**Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer** e-ISSN: 2548-964X
Vol. 2, No. 7, Juli 2018, hlm. 2549-2557 http://j-ptiik.ub.ac.id

# Pengembangan Sistem Penyimpanan Data Berbasis MongoDB dan GridFS Untuk Menyimpan Data Yang Beragam Dari Node Sensor

**Gabreil Arganata[1], Eko Sakti Pramukantoro[2], Widhi Yahya[3]**

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]gabreilarganata13@gmail.com, [2]ekosakti@ub.ac.id, [3]widhi.yahya@ub.ac.id

**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. Selain itu 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, *response time* penyimpanan dan pengambilan data, serta analisis performa dari *data storage*. Hasil dari pengujian fungsional didapatkan bahwa sistem penyimpanan data yang dikembangkan sudah berjalan sesuai dengan fungsinya dalam menyimpan beragam data ke dalam *data storage*. 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 heterogeneous data from variety of sensors, and it will keep getting bigger everyday. The examples of heterogeneous 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, response time of saving and retrieving data, also analysis performance of data storage. 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. 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*

## 1. PENDAHULUAN

*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.

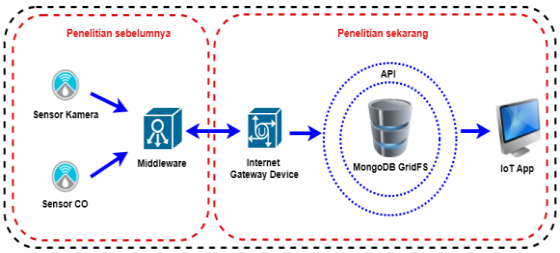
Di antara database NoSQL ini, MongoDB cukup populer karena memiliki banyak dukungan dari berbagai framework web. Semantik data pada MongoDB menggunakan JSON yang mudah dimengerti. MongoDB juga mempunyai model data yang fleksibel. Hal ini sesuai dengan representasi data pada penelitian sebelumnya yang 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*).

## 2. 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* GridFS. Untuk memudahkan proses *get* dan *post* data, dibangun API *web service*. Untuk melihat data yang disimpan dalam *data storage* GridFS peneliti membangun sebuah IoT App menggunakan *websocket*. Pada gambar 1 menjelaskan tentang gambaran umum sistem.




**Gambar 1** Gambaran umum sistem

## 3. PERANCANGAN

### 3.1 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 2 menjelaskan proses dari *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.

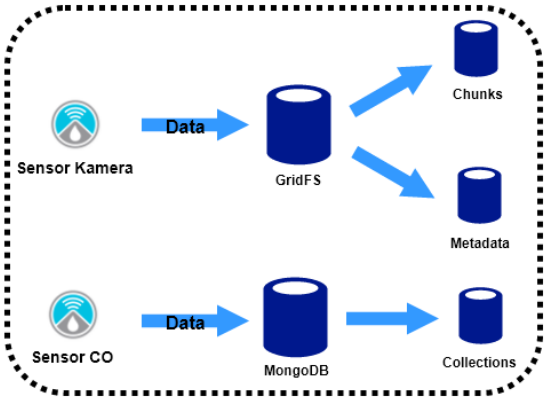
### 3.2 Perancangan API *RESTfull Web Service*

Data dari *Internet Gateway Device* akan dikirimkan dan disimpan dalam *data storage* 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 2 menunjukkan cara kerja API *RESTfull web service*.


**Gambar 2** Cara kerja API RESTfull *web service*

### 3.3 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 3 menunjukkan alur komunikasi sistem.




**Gambar 3** 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.

### 3.4 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.

Pada Gambar 5 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.

