapat ditunjukkan pada Tabel 6.13

Tabel 6.13 Menampilkan Data pada IoT App

Kode	[DC_010]
Nama Kasus Uji	Menampilkan Data pada IoT App
Tujuan Pengujian	Menguji proses IoT App dalam menampilkan data berdasarkan topik, topik /Gambar, home/CO
Prosedur Pengujian	1. Middleware sudah berjalan 2. Pengguna memonitoring log pada middleware 3. Pengguna menjalankan *Internet Gateway Device* 4. Pengguna membuka dan menjalankan IoT App 5. Pengguna menginputkan topik yang akan ditampilkan pada IoT App 6. IoT App menampilkan data

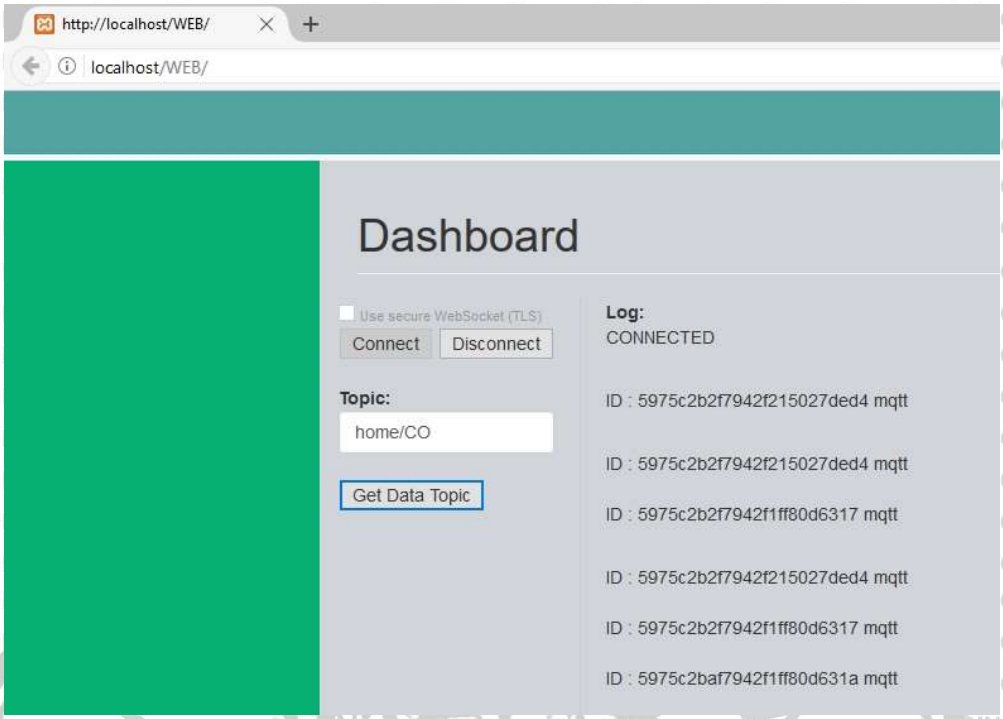
Hasil yang Diharapkan	IOT App dapat menampilkan data berdasarkan topik, topik /Gambar, home/CO
Hasil Pengujian	IOT App berhasil menampilkan data berdasarkan topik, topik /Gambar, home/CO

Gambar 6.17 menunjukkan IOT App dapat menerima permintaan menampilkan data gambar melalui topik /Gambar.



Gambar 6.17 Menampilkan Data Gambar pada IoT App

Gambar 6.18 menunjukkan IOT App dapat menerima permintaan menampilkan data karbon monoksida melalui topik home/CO.



Gambar 6.18 Menampilkan Data CO pada IoT App

6.1.11 Analisis Pengujian Fungsional

Berdasarkan pengujian yang sudah dilakukan sesuai perancangan pengujian fungsional pada bab 4, diperoleh hasil apakah seluruh fungsi dalam sistem yang dikembangkan dapat bekerja dengan sesuai. Pada Tabel 6.14 menunjukkan hasil pengujian fungsional.

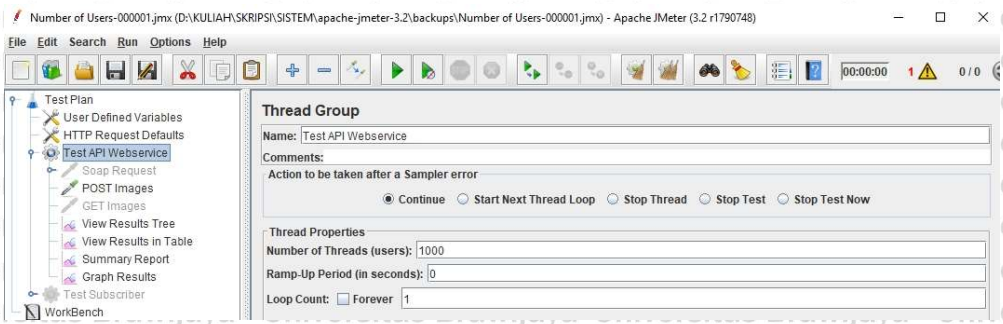
Tabel 6.14 Analisis hasil pengujian fungsional

Kode	Fungsi	Status
DC_001	*Internet Gateway Device* dapat men-subscribe topik dari *middleware*, yaitu topik /Gambar, home/CO	Valid
DC_002	*Internet Gateway Device* dapat mengambil data berdasarkan topik di *middleware,* yaitu topik /Gambar, home/CO	Valid
DC_003	*Internet Gateway Device* dapat mengirim data berdasarkan topik ke API, yaitu topik /Gambar, home/CO	Valid

DC_004	API web service dapat menerima data dari *Internet Gateway Device*	Valid
DC_005	API web service dapat mengirim data dari *Internet Gateway Device* ke GridFS	Valid
DC_006	API web service dapat menyimpan data dari *Internet Gateway Device* ke GridFS	Valid
DC_007	GridFS dapat menyimpan data sensor	Valid
DC_008	GridFS dapat menghapus data sensor	Valid
DC_009	IOT App dapat menerima permintaan menampilkan data berdasarkan topik, topik /Gambar, home/CO	Valid
DC_010	IOT App dapat menampilkan data berdasarkan topik, topik /Gambar, home/CO melalui protokol WebSocket	Valid

6.2 Pengujian Skalabilitas API *Web Service*

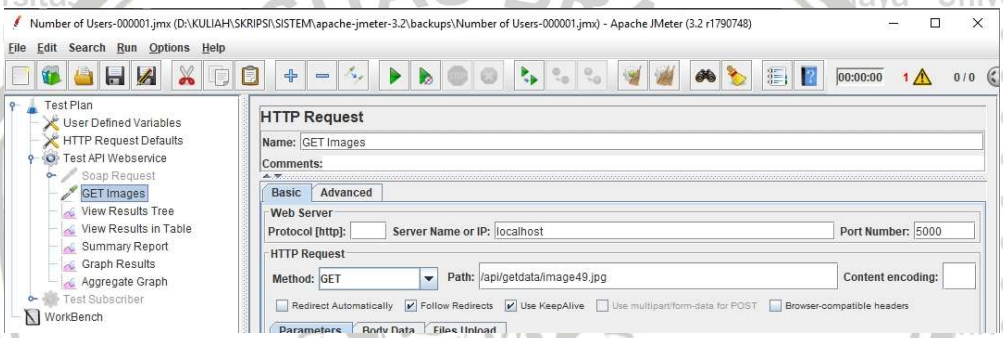
Dalam pengujian ini, ada sepuluh skenario untuk menguji fungsi GET dan GET pada *Web service*. Tujuan dari pengujian ini untuk mengetahui seberapa banyak request yang bisa diambil (GET) dalam sekali proses, dan seberapa banyak data yang bisa dikirim (POST) dalam sekali proses. Untuk pengujian ini menggunakan Apache JMeter yang ditunjukkan pada Gambar 6.19.



Gambar 6.19 Aplikasi Apache JMeter

6.2.1 Pengujian GET Data

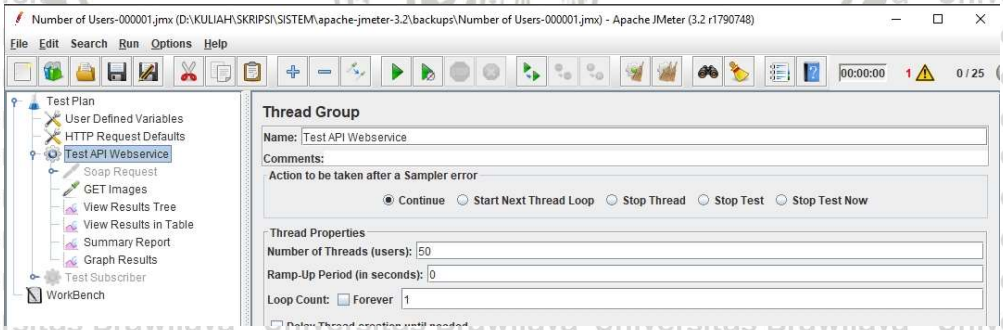
Dalam pengujian ini, penguji mencoba GET data yang sudah disimpan di *data storage* GridFS, dengan nama file image49.jpg. Gambar 6.20 menunjukkan scenario pengujian GET menggunakan JMeter.



Gambar 6.20 Skenario pengujian GET menggunakan JMeter

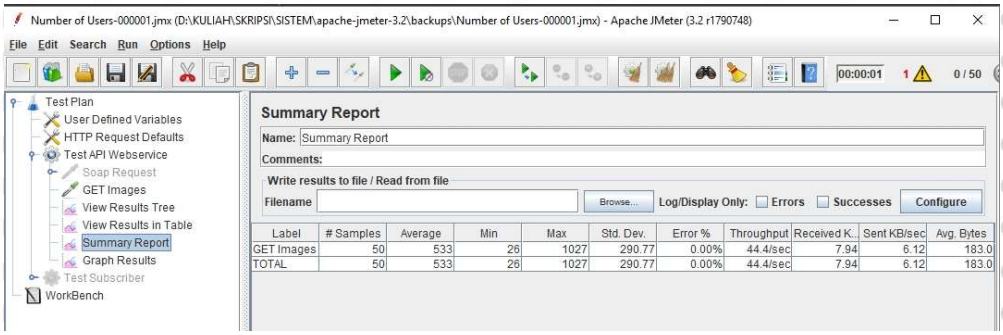
6.2.1.1 Skenario 1 : Dilakukan GET data sebanyak 50 request data

Pada skenario pertama, penguji membuat 50 request GET data ke *web service*. Gambar 6.21 menunjukkan 50 request.



Gambar 6.21 Skenario 50 request

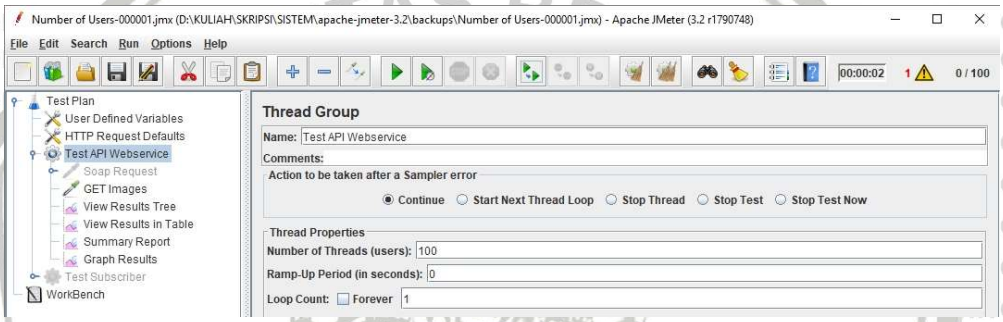
Dari pengujian 50 request get data didapatkan hasil bahwa seluruh data berhasil diambil dengan error 0%, dan throughput 44,4/sec. Gambar 6.22 menunjukkan hasil report tes 50 data.



