IMPLEMENTASI *CONSTRAINED APPLICATION PROTOCOL* (CoAP) PADA *SEMANTIC IOT WEB SERVICE*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Made Rezananda Putra
NIM: 155150201111079

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

IMPLEMENTASI *CONSTRAINED APPLICATION PROTOCOL* (CoAP) PADA *SEMANTIC IOT WEB SERVICE*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Made Rezananda Putra
NIM: 155150201111079

Skripsi ini telah diuji dan dinyatakan lulus pada
13 Mei 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Eko Sakti P., S.Kom., M.Kom	Fariz Andri Bakhtiar, S.T., M.Kom
NIK: 201102 860805 1 001	NIK: 201709 840314 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 referensi.

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, 13 Mei 2019

Made Rezananda Putra

NIM: 155150201111079

PRAKATA

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, karena berkat rahmat dan karunia-nya, penulis dapat menyelesaikan laporan skripsi yang berjudul "IMPLEMENTASI *CONSTRAINED APPLICATION PROTOCOL* (CoAP) PADA *SEMANTIC IOT WEB SERVICE*" dengan baik. Adapun tujuan dari penulisan laporan skripsi ini adalah untuk mendapatkan gelar Sarjana Komputer pada Program Studi Teknik Informatika, Fakultas Ilmu Komputer Universitas Brawijaya.

Selama proses pengerjaan laporan skripsi ini, penulis mendapatkan pengalaman serta pelajaran baru yang diaplikasikan ke dalam laporan skripsi ini. Penulis menyadari bahwa laporan skripsi ini tidak akan berhasil tanpa adanya dukungan serta doa dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis menyampaikan rasa hormat dan terimakasih kepada:

1. Bapak Eko Sakti Pramukantoro, S.Kom., M.Kom., dan bapak Fariz Andri Bakhtiar, S.T., M.Kom selaku dosen pembimbing skrispi yang memberikan berbagai masukan serta arahan yang sangat baik kepada penulis sehingga dapat menyelesaikan skripsi ini.
2. Kepada kedua orang tua penulis yang senantiasa memberikan doa serta dukungannya selama penulis menempuh pendidikan di Malang dan juga saat proses pengerjaan skripsi ini.
3. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D., selaku Dekan Fakultas Ilmu Komputer, Universitas Brawijaya.
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D., selaku ketua Jurusan Teknik Informatika, Universitas Brawijaya.
5. Bapak Agus Wahyu Widodo, S.T., M.Cs., selaku ketua Program Studi Teknik Informatika, Universitas Brawijaya.
6. Orang tua penulis yang selalu memberikan dukungan moril dan materil selama penyusunan skripsi ini.
7. Teman-teman yang telah membantu memberikan saran dan kritikan dalam menyempurnakan skripsi ini.

Penulis mengakui bahwa dalam penulisan dokumen skripsi ini masih terdapat banyak kekurangan, sehingga kritik yang membangun serta saran sangat penulis butuhkan. Akhir kata penulis berharap agar skripsi ini dapat bermanfaat bagi seluruh pihak yang menggunakannya.

Malang, 13 Mei 2019

Penulis

Rezananda6897@student.ub.ac.id

ABSTRAK

Made Rezananda Putra, Implementasi *Constrained Application Protocol* (CoAP) Pada *Semantic IoT Web Service*

Pembimbing: Eko Sakti P., S.Kom., M.Kom dan Fariz Andri Bakhtiar, S.T., M.Kom

Keberagaman protokol komunikasi dalam *internet of things* mengakibatkan permasalahan interoperabilitas, khususnya *syntactical interoperability*. Pada penelitian sebelumnya telah dikembangkan *semantic IoT web service* yang mampu menerima data dari *node sensor* melalui HTTP. Namun penggunaan HTTP saja tidak cukup untuk menerima data dari *node sensor* dengan protokol komunikasi yang beragam. Permasalahan tersebut juga ditambah dengan adanya *node sensor* yang memiliki keterbatasan atau *resource-constrained*. Salah satu solusi dari permasalahan tersebut adalah penambahan protokol komunikasi pada *semantic IoT web service* agar dapat terhubung dengan *node sensor* yang *resource-constrained. Constrained Application Protocol* (CoAP) merupakan protokol komunikasi yang mampu menangani permasalahan *resource-constrained.* CoAP dirancang untuk menggunakan sumber daya yang minim sehingga cocok diimplementasikan pada lingkungan IoT yang memiliki keterbatasan. Oleh sebab itu, dalam penelitian ini akan ditambahkan CoAP sebagai protokol komunikasi pada *semantic IoT web service*. Hasil pengujian dari segi fungsional menunjukkan *semantic IoT web service* mampu menerima data berbentuk JSON, gambar dan video melalui CoAP. Penambahan CoAP pada *semantic IoT web service* mampu menangani permasalahan *resource-constrained* dan interoperabilitas, khususnya *syntactical interoperability* dengan menyediakan protokol komunikasi melalui CoAP dan HTTP kepada *node sensor*.