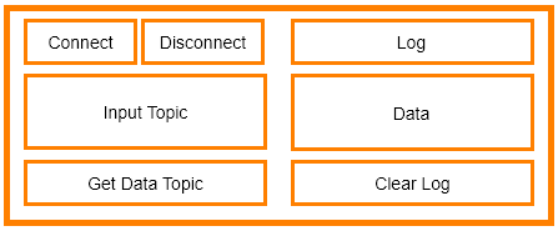
## 4. IMPLEMENTASI

### 4.1 Implementasi Lingkungan Sistem

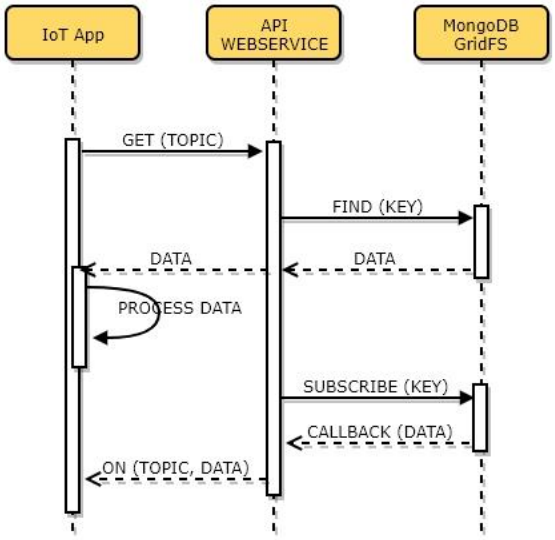
Implementasi topologi jaringan akan membahas konfigurasi yang dibutuhkan oleh *Internet Gateway Device* dan laptop dalam membangun sistem untuk pengujian sistem. Konfigurasi yang akan dilakukan adalah mengatur alamat IP pada *interface* eth0 menjadi *static*. Selain itu diperlukan juga konfigurasi IP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

#### 4.1.1 Konfigurasi Middleware

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.




**Gambar 4** Rancangan IoT App




**Gambar 5** Sequence diagram IoT App

#### 4.1.2 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 6 menunjukkan pengaturan koneksi pada *Internet Gateway Device*.

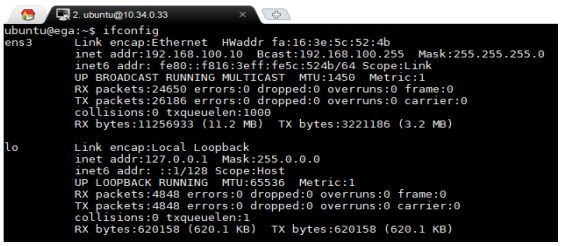
```
mqttc = mqtt.Client("server",
clean_session=False)
mqttc.connect("10.34.15.56", 1883)
```

**Gambar 6** 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.

### 4.1.3 Konfigurasi Pada Data Storage GridFS

*Data storage* disini dikembangkan di sebuah *Virtual Private Server* (VPS) dengan IP eksternal 10.34.0.33. Gambar 7 menunjukkan konfigurasi IP pada *data storage*.




**Gambar 7** Konfigurasi pada VPS

## 4.2 Implementasi Internat 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 8 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 8** Kode program untuk *subscribe*

```
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 9** Kode program subscribe topik pada middleware

Pada gambar 9 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*.

## 4.3 Implementasi API RESTfull Web Service

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

### 4.3.1 Methods GET

Implementasi *web service* GET yang dibuat dapat dilihat pada gambar 10.

```
@app2.route('/api/getdata/<string:name>', methods=['GET',"POST"])
def getdata(name):
    db = MongoClient().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 "sukses"
```

**Gambar 10** 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".

### 4.3.2 Methods POST

Implementasi *web service* POST yang dibuat dapat dilihat pada gambar 11 dan gambar 12.

```
@app2.route('/api/postdata', methods=['POST'])
def postdata():
        data = request.get_json()
        db = MongoClient().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 11** Kode program *web service* POST untuk data gambar