Gambar 6.22 Report tes 50 request

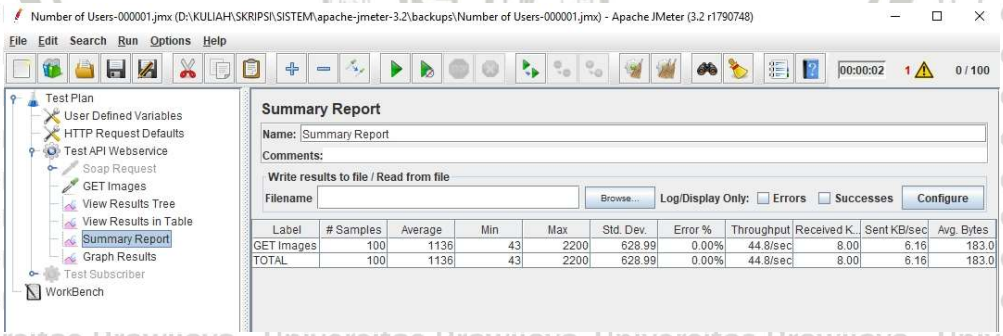
6.2.1.2 Skenario 2 : Dilakukan GET data sebanyak 100 request data

Pada skenario kedua, penguji membuat 100 request GET data ke *web service*. Gambar 6.23 menunjukkan 100 data yang direquest.



Gambar 6.23 Skenario 100 request

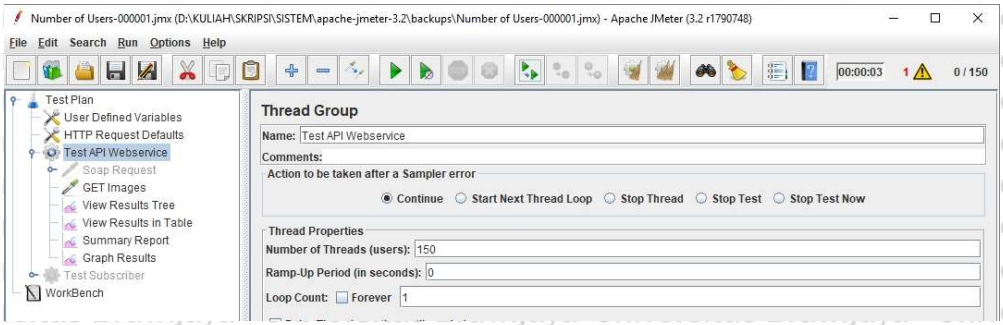
Dari pengujian 100 request get data didapatkan hasil bahwa seluruh data berhasil diambil dalam waktu 4 detik dengan error 0%, dan throughput 44,8/detik. Gambar 6.24 menunjukkan hasil report tes 100 data.



Gambar 6.24 Report tes 100 request

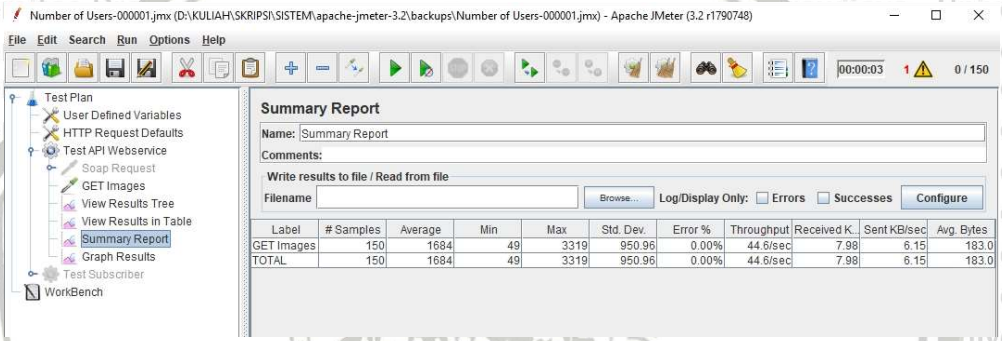
6.2.1.3 Skenario 3 : Dilakukan GET data sebanyak 150 request data

Pada skenario ketiga, penguji membuat 150 request GET data ke *web service*. Gambar 6.25 menunjukkan 150 data yang direquest.



Gambar 6.25 Skenario 150 request

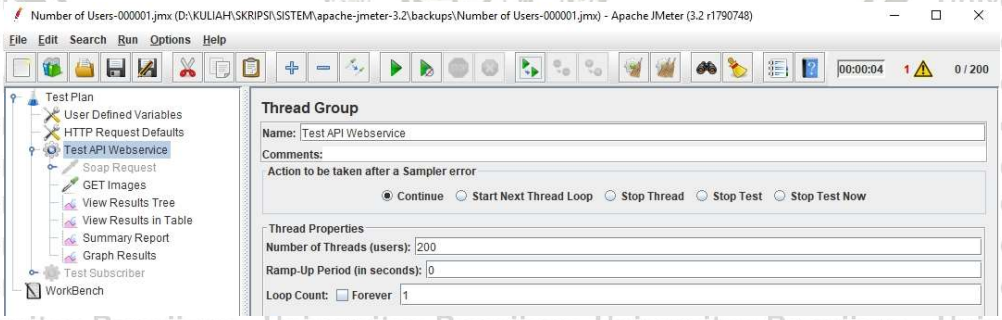
Dari pengujian 150 request get data didapatkan hasil bahwa seluruh data berhasil diambil dalam waktu 4 detik dengan error 0%, dan throughput 44,6/detik. Gambar 6.26 menunjukkan hasil report tes 150 data.



Gambar 6.26 Report tes 150 request

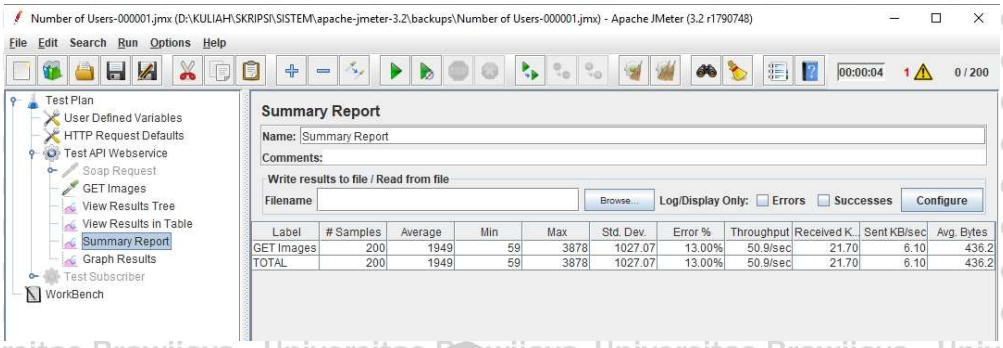
6.2.1.4 Skenario 4 : Dilakukan GET data sebanyak 200 request data

Pada skenario keempat, penguji membuat 200 request GET data ke *web service*. Gambar 6.27 menunjukkan 200 data yang direquest.



Gambar 6.27 Skenario 200 request

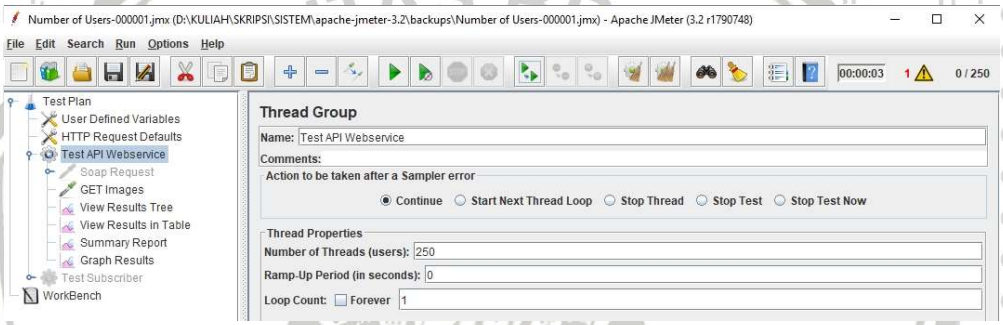
Dari pengujian 200 request get data didapatkan hasil bahwa seluruh data berhasil diambil dalam waktu 4 detik dengan error 13%, dan throughput 50,9/detik. Gambar 6.28 menunjukkan hasil report tes 200 data.



Gambar 6.28 Report test 200 request

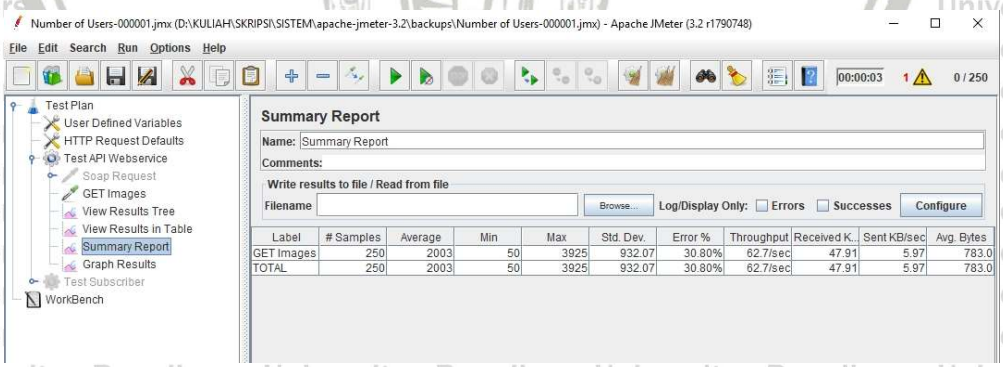
6.2.1.5 Skenario 5 : Dilakukan GET data sebanyak 250 request data

Pada skenario kelima, penguji membuat 250 request GET data ke *web service*. Gambar 6.29 menunjukkan 250 data yang direquest.



Gambar 6.29 Skenario 250 data request

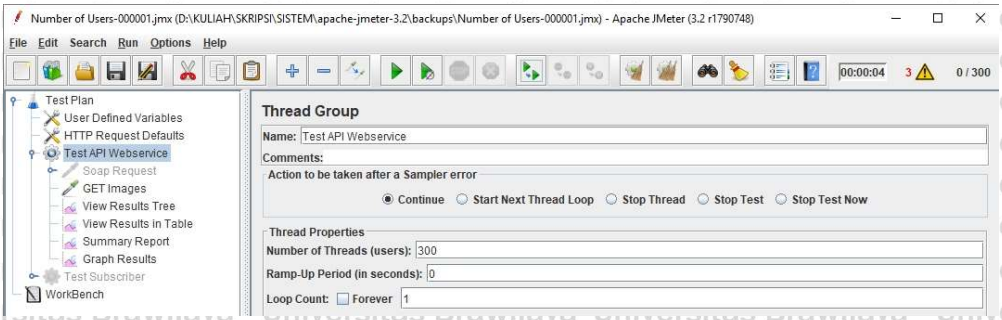
Dari pengujian 250 request get data didapatkan hasil bahwa seluruh data berhasil diambil dalam waktu 4 detik dengan error 30,80%, dan throughput 62,7/detik. Gambar 6.30 menunjukkan hasil report tes 250 data.



Gambar 6.30 Report test 250 data

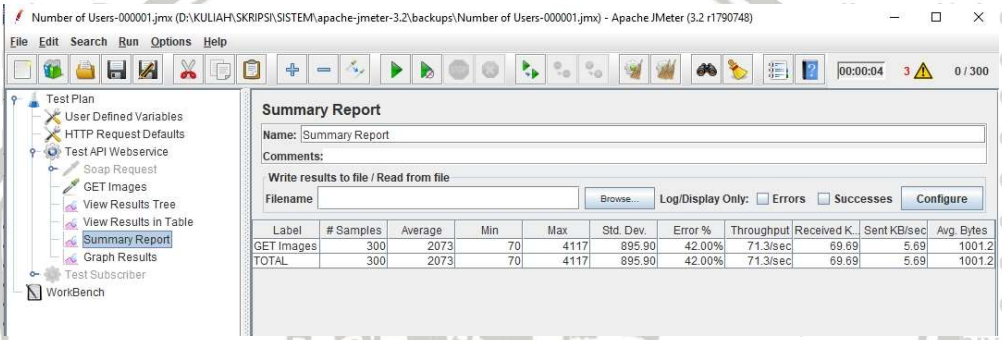
6.2.1.6 Skenario 6 : Dilakukan GET data sebanyak 300 request

Pada skenario keenam, penguji membuat 300 request GET data ke *web service*. Gambar 6.31 menunjukkan 300 data yang direquest.