Kata kunci: *Syntactical Interoperability, Resource Constrained, Semantic IoT Web Service*, *Constrained Application Protocol* (CoAP)

ABSTRACT

Made Rezananda Putra, ***The Implementation of Constrained Application Protocol (CoAP) In Semantic IoT Web Service***

Supervisors: Eko Sakti P., S.Kom., M.Kom dan Fariz Andri Bakhtiar, S.T., M.Kom

The Heterogeneous of communication protocols in the Internet of things results in interoperability problems, especially syntactical interoperability. The previous work has developed a semantic IoT web service capable of receiving data from the sensor nodes over HTTP. But the use of HTTP alone is not enough to receive data from sensor nodes with various communication protocols. The problem is also coupled with the presence of sensor nodes that have limitations or resource-constrained. One solution of the problem is the addition of communication protocols in the semantic IoT web service in order to connect with the resource-constrained sensor nodes. Constrained Application Protocol (CoAP) is a communication protocol capable of handling resource-constrained problems. CoAP is designed to use a minimal resource so that it is suitable to be implemented in a limited IoT environment. Therefore, in this study will be added CoAP as a communication protocol on the semantic IoT web service. A functional faceted test results in a semantic IoT web service capable of receiving JSON data, images and video via CoAP. Added CoAP on Semantic IoT Web service capable of handling resource-constrained issues and interoperability, in particular syntactical interoperability by providing communication protocols via CoAP and HTTP to sensor nodes.

Keywords: Syntactical Interoperability, Resource Constrained, Semantic IoT Web Service, *Constrained Application Protocol* (CoAP)

DAFTAR ISI

DAFTAR TABEL

DAFTAR PSEUDOCODE

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Internet of Things (IoT) didefinisikan sebagai suatu jaringan yang dapat menghubungkan berbagai perangkat dalam dunia fisik menggunakan protokol yang tersedia (Guoqiang, et al., 2013). Secara umum IoT terdiri dari tiga komponen yaitu *things*, *middleware* dan *cloud* (Pramukantoro, et al., 2019). Masing-masing komponen dalam *internet of things* memiliki keberagaman, sebagai contoh penggunaan *processor* (8-/16-bit), sistem operasi, aplikasi dalam berbagai bahasa pemrograman, serta protokol komunikasi yang digunakan seperti CoAP, MQTT, HTTP, XMPP, dsb (Chander, et al., 2017). Keberagaman protokol komunikasi dalam *internet of things* mengakibatkan permasalahan interoperabilitas. (Rezaei, et al., 2014) mendefinisikan interoperabilitas sebagai kemampuan dua (atau lebih) sistem untuk saling bertukar data dan memproses data tersebut. (Desai, et al., 2015) membagi interoperabilitas menjadi tiga yaitu *Technical/Network Interoperability*, *Syntactical Interoperability* dan *Semantic Interoperability*. *Syntactical Interoperability* mengacu pada protokol perukaran pesan yang digunakan. Permasalahan interoperabilitas, khususnya *syntactical interoperability* dapat mengakibatkan keterbatasan interaksi antar komponen dalam *internet of things* karena protokol komunikasi yang beragam.