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

**Gambar 12** Kode program *web service* POST untuk data CO

### 4.4 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.

#### 4.4.1 Instalasi MongoDB

Perintah dalam menginstall *data storage* MongoDB ditunjukkan dalam Gambar 13.

```
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 13** Instalasi MongoDB

Setelah *data storage* MongoDB telah terinstall, sekarang saatnya menjalankan *data storage* tersebut dengan perintah:

Gambar 14 menunjukkan perintah menjalankan MongoDB.

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

**Gambar 14** Perintah Menjalankan MongoDB

#### 4.4.2 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:

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

**Gambar 15** Implementasi *data storage* GridFS data gambar

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

**Gambar 16** menunjukkan 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.

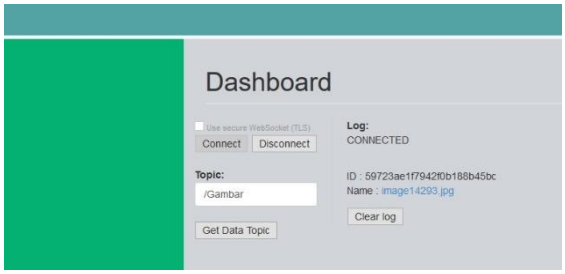
```
client = MongoClient("10.34.0.33", 27017)
```

**Gambar 16** Pengalamatan penyimpanan pada MongoDB GridFS

Pada Gambar 17 menjelaskan penyimpanan data pada *data storage* yang berapa pada VPS di alamat 10.34.0.33.

### 4.5 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 18.




**Gambar 17** Tampilan Web

## 5. PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

### 5.1 Pengujian Fungsional

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. Tabel 1 menunjukkan analisis hasil pengujian fungsional.

**Tabel 1** 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 |

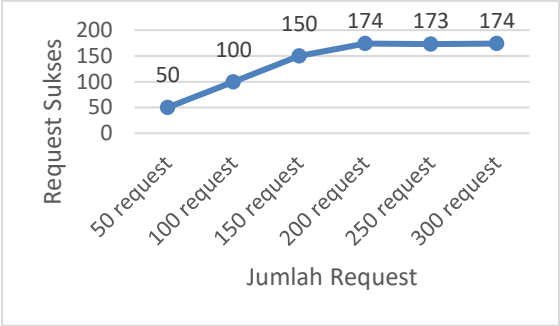
## 5.2 Pengujian Skalabilitas API Web Service

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.

### 5.2.1 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.

Pada gambar 19 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.

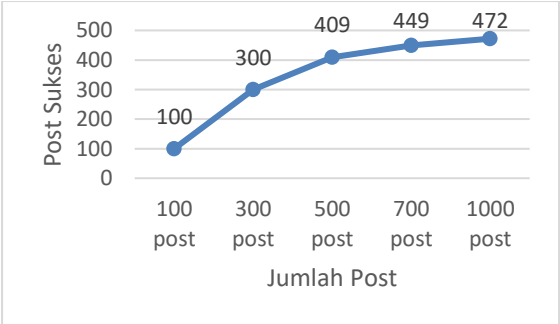



**Gambar 18** Grafik analisis pengujian skalabilitas GET API *web service*

### 5.2.2 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.

Pada gambar 20 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.



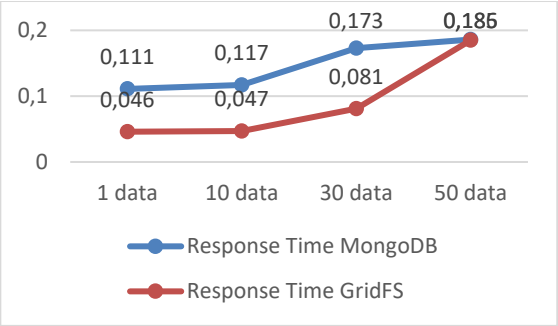

**Gambar 19** Grafik analisis pengujian skalabilitas POST API *web service*

## 5.3 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).

Pada gambar 21 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 20** 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*.

### 5.4 Analisis Performa MongoDB

Untuk mengevaluasi performa dalam MongoDB menggunakan perintah serverStatus untuk memberikan gambaran umum tentang status proses pada *database*. Saat kita mengembangkan dan mengoperasikan aplikasi dengan MongoDB, kita perlu menganalisis kinerja aplikasi dan *database*-nya. Dengan menggunakan perintah serverStatus, kita bisa melihat berbagai kondisi dalam MongoDB. Misalkan 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 21** Analisa server status

## 6. 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.

## 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.

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.

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,fromhttp://docs.solace.com/Features/MQTT-Topics.htm

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