Gambar 6.31 Skenario 300 request

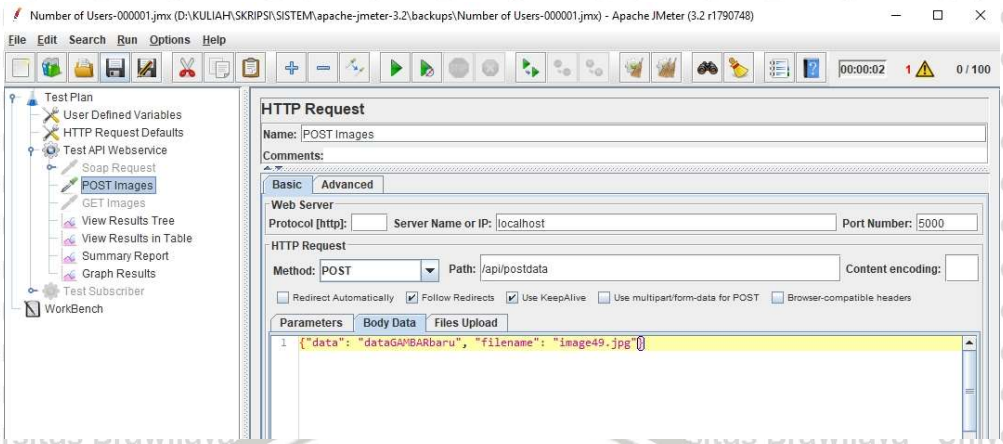
Dari pengujian 300 request get data didapatkan hasil bahwa seluruh data berhasil diambil dalam waktu 4 detik dengan error 42%, dan throughput 71,3/detik. Gambar 6.32 menunjukkan hasil report tes 300 data.



Gambar 6.32 Report test 300 request

6.2.2 Pengujian POST Data

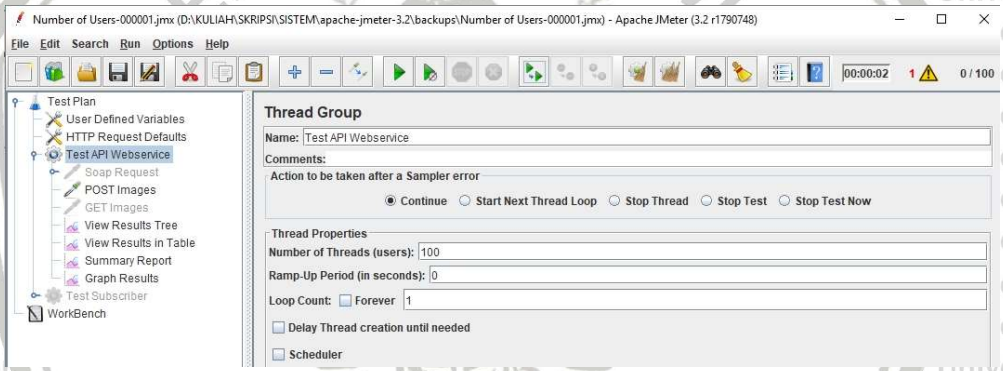
Dalam pengujian ini, penguji mencoba POST data ke *web service*, data yang dikirim dengan format json, yaitu data dan nama file seperti pada Gambar 6.33.



Gambar 6.33 Skenario pengujian POST menggunakan JMeter

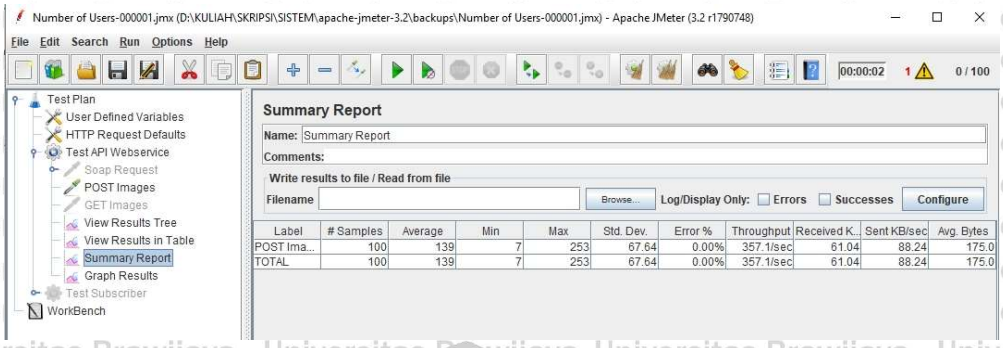
6.2.2.1 Skenario 1 : Dilakukan POST data sebanyak 100 kali

Pada skenario pertama, penguji membuat 100 kali POST data dalam sekali proses ke *web service*. Gambar 6.34 menunjukkan 100 post.



Gambar 6.34 Skenario 100 post

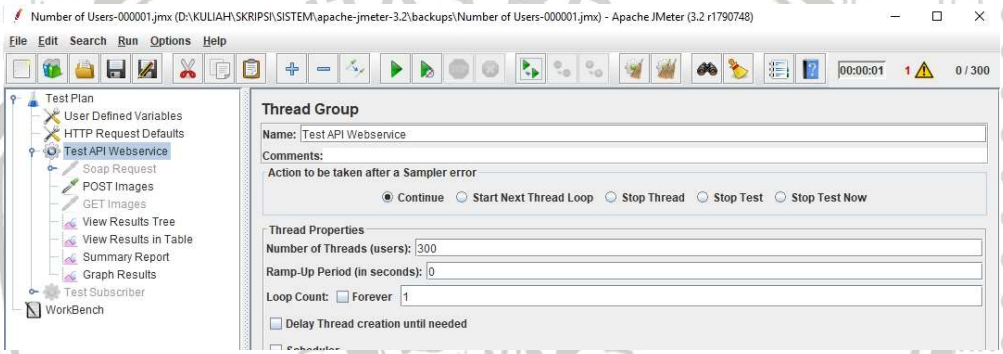
Dari pengujian 100 kali post data dalam sekali proses didapatkan hasil bahwa seluruh data berhasil diambil dengan error 0%, dan throughput 357,1/sec. Gambar 6.35 menunjukkan hasil report tes 100 data.



Gambar 6.35 Report tes 100 post

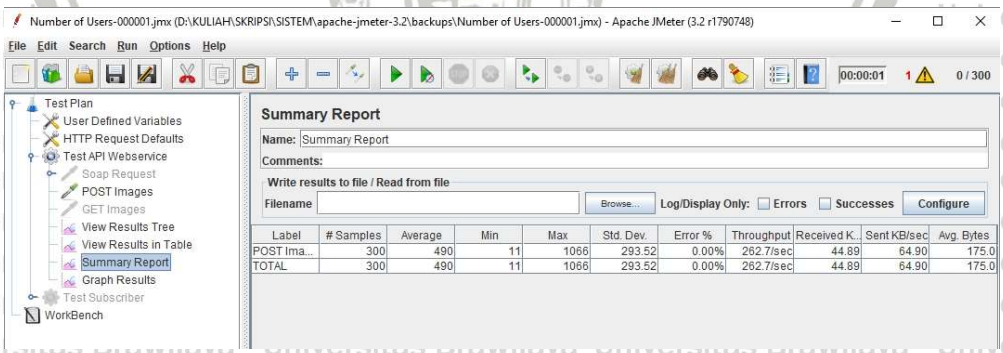
6.2.2.2 Skenario 2 : Dilakukan POST data sebanyak 300 kali

Pada skenario kedua, penguji membuat 300 kali POST data dalam sekali proses ke *web service*. Gambar 6.36 menunjukkan 300 post.



Gambar 6.36 Skenario 300 post

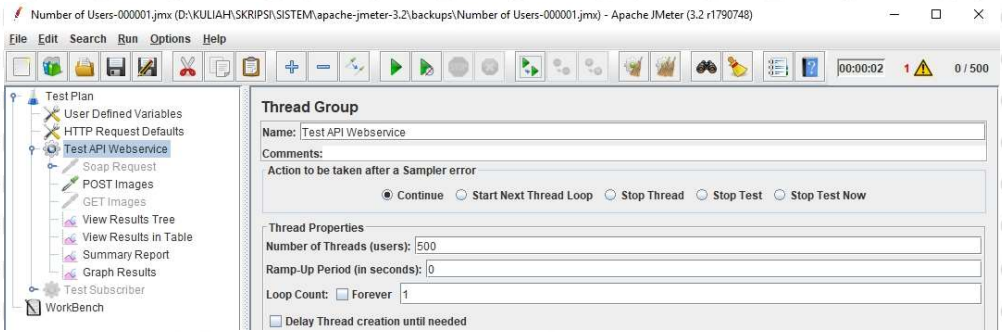
Dari pengujian 300 kali post data dalam sekali proses didapatkan hasil bahwa seluruh data berhasil diambil dengan error 0%, dan throughput 262,7/sec. Gambar 6.37 menunjukkan hasil report tes 300 data.



Gambar 6.37 Report tes 300 post

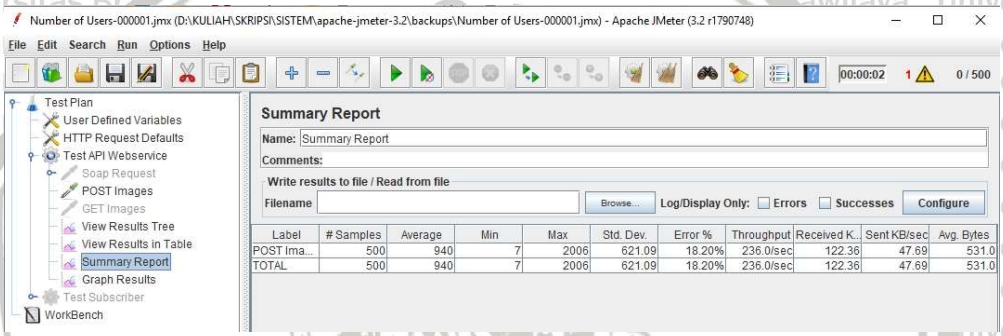
6.2.2.3 Skenario 3 : Dilakukan POST data sebanyak 500 kali

Pada skenario ketiga, penguji membuat 500 kali POST data dalam sekali proses ke *web service*. Gambar 6.38 menunjukkan 500 post.



Gambar 6.38 Skenario 500 post

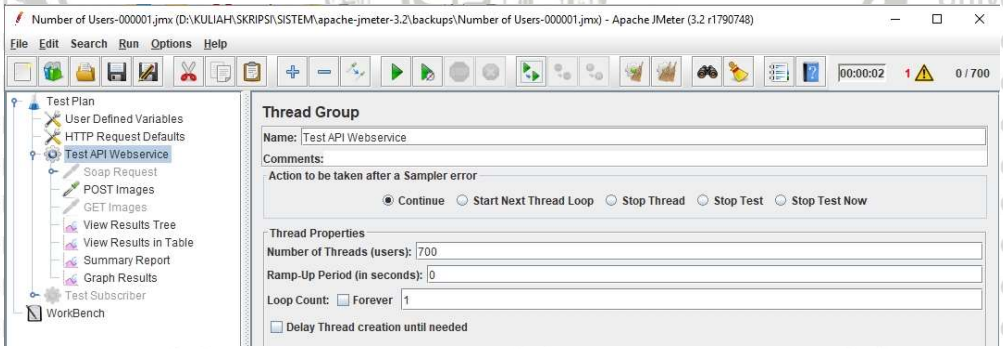
Dari pengujian 500 kali post data dalam sekali proses didapatkan hasil bahwa seluruh data berhasil diambil dengan error 18,20%, dan throughput 236/sec. Gambar 6.39 menunjukkan hasil report tes 500 data.