Cloud merupakan komponen dalam *internet of things* yang memuat web *service*. Salah satu fungsi dari web *service* adalah sebagai antar muka untuk menerima data dari *node sensor* melalui protokol komunikasi HTTP. Pada penelitian sebelumnya, telah dikembangkan *semantic IoT web service* pada *cloud* yang mampu menentukan data *storage* (MongoDB atau GridFS) berdasarkan topik dan tipe dari data yang dikirim oleh *node sensor*. Metode penentuan data *storage* tersebut dinamakan *semantic filtering* (Pramukantoro, et al., 2019). *Node sensor* melakukan pengiriman data kepada *semantic IoT web service* melalui protokol komunikasi HTTP. Namun penggunaan HTTP saja tidak cukup untuk menangani pengiriman data dari *node sensor* dengan protokol komunikasi yang beragam. Permasalahan tersebut juga ditambah dengan adanya *node sensor* yang memiki keterbatasan atau *resource-constrained*. *Resource-constrained* dalam *internet of things* dapat berupa perangkat dengan ukuran kecil, komputasi dan memori yang rendah serta penggunaan jaringan yang terbatas (Madhu, et al., 2017). Salah satu solusi dari permasalahan tersebut adalah penambahan protokol komunikasi pada *semantic IoT web service* agar dapat terhubung dengan *node sensor* yang *resource-constrained.*

Berbagai penelitian telah dilakukan dengan merumuskan solusi dari permasalahan diatas. (Dalipi, et al., 2016) dan (Ruta, et al., 2017) dalam penelitiannya telah mengusulkan sebuah *framework* untuk perangkat IoT yang *resource-constrained* melalui penerapan *Constrained Application Protocol* (CoAP). Penggunaan CoAP didasarkan pada protokol layer *transport* yang digunakan yaitu *User Datagram Protocol* (UDP) dengan *header* lebih kecil dibandingkan *header*

Transmission Control Protocol (TCP). CoAP juga dirancang untuk menggunakan sumber daya yang minim, sehingga cocok diimplementasikan pada lingkungan IoT yang memiliki keterbatasan (Wiryawan, et al., 2018). (Tanganelli, et al., 2015) dalam penelitiannya melakukan implementasi *Constrained Application Protocol* (CoAP) dalam bentuk *library* dengan bahasa pemrograman python bernama CoAPthon. *Library* tersebut bertujuan untuk mempermudah pengembangan aplikasi dalam *internet of things* menggunakan protokol komunikasi CoAP.

Berdasarkan pembahasan serta solusi diatas, dalam penelitian ini akan ditambahkan *Constrained Application Protocol* (CoAP) pada *semantic IoT web service* (Pramukantoro, et al., 2019). Penambahan protokol komunikasi CoAP akan diimplementasikan dalam bentuk *Application Programming Interface* (API) menggunakan *library* CoAPthon (Tanganelli, et al., 2015). API tersebut terbagi menjadi dua bagian, yaitu API di sisi *client* dan API di sisi *server*. API CoAP di sisi *client* bertugas untuk mengirim data, sedangkan API CoAP di sisi *server* akan berada pada *cloud* yang bertugas untuk menerima data dan meneruskannya kepada *semantic IoT web service*. Harapan dari penelitian ini adalah *semantic IoT web service* yang mampu mengatasi permasalahan *resource-constrained* dan interoperabilitas, khususnya *syntactical interoperability* dengan menyediakan protokol komunikasi melalui CoAP dan HTTP kepada *node sensor*.

1.2 Rumusan masalah

Berdasarkan pada permasalahan yang diangkat, maka rumusan masalah difokuskan pada:

1. Bagaimana menambahkan *Constrained Application Protocol* (CoAP) sebagai protokol komunikasi pada *semantic IoT web service*?
2. Bagaimana kinerja *semantic IoT web service* yang telah ditambah *Constrained Application Protocol* (CoAP) dari sisi interoperabilitas?
3. Bagaimana kinerja *semantic IoT web service* yang telah ditambah *Constrained Application Protocol* (CoAP) dari sisi penggunaan memori, penggunaan CPU, *throughput* dan *runtime*?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Menambah *Constrained Application Protocol* (CoAP) sebagai protokol komunikasi pada *semantic IoT web service*.
2. Mengetahui kinerja *semantic IoT web service* yang telah ditambah *Constrained Application Protocol* (CoAP) dari sisi interoperabilitas.
3. Mengetahui kinerja *semantic IoT web service* yang telah ditambah *Constrained Application Protocol* (CoAP) dari sisi penggunaan memori, penggunaan CPU, *throughput* dan *runtime*.

1.4 Manfaat

Manfaat dari penelitian ini diharapkan dapat menjadi bahan referensi dalam membangun *IoT web service* oleh berbagai pihak khususnya pengembangan IoT yang menggunakan protokol komunikasi CoAP. Selain itu penelitian ini dapat digunakan sebagai dasar dari penelitian-penelitian selanjutnya karena masih banyak aspek dari penelitian ini yang dapat dikembangkan.