Gambar 6.39 Report tes 500 post

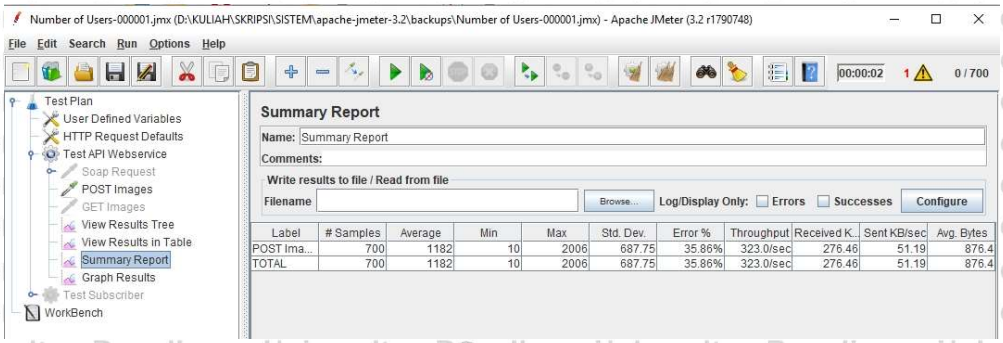
6.2.2.4 Skenario 4 : Dilakukan POST data sebanyak 700 kali

Pada skenario keempat, penguji membuat 700 kali POST data dalam sekali proses ke *web service*. Gambar 6.40 menunjukkan 700 post.



Gambar 6.40 Skenario 700 post

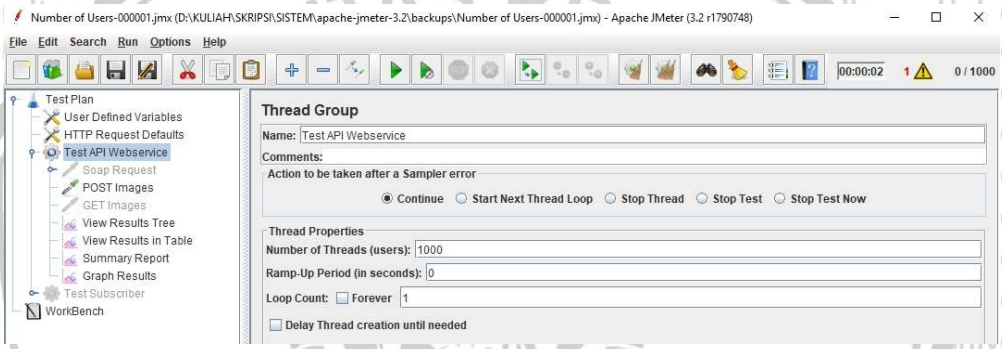
Dari pengujian 700 kali post data dalam sekali proses didapatkan hasil bahwa seluruh data berhasil diambil dengan error 35,86%, dan throughput 323/sec. Gambar 6.41 menunjukkan hasil report tes 700 data.



Gambar 6.41 Report tes 700 post

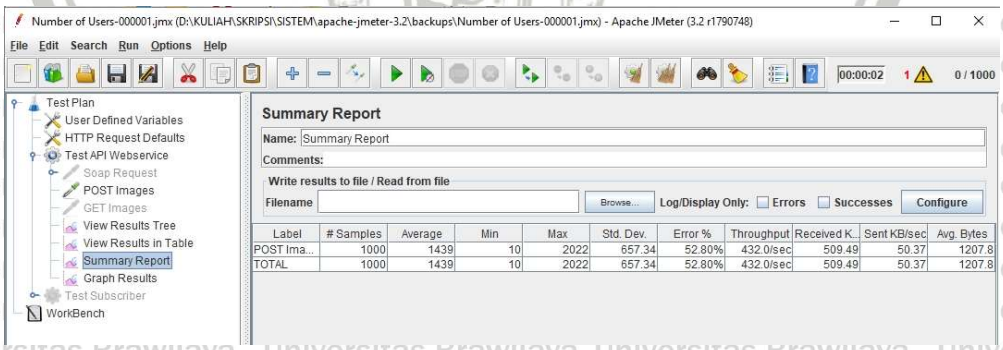
6.2.2.5 Skenario 5 : Dilakukan POST data sebanyak 1000 kali

Pada skenario ketiga, penguji membuat 1000 kali POST data dalam sekali proses ke *web service*. Gambar 6.42 menunjukkan 1000 post.



Gambar 6.42 Skenario 1000 post

Dari pengujian 1000 kali post data dalam sekali proses didapatkan hasil bahwa seluruh data berhasil diambil dengan error 52,20%, dan throughput 432/sec. Gambar 6.43 menunjukkan hasil report tes 1000 data.



Gambar 6.43 Report tes 1000 post

6.2.3 Analisis Pengujian Skalabilitas GET API *Web service*

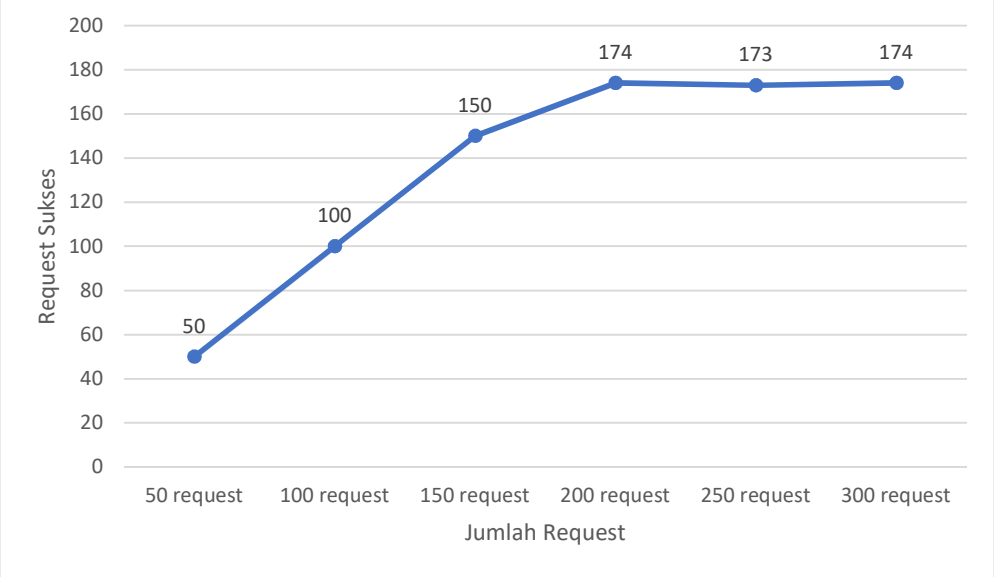
Pengujian skalabilitas get api web service dilakukan dengan membuat request get ke web service. Pengujian ini menggunakan beberapa skenario, yaitu skenario 1 melakukan 50 request dalam satu waktu, skenario 2 melakukan 100

request dalam satu waktu, skenario 3 melakukan 150 request dalam satu waktu, skenario 4 melakukan 200 request dalam satu waktu, skenario 5 melakukan 250 request dalam satu waktu, dan skenario 6 melakukan 300 request dalam satu waktu. Tabel 6.15 menunjukkan hasil dari pengujian skalabilitas get API.

Tabel 6.15 Skalabilitas GET API *web service*

No.	Pengujian	Request Sukses	Request Error	Presentase Sukses (%)
1	50 request	50	0	100
2	100 request	100	0	100
3	150 request	150	0	100
4	200 request	174	26	87
5	250 request	173	77	69,2
6	300 request	174	126	58

Pada Gambar 6.44 menunjukkan grafik dari hasil analisis pada pengujian skalabilitas get API. Untuk 50, 100, dan 150 request berhasil dilakukan. Sedangkan untuk 200, 250, dan 300 request, hanya 174, 173, dan 174 request yang berhasil.



Gambar 6.44 Grafik analisis pengujian skalabilitas GET API *web service*

6.2.4 Analisis Pengujian Skalabilitas POST API *Web service*

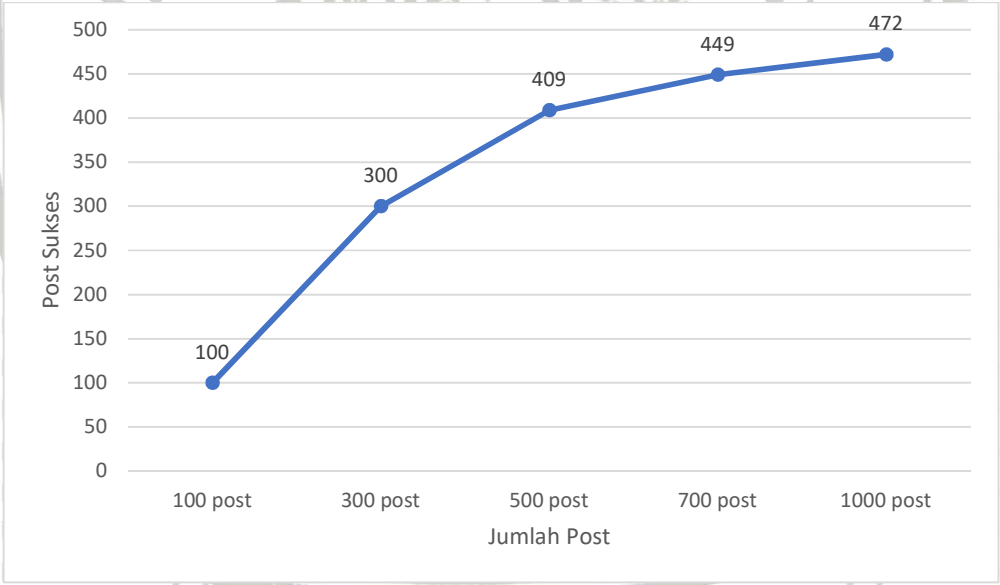
Pengujian skalabilitas post api web service dilakukan dengan membuat request post ke web service. Pengujian ini menggunakan beberapa skenario, yaitu skenario 1 melakukan 100 request post dalam satu waktu, skenario 2 melakukan 300 request post dalam satu waktu, skenario 3 melakukan 500

request post dalam satu waktu, skenario 4 melakukan 700 request post dalam satu waktu, dan skenario 5 melakukan 1000 request post dalam satu waktu. Tabel 6.16 menunjukkan hasil dari pengujian skalabilitas post API.

Tabel 6.16 Skalabilitas POST API *web service*

No.	Pengujian	Post Sukses	Post Error	Presentase Sukses (%)
1	100 post	100	0	100
2	300 post	300	0	100
3	500 post	409	91	100
4	700 post	449	251	87
5	1000 post	472	528	69,2

Pada Gambar 6.45 menunjukkan grafik dari hasil analisis pada pengujian skalabilitas post API. Untuk 100, dan 300 request post berhasil dilakukan. Sedangkan untuk 500, 700, dan 1000 request post, hanya 409, 449, dan 472 request yang berhasil.



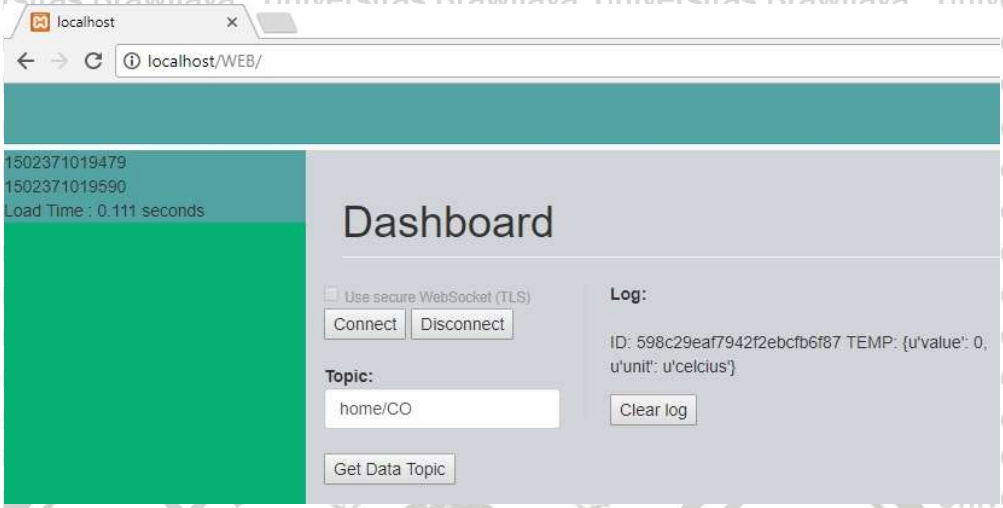
Gambar 6.45 Grafik analisis pengujian skalabilitas POST API *web service*

6.3 Pengujian *Response Time*

Pengujian *response time* dilakukan dengan menampilkan data pada *IoT Application* pada setiap topik yang ada pada MongoDB dan GridFS. Pengujian ini menggunakan beberapa skenario, yaitu skenario 1 melakukan 1 get data, skenario 2 melakukan 10 get data, skenario 3 melakukan 30 get data, dan skenario 4 melakukan 50 get data.

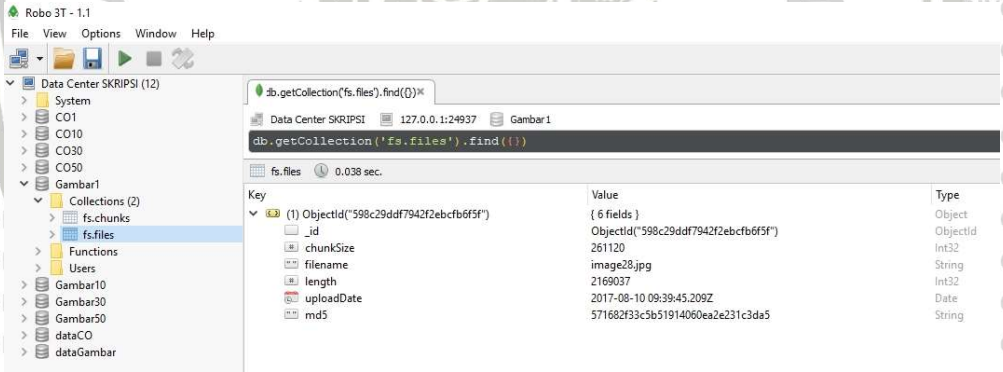
6.3.1 Skenario 1: Get 1 Data

Gambar 6.46 menunjukkan data yang tersimpan dalam MongoDB pada skenario 1 ini yaitu 1 data CO.



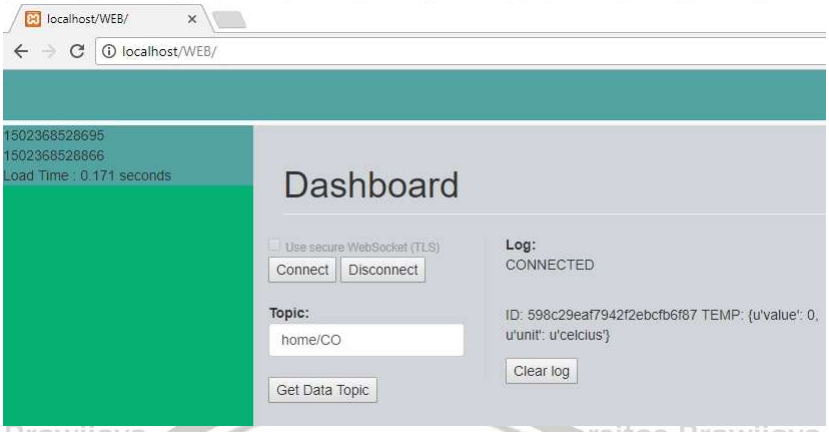
Gambar 6.46 Data 1 CO pada MongoDB

Gambar 6.47 menunjukkan data yang tersimpan dalam GridFS pada skenario 1 ini yaitu 1 data Gambar.

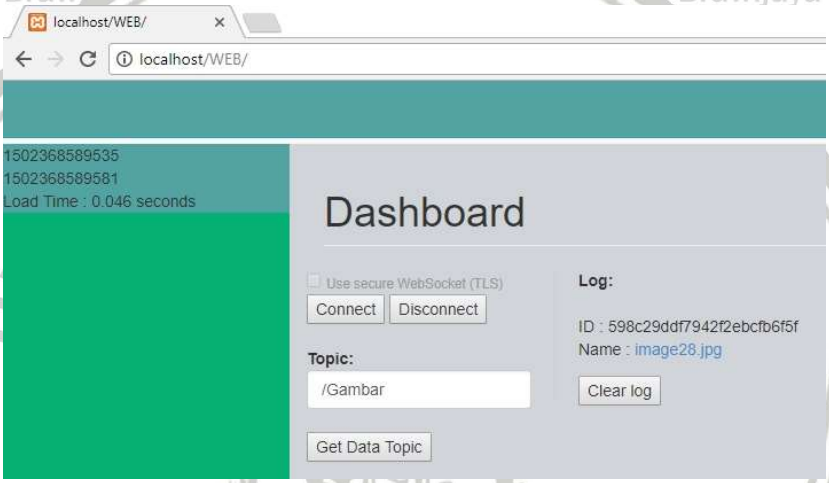


Gambar 6.47 Data 1 Gambar pada GridFS

Pada skenario pertama, dihasilkan *response time* untuk *get* dan menampilkan 1 data CO ialah 0,111 detik, dan 1 data gambar ialah 0,046 detik. Gambar 6.48 dan Gambar 6.49 menunjukkan hasil get 1 data.



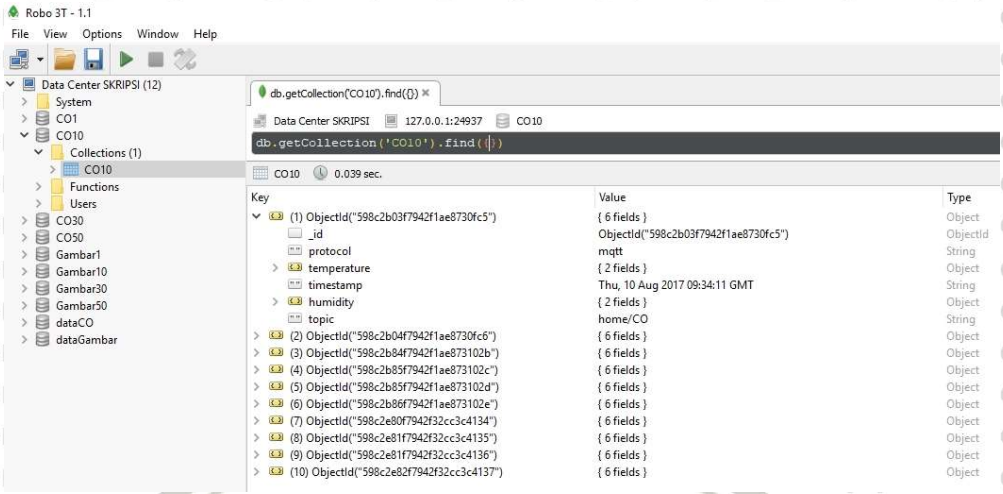
Gambar 6.48 Tampilan dan hasil *get* pada 1 data CO



Gambar 6.49 Tampilan dan hasil *get* pada 1 data Gambar

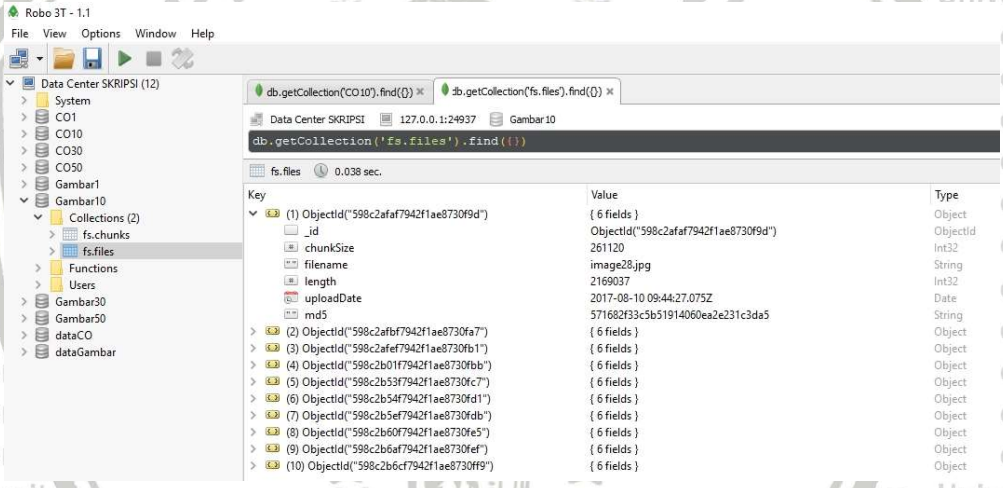
6.3.2 Skenario 2: Get 10 Data

Gambar 6.50 menunjukkan data yang tersimpan dalam MongoDB pada skenario 2 ini yaitu 10 data CO.



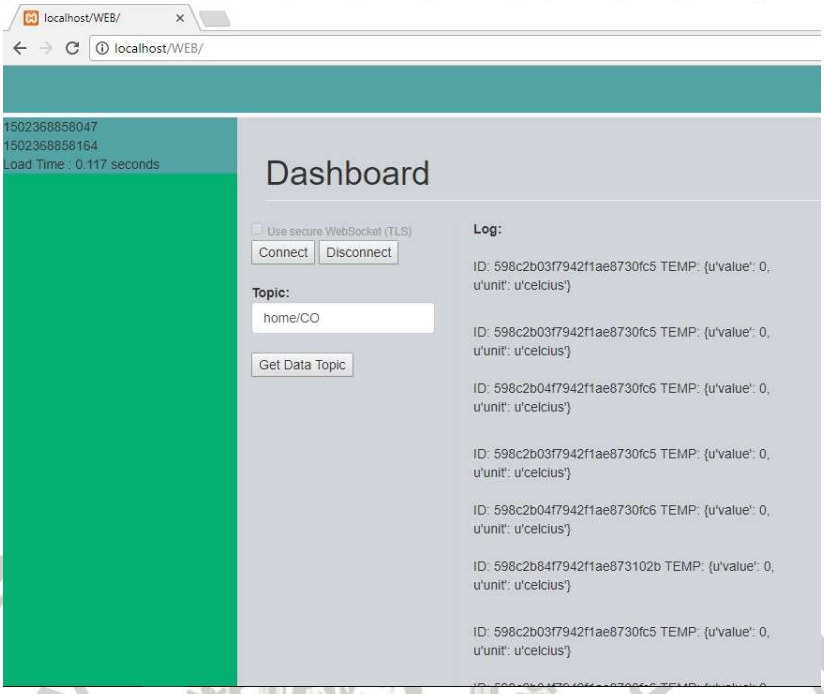
Gambar 6.50 Data 10 CO pada MongoDB

Gambar 6.51 menunjukkan data yang tersimpan dalam GridFS pada skenario 2 ini yaitu 10 data Gambar.