1.5 Batasan masalah

Agar implementasi yang dikerjakan dapat mencapai hasil yang sesuai dan maksimal, maka penelitian ini dibatasi dalam hal:

1. Penelitian ini berfokus pada penambahan protokol CoAP pada *semantic IoT web service*.
2. Implementasi dari CoAP dengan menggunakan CoAPthon.
3. Penelitian ini berfokus pada pengolahan data yang didapatkan dari *node* sensor untuk disimpan ke dalam data *storage* melalui *semantic IoT web service.*
4. Media penyimpanan data pada *semantic IoT web service* menggunakan MongoDB dan GridFS.
5. Bentuk data yang digunakan adalah data terstruktur yang berupa JSON, serta data tidak terstruktur yang berupa *file* gambar dan *file* video.
6. Penelitian ini tidak melakukan enkripsi terhadap data

1.6 Sistematika pembahasan

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

BAB I PENDAHULUAN

Pada Bab I Pendahuluan akan menjelaskan mengenai latar belakang, rumusan masalah, tujuan, manfaat penelitian, batasan masalah dan sistematika pembahasan dari laporan penelitian ini.

BAB II LANDASAN KEPUSTAKAAN

Pada Bab II Landasan Kepustakaan akan menjelaskan mengenai landasan teori dan referensi penelitian yang ada serta relevan dengan proses penyusunan skripsi ini.

BAB III METODOLOGI

Pada Bab III Metodologi akan menjelaskan mengenai langkah kerja yang dilakukan dalam penelitian, teknik pengumpulan data dan teknik analisis hipotesa.

BAB IV PERANCANGAN

Pada Bab IV Perancangan akan menjelaskan mengenai rekayasa kebutuhan serta perancangan dalam mengintegrasikan CoAP dengan *semantic IoT web service*.

BAB V IMPLEMENTASI

Pada Bab V pada bagian implementasi akan dijelaskan proses membangun sistem yang telah dirancang dalam bab sebelumnya.

BAB VI PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

Pada Bab VI Pengujian dan Analisis akan dilakukan proses pengujian terhadap sistem dan aplikasi untuk mengetaui performansi dan kinerja dari sistem yang diimplementasikan.

BAB VII PENUTUP

Pada Bab VII Berisi Kesimpulan dan Saran yang akan menjelaskan kesimpulan dari rumusan masalah yang berdasarkan analisis dan pengujian yang dilakukan. Selain itu akan ditambahkan saran-saran dan pengembangan terhadap penelitian ini.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi tentang kajian pustaka dan dasar teori yang berkaitan dengan penelitian yang sedang dilakukan. Kajian pustaka memberikan informasi terkait hubungan antara penelitian sebelumnya dengan topik penelitian yang sedang dikerjakan. Dasar teori berisi informasi mengenai teori yang berhubungan dengan topik penelitian mulai dari yang umum sampai yang paling khusus sebagai penunjang dalam penelitian.

2.1 Kajian Pustaka

Kajian pustaka akan membahas mengenai penelitian terdahulu yang terkait dengan penelitian yang saat ini dilakukan. Pada kajian pustaka akan dibandingkan penelitian terhadulu dengan penelitian yang saat ini dari sisi persamaan penelitian, perbedaan serta rencana penelitian yang akan dilakukan. Tabel 2.1 akan menjelaskan mengenai kajia pustaka dalam penelitian yang saat ini dilakukan.

Tabel 2.1 Tabel kajian pustaka

No	Nama Penulis, Tahun, Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	(Pramukantoro, et al., 2019) *A Semantic RESTful API for Heterogeneous IoT Data Storage*	Penggunaan *semantic IoT web service* dalam menerima dan menyimpan data pada data *storage*	Pengembangan *semantic IoT web service* dengan protokol komunikasi menggunakan HTTP	Penambahan *Constrained Application Protocol* (CoAP) pada *semantic IoT web service*
2	(Rezaei, et al., 2014) *Interoperability evaluation models: A systematic review*	Menggunakan metode pengujian *Interoperability assessment methodology*	Mendeskripsikan mengenai berbagai metode pengujian interoperabilitas	Menggunakan parameter *Protocol*, *Intepretation* dan *Information Untilization* dari metode *Interoperability assessment methodology*