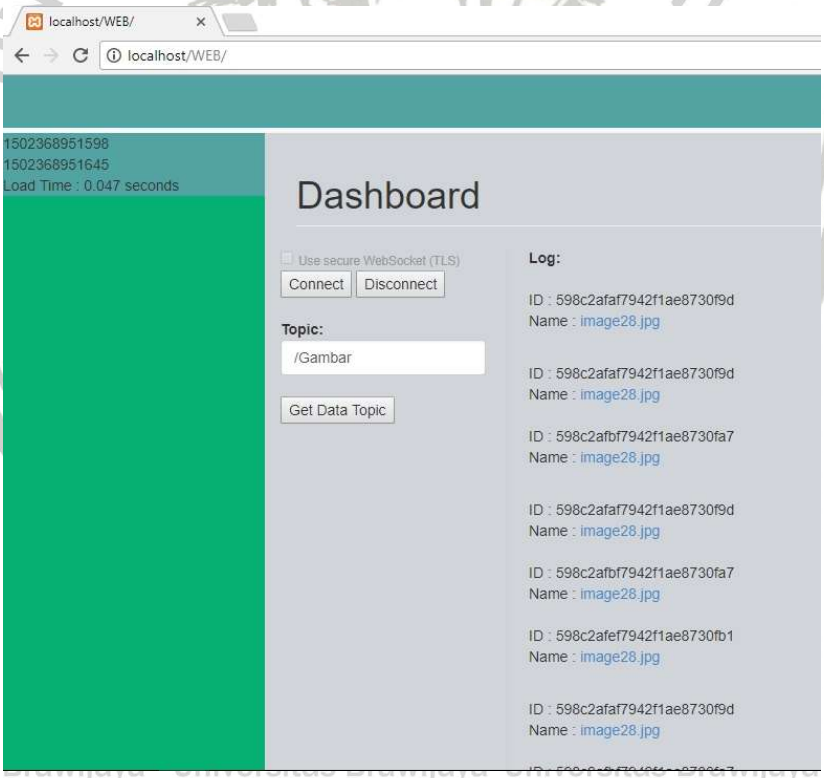


Gambar 6.51 Data 10 Gambar pada GridFS

Pada skenario kedua, dihasilkan *response time* untuk *get* dan menampilkan 10 data CO ialah 0,117 detik, dan 10 data gambar ialah 0,047 detik. Gambar 6.52 dan Gambar 6.53 menunjukkan hasil get 10 data.



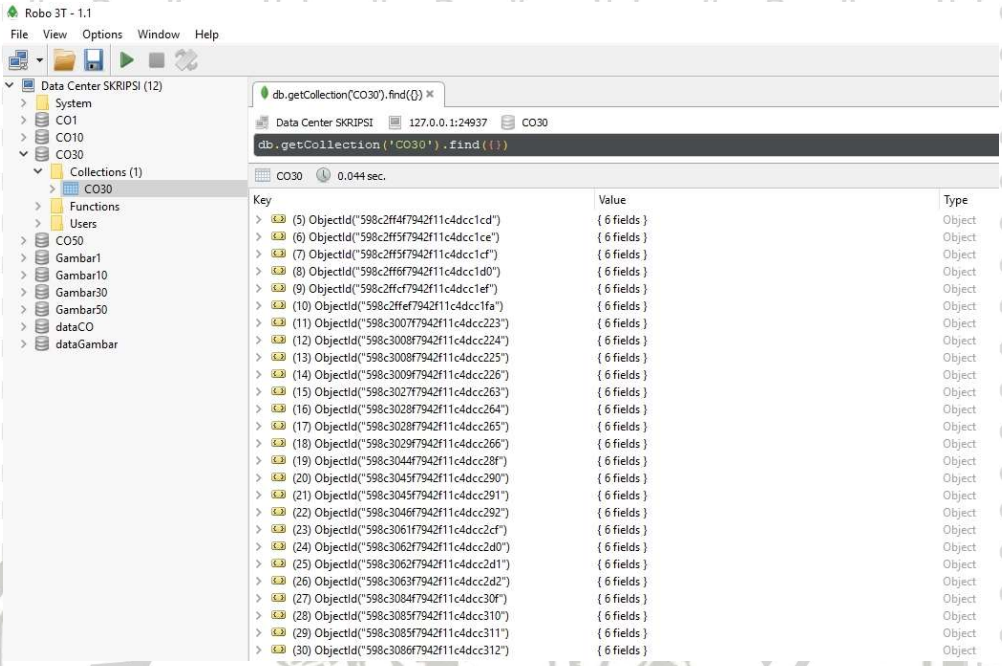
Gambar 6.52 Tampilan dan hasil *get* pada 10 data CO



Gambar 6.53 Tampilan dan hasil *get* pada 10 data Gambar

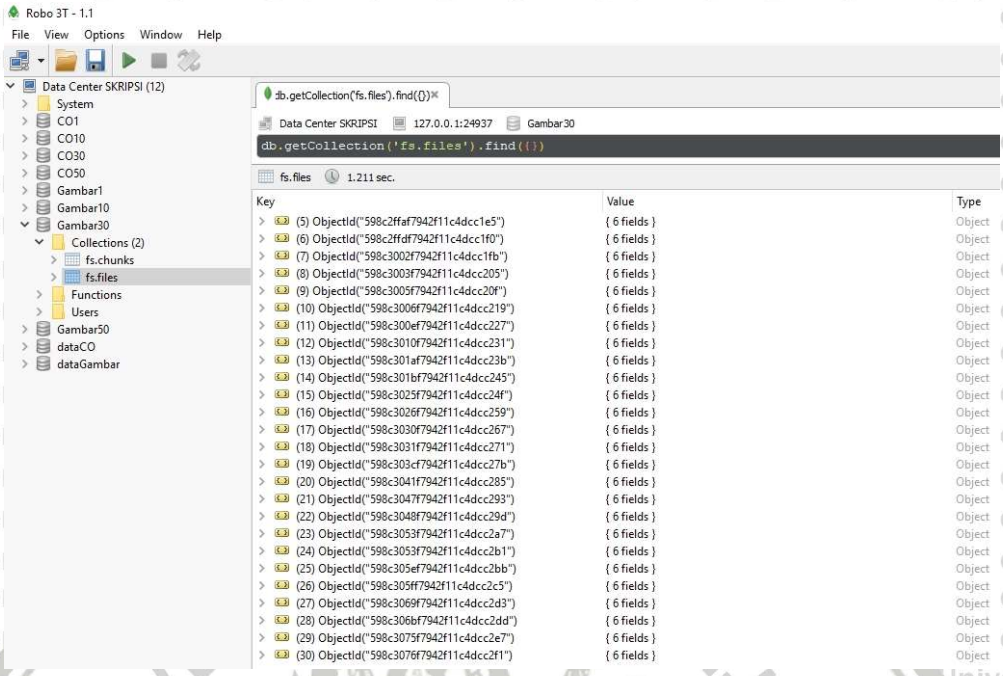
6.3.3 Skenario 3: Get 30 Data

Gambar 6.54 menunjukkan data yang tersimpan dalam MongoDB pada skenario 3 ini yaitu 30 data CO.



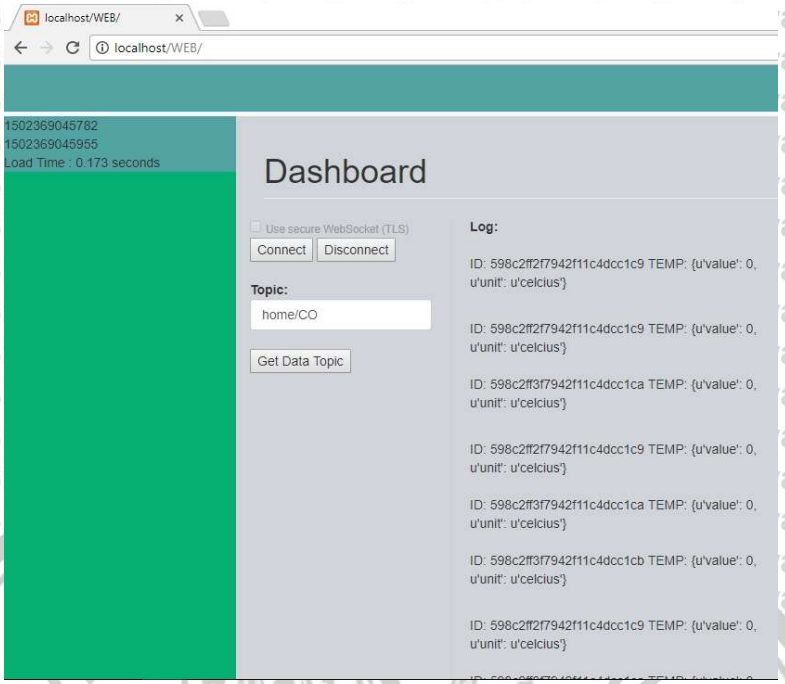
Gambar 6.54 Data 30 CO pada MongoDB

Gambar 6.55 menunjukkan data yang tersimpan dalam GridFS pada skenario 3 ini yaitu 30 data Gambar.

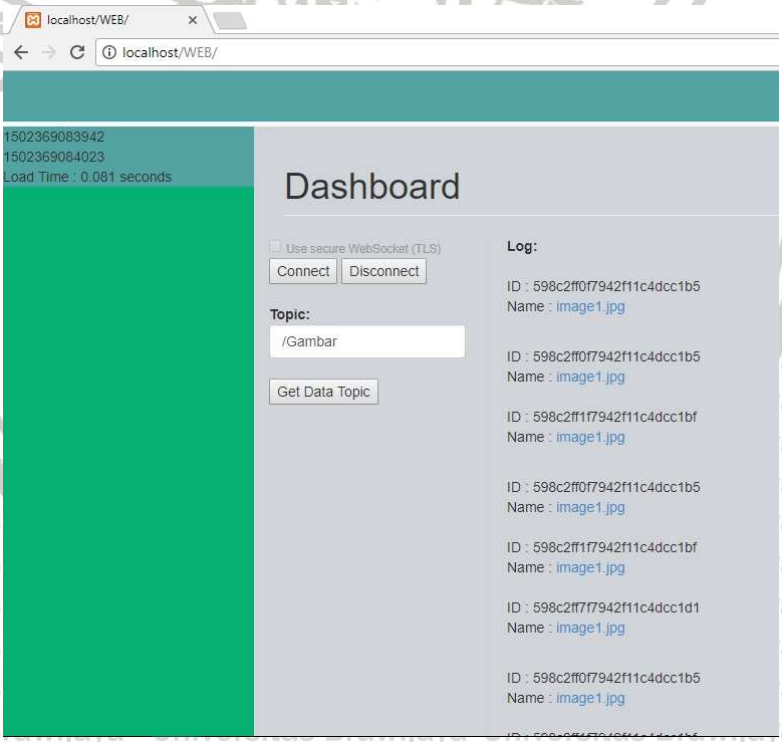


Gambar 6.55 Data 30 Gambar pada GridFS

Pada skenario ketiga, dihasilkan *response time* untuk *get* dan menampilkan 30 data CO ialah 0,173 detik, dan 30 data gambar ialah 0,081 detik. Gambar 6.56 dan Gambar 6.57 menunjukkan hasil get 30 data.



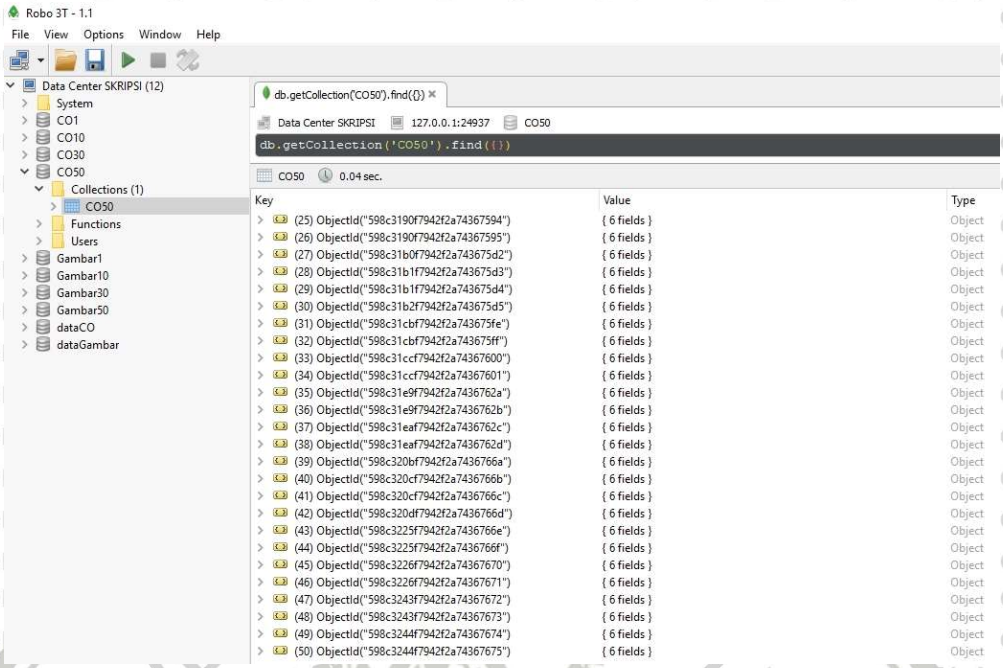
Gambar 6.56 Tampilan dan hasil *get* pada 30 data CO



Gambar 6.57 Tampilan dan hasil *get* pada 30 data Gambar

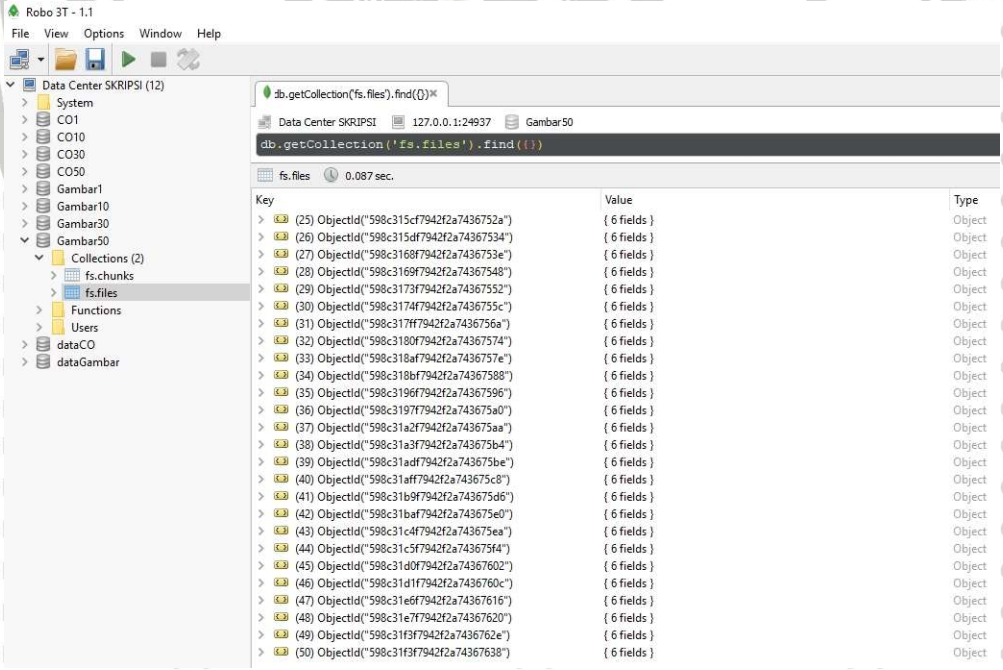
6.3.4 Skenario 4: Get 50 Data

Gambar 6.58 menunjukkan data yang tersimpan dalam MongoDB pada skenario 4 ini yaitu 50 data CO.



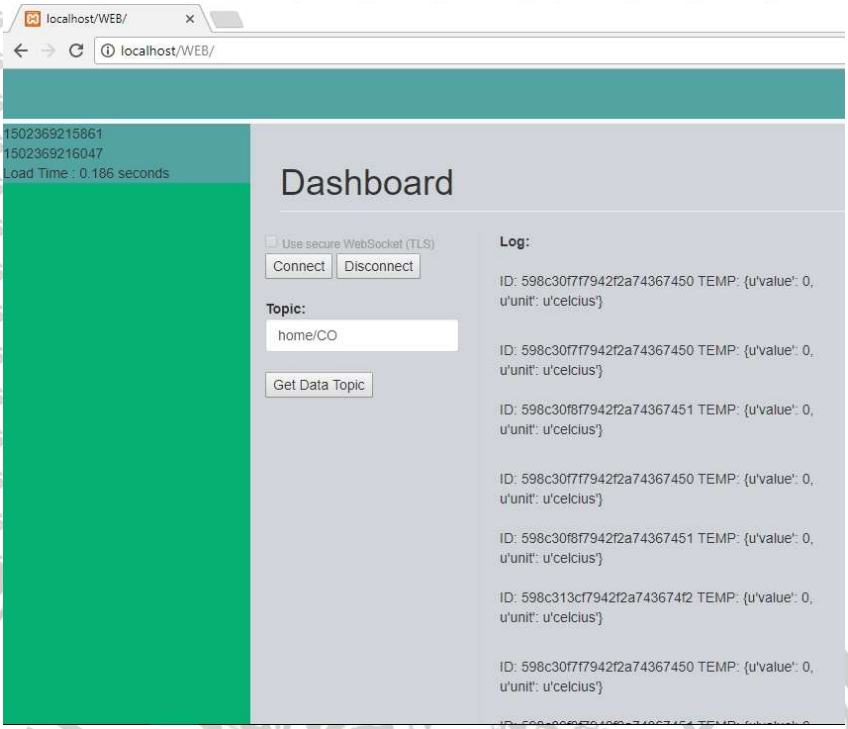
Gambar 6.58 Data 50 CO pada MongoDB

Gambar 6.59 menunjukkan data yang tersimpan dalam GridFS pada skenario 4 ini yaitu 50 data Gambar.

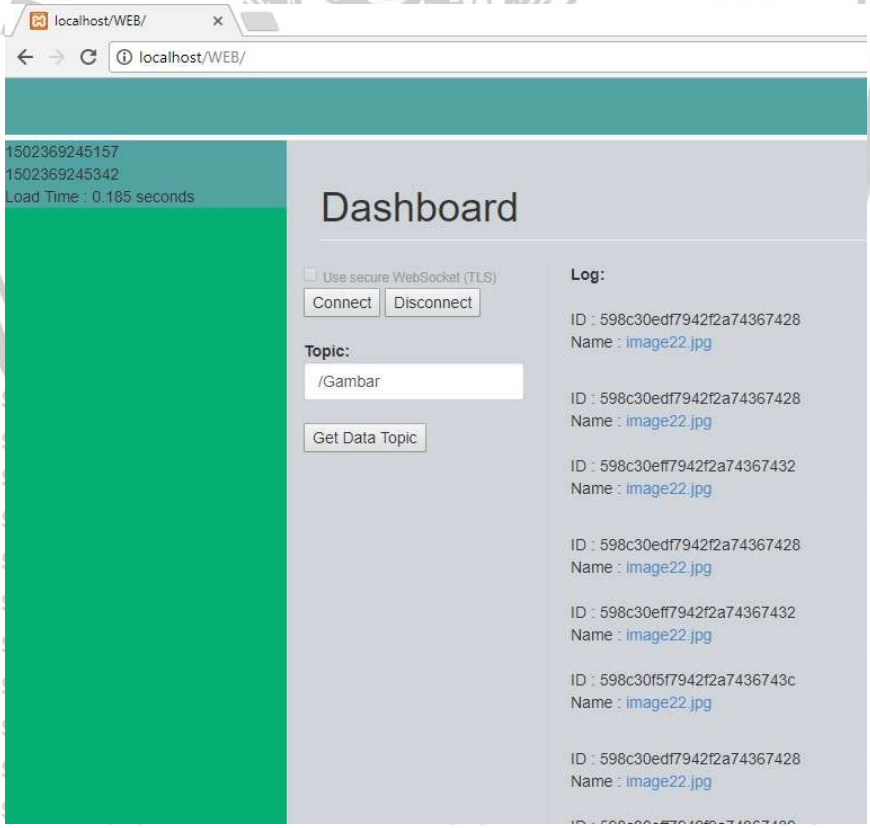


Gambar 6.59 Data 50 Gambar pada GridFS

Pada skenario keempat, dihasilkan *response time* untuk *get* dan menampilkan 50 data CO ialah 0,186 detik, dan 50 data gambar ialah 0,185 detik. Gambar 6.60 dan Gambar 6.61 menunjukkan hasil get 50 data.



Gambar 6.60 Tampilan dan hasil *get* pada 50 data CO



Gambar 6.61 Tampilan dan hasil *get* pada 50 data Gambar

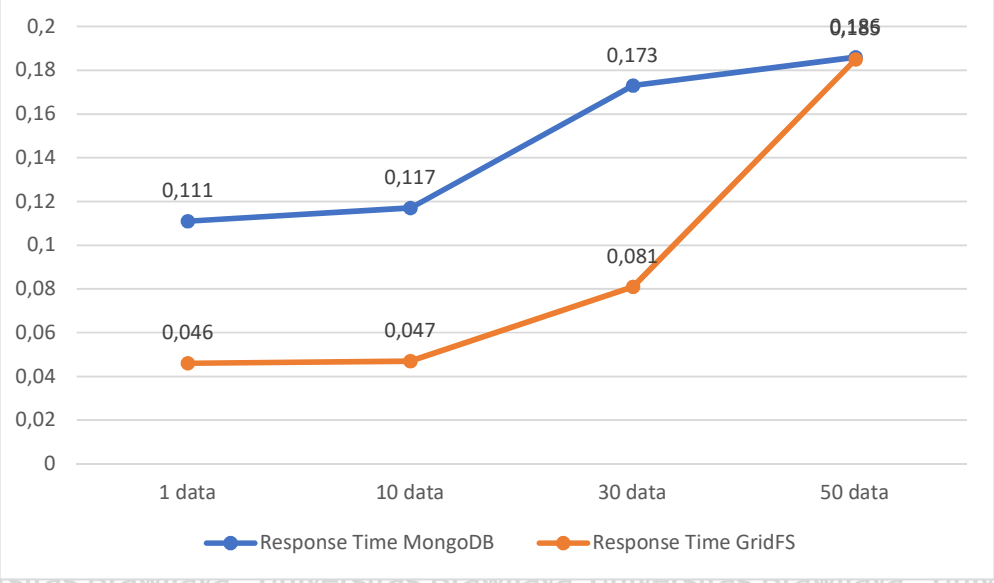
6.3.5 Analisis Pengujian *Response Time*

Pengujian *response time* dilakukan dengan menampilkan data pada *IoT Application* dengan mengambil setiap topik yang ada pada GridFS. Pengujian ini menggunakan beberapa skenario, yaitu skenario 1 melakukan 1 get data, skenario 2 melakukan 10 get data, skenario 3 melakukan 30 get data, dan skenario 4 melakukan 50 get data. Data yang digunakan dari data yang berasal dari MongoDB dan GridFS yang berupa BSON (Binary JSON). Tabel 6.17 menunjukkan hasil pengujian *response time*.

Tabel 6.17 Hasil pengujian *response time*

Get Data	Response Time MongoDB (seconds)	Response Time GridFS (seconds)
1	0,111	0,046
10	0,117	0,047
30	0,173	0,081
50	0,186	0,185

Pada Gambar 6.62 menunjukkan grafik dari hasil analisis pada pengujian *response time*. Untuk response time pada GridFS lebih cepat karena data yang diambil berupa chunks yang lebih kecil daripada data pada MongoDB.



Gambar 6.62 Hasil *response time*

Pengukuran dilakukan IoT App yang merupakan web, sehingga ada beberapa faktor yang mempengaruhi pengujian *response time* ini, antara lain *transmission delay, delay proccesing/query*.

6.4 Analisis Performa MongoDB

6.4.1 Operasi yang Berjalan Pada MongoDB

Untuk mengevaluasi performa dari operasi yang berjalan saat ini menggunakan perintah db.currentOp() pada mongo *shell*. Terdapat beberapa status pada operasi ini, yaitu *waitingForLock*, *active*, *inprog*. Dalam Gambar 6.63 statusnya *inprog* yang menandakan bahwa operasi pengindeksan dalam keadaan aktif dengan berbagai keterangan.

Gambar 6.63 Tampilan operasi yang berjalan pada MongoDB

6.4.2 Analisa Performa MongoDB

Perintah serverStatus memberikan gambaran umum tentang status proses pada *database*. Saat kita mengembangkan dan mengoperasikan aplikasi dengan MongoDB, kita perlu menganalisis kinerja aplikasi dan *database*-nya. Pada Gambar 6.64 terdapat penjelasan dari host dengan perintah db.serverStatus().host, versi dari MongoDb dengan perintah db.serverStatus().version, proses yang berjalan pada sistem dengan perintah db.serverStatus().process, lama proses mongo aktif dalam sistem dengan perintah db.serverStatus().uptime.

```
> db.serverStatus().host
ega
> db.serverStatus().version
3.2.16
> db.serverStatus().process
mongod
> db.serverStatus().uptime
588492
```

Gambar 6.64 Analisa server status *host, version, process, uptime*

Perintah db.serverStatus().locks berisi sebuah dokumen yang menyediakan laporan untuk setiap jenis dan model lock. Diantaranya lock global, database, collection, dan metadata.

```
> db.serverStatus().locks
{
        "Global" : {
                "acquireCount" : {
                        "r" : NumberLong(6135239),
                        "w" : NumberLong(1414),
                        "W" : NumberLong(3)
                }
        },
        "Database" : {
                "acquireCount" : {
                        "r" : NumberLong(3066874),
                        "w" : NumberLong(1388),
                        "R" : NumberLong(37),
                        "W" : NumberLong(26)
                }
        },
        "Collection" : {
                "acquireCount" : {
                        "r" : NumberLong(3135312),
                        "w" : NumberLong(1392)
                }
        },
        "Metadata" : {
                "acquireCount" : {
                        "w" : NumberLong(1)
                }
        }
}
```

Gambar 6.65 Analisa server status *locks*

Perintah db.serverStatus().globalLock berisi informasi mengenai status kunci database saat ini, antrian operasi saat ini, dan jumlah klien aktif. Pada Gambar 6.66, nilai totalTime mewakili waktu dalam mikrodetik, dihitung dari database terakhir dimulai dan membuat globalLock. Hal ini sama dengan total server uptime. Nilai CurrentQueue memberikan informasi mengenai jumlah operasi yang diantrikan, nilai activeClients memberikan informasi tentang jumlah klien yang terhubung dan jenis operasi, misal baca atau tulis yang dilakukan oleh klien ini.

```
> db.serverStatus().globalLock
{
        "totalTime" : NumberLong("588542295000"),
        "currentQueue" : {
                "total" : 0,
                "readers" : 0,
                "writers" : 0
        },
        "activeClients" : {
                "total" : 8,
                "readers" : 0,
                "writers" : 0
        }
}
```

Gambar 6.66 Analisa server status *global lock*

Perintah db.serverStatus().mem memberikan informasi mengenai arsitektur sistem dan penggunaan memori saat ini. Nilai bits tergantung pada arsitektur system, nilai resident menjelaskan penggunaan RAM dalam proses database dalam satuan megabytes (MB), nilai virtual menjelaskan penggunaan memori virtual pada database MongoDb dalam satuan megabyte, nilai support true menjelaskan bahwa sistem mendukung informasi memori jika ditambah/diperluas, nilai mapped menjelaskan seberapa banyak memeori yang dipetakan pada MongoDb.

```
> db.serverStatus().mem
{
        "bits" : 64,
        "resident" : 309,
        "virtual" : 508,
        "supported" : true,
        "mapped" : 0,
        "mappedWithJournal" : 0
}
```

Gambar 6.67 Analisa server status memori

Perintah db.serverStatus().connections menjelaskan mengenai status koneksi yang masuk saat ini dan ketersediaan *server database*. Nilai ini dapat digunakan untuk menilai persyaratan beban dan kapasitas saat ini dari server. Nilai current menjelaskan jumlah koneksi ke server database dari klien, nilai available menjelaskan jumlah koneksi masuk yang tidak terpakai yang disediakan oleh database, nilai totalCreated menjelaskan hitungan semua koneksi masuk yang dibuat ke server. Jumlah ini termasuk koneksi yang sudah ditutup.

```
> db.serverStatus().connections
{ "current" : 1, "available" : 51199, "totalCreated" : NumberLong(180) }
```

Gambar 6.68 Analisa server status koneksi

Perintah db.serverStatus().extra_info menjelaskan data tersimpan yang dikumpulkan oleh MongoDb tentang sistem. Nilai note menjelaskan bahwa data dalam struktur ini bergantung pada platform yang mendasarinya, nilai heap_usage_bytes menjelaskan ukuran total byte ruang heap yang digunakan

oleh proses basis data, nilai page_faults menjelaskan jumlah total kesalahan halaman yang memerlukan operasi disk. Kesalahan halaman mengacu pada operasi yang memerlukan server database untuk mengakses data yang tidak tersedia dalam memori aktif. Penghitung halaman_fault dapat meningkat secara dramatis saat kondisi kinerja buruk dan mungkin berkorelasi dengan lingkungan memori yang terbatas dan kumpulan data yang lebih besar.

```
> db.serverStatus().extra_info
{
        "note" : "fields vary by platform",
        "heap_usage_bytes" : 278810824,
        "page_faults" : 215
}
```

Gambar 6.69 Analisa server status info

Perintah db.serverStatus().network menjelaskan tentang data penggunaan jaringan MongoDB. Nilai bytesIn menjelaskan jumlah lalu lintas jaringan dalam bytes yang diterima oleh database. Nilai bytesOut menjelaskan jumlah lalu lintas jaringan, dalam bytes yang dikirim dari database ini. Nilai ini digunakan untuk memastikan bahwa lalu lintas jaringan yang dikirim ke proses mongod sesuai dengan harapan dan lalu lintas antar aplikasi secara keseluruhan. Nilai numRequests menjelaskan jumlah permintaan berbeda yang diterima server. Gunakan nilai ini untuk memberikan konteks nilai bytesIn dan bytesOut dalam memastikan utilisasi jaringan MongoDB sesuai dengan harapan dan penggunaan aplikasi.

```
> db.serverStatus().network
{
        "bytesIn" : NumberLong(218789872),
        "bytesOut" : NumberLong(71180255),
        "numRequests" : NumberLong(10847)
}
```

Gambar 6.70 Analisa server status jaringan

Perintah db.serverStatus().opcounters menjelaskan gambaran umum operasi basis data berdasarkan jenis dan memungkinkan untuk menganalisis beban pada database secara lebih terperinci. Angka-angka ini akan berkembang seiring berjalannya waktu dan sebagai tanggapan terhadap penggunaan *database*. Analisis nilai-nilai ini dari waktu ke waktu untuk melacak utilisasi *database*.

```
> db.serverStatus().opcounters
{
        "insert" : 1348,
        "query" : 1268,
        "update" : 0,
        "delete" : 34,
        "getmore" : 10,
        "command" : 8189
}
```

Gambar 6.71 Analisa server status *opcounters*

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi dan pengujian yang telah dilakukan didapatkan kesimpulan sebagai berikut:

1. Sistem *data storage* dapat dibangun dengan menggunakan *Internet Gateway Device* sebagai penerima data dari setiap data sensor yang ada diluar sistem, sekaligus menyimpan data ke sebuah *data storage* yang berupa MongoDB GridFS. MongoDB GridFS dapat mengatasi permasalahan dalam penyimpanan data berukuran besar, laju penambahan data, dan format data yang beragam. Mongo GridFS akan menyimpan data menjadi dua koleksi, yaitu koleksi file chunks dan file metadata. File yang disimpan berbentuk binary JSON sehingga dapat mempercepat proses penyimpanan dan pengaksesan ke dalam *data storage*. Dengan adanya GridFS, data sensor yang beragam dapat disimpan tanpa harus memikirkan format data dan tipe data. Terlebih GridFS merupakan spesifikasi *storage* dari MongoDB untuk menyimpan data yang besarnya lebih dari 16MB.
2. Berdasarkan hasil pengujian fungsional, seluruh fungsi yang diuji berhasil berjalan dengan benar. Dan untuk pengujian non fungsional, pada pengujian skalabilitas API web service untuk GET data dengan mengirimkan 50 sampai 300 request, didapatkan rata-rata request yang berhasil adalah 173,66. Untuk pengujian skalabilitas API web service untuk POST data dengan mengirimkan 100 sampai 1000 post, didapatkan rata-rata *request post* yang berhasil adalah 443,33. Untuk pengujian *response time*, didapatkan pada GridFS lebih cepat karena data yang diambil berupa chunks yang lebih kecil daripada data pada MongoDB.

7.2 Saran

Berdasarkan kesimpulan penelitian, maka penulis merekomendasikan berupa saran-saran untuk penelitian berikutnya sebagai berikut:

1. Sistem *data storage* sebagai solusi penyimpanan data sensor yang heterogen dapat dikembangkan dengan mencari *database* lain, selain MongoDB GridFS dan memungkinkan solusi yang dihasilkan lebih baik.
2. Penelitian berikutnya dapat dilakukan dengan membandingkan lebih dari satu *data storage* untuk dapat menguji performa dari membandingkan *data storage* dengan MongoDB GridFS dan *data storage* yang lain dalam menyimpan data sensor yang heterogen.

DAFTAR PUSTAKA

Adi, H. K., Sakti, E., & Amron, K. (2016). *PENGUMPULAN DATA MENGGUNAKAN METODE PUBLISH SUBSCRIBE PADA NODE SENSOR DALAM WIRELESS MESH NETWORK.* MALANG.

Anwari, H. (2017). PENGEMBANGAN IOT MIDDLEWARE BERBASIS EVENTBASED DENGAN PROTOKOL KOMUNIKASI COAP, MQTT. *J-PTIIK*.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Elsevier*.

Chodorow, K., & Dirolf, M. (2010). MongoDB: The Definitive Guide (1st ed.). *O'Reilly Media*.

Daniel, J. V., Dr.V.Parthasarathy, & Suresh, P. (2014). A state of the art review on the Internet of Things (IoT) History,Technology and Fields of Deployment. *IEEE*.

Idrees, M. (2012). Software Developer's New Ideas & Solutions for Professional. 3-4.

Jiang, L., & Xu, L. D. (2014). An IoT-Oriented Data Storage Framework in Cloud Computing Platform. *IEEE*.

Kende, M., Minton, G., Olshansky, S., Wilton, R., Wood, G., & York, D. (2015). The Internet of Things: An Overview . *The Internet Society (ISOC)*.

Lampkin, V., Leong, W. T., Olivera, L., Rawat, S., Subrahmanyam, N., & Xiang, R. (2012). *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry* (1st ed.). USA: WebSphere MQ.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, United States Department of Commerce*.

Ngu, A. H., & Gutierrez, M. (2016). IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*.

Purbo, O. W. (2011). *Petunjuk Praktis Cloud Computing Menggunakan Open Source*. Yogyakarta: CV Andi Offset.

Seguin, K. (2012). *The Little MongoDB Book.* San Fransisco: Git Hub Inc.

Skvorc, D., Horvat, M., & Srbljic, S. (2014). Performance Evaluation of Websocket Protokol for Implementation of Full-Duplex Web Streams. *IEEE*.

Solace. (n.d.). *MQTT Topics*. Retrieved 12 30, 2016, from http://docs.solace.com/Features/MQTT-Topics.htm

Tingli, L., Yang, L., Ye, T., Shuo, S., & Wei, M. (2012). A Storage Solution for Massive IoT Data Based on NoSQL. *IEEE*.

Tiwari, S. (2011). *Professional NoSQL.* Indianapolis: John Wiley & Sons, Inc.