Tabel 2.1 Tabel kajian pustaka (lanjutan)

3	(Tanganelli, et al., 2015) CoAPthon: Easy Development of CoAP-based IoT Applications with Python	Menggunakan *library* CoAPthon dalam implementasi API CoAP	Mengembangkan *library* untuk protokol CoAP dalam bahasa pemrograman python	Memanfaatkan *library* CoAPThon dalam implementasi API CoAP.
4	(Kurniawan, et al., 2018) Perbandingan Kinerja Cassandra dan MongoDB Sebagai Backend IoT Data Storage	Pengguaan parameter pengujian CPU *usage*, *memory usage*, *runtime* dan *throughput*.	Membandingkan kinerja database cassandra dan MongoDB	Menguji kinerja dari *semantic IoT web service* yang telah ditambah CoAP

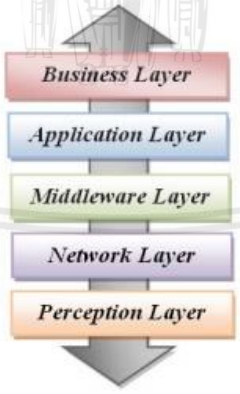
2.2 Dasar Teori

Dasar teori membahas mengenai berbagai teori yang mendukung penelitian yang saat ini sedang dilakukan.

2.2.1 *Internet of Things* (IoT)

Internet of Things (IoT) merupakan suatu keadaan dimana benda memiliki identitas, memiliki intelenjensi dan dapat berkomunikasi dengan sosial, lingkungan maupun penggunanya (Atzori, Lera, & Morabito, 2010). *Internet of things* merupakan perkembangan keilmuan dalam mengoptimalkan kehidupan berdasarkan sensor-sensor serta peralatan pintar lainnya yang terkoneksi dengan jaringan internet (Keoh, Kumar, & Tschofenig, 2014). Ide dasar dari dikembangkanya IoT adalah cara dalam menghubungkan berbagai objek seperti sensor, *Radio-frequency identification* (RFID) dan telepon seluler agar dapat berinteraksi satu sama lain. IoT akan mebuat objek dapat mendengar, berpikir dan melakukan pekerjaan dengan cara melakukan komunikasi antara objek satu dengan objek yang lainnya sehingga dapat berbagi informasi (Shah & Yaqoob, 2016). Menurut *Amazon Web Service*, *Internet of Things* (IoT) dapat mengkoneksikan dunia fisik dengan perantara internet, sehingga data yang diperoleh dari perangkat yang terhubung dapat digunakan untuk meningkatkan produktivitas dan efisiensi. Dari perngertian diatas dapat disimpulkan bahwa *internet of things* merupakan sebuah konsep dalam mengkoneksikan perangkat-perangkat berupa sensor atau peralatan pintar yang memiliki identitas dan dapat saling berkomunikasi dengan perantara internet. Menurut Kraijak & Tuwanut (2015) Arsitektur dari IoT terdiri dari 5 layer yaitu:

1. *Perception Layer*: layer ini memiliki kemiriban dengan layer fisik dari mode OSI yang terdiri dari beberapa sensor. Lapisan layer ini memiliki fungsi untuk melakukan manajemen perangkat seperti identifikasi dan pengumpulan informasi oleh setiap perangkat sensor. Informasi dapat berupa suhu, kelembaban, Ph dan lainnya. Informasi tersebut selanjutnya akan dikirim menggunakan layer network ke media penyimpanan data terpusat.
2. *Network Layer*: layer ini memiliki peran penting dalam melakukan pengiriman informasi secara aman dari perangkat sensor, ke pusat data dengan melalui jaringan 3G, 4G, WiFi, inframerah dan lain sebagainya. Selain itu layer ini bertanggung jawab juga untuk melakukan pengiriman informasi ke layer diatasnya.
3. *Middleware Layer*: Adapun fungsi utama dari middleware yaitu melakukan pengaturan service dan menyimpan informasi ke dalam basis data. Selain itu layer ini memiliki kemampuan untuk mengambil, memproses, menghitung informasi yang selanjutnya akan mengeluarkan keputusan secara otomatis berdasarkan hasil perhitungan.
4. *Application Layer*: layer aplikasi bertanggung jawab dalam manajemen aplikasi berdasarkan informasi yang telah diproses dalam layer middleware. Adapun contoh dari layer ini berupa smart health, smart car dan smart home.
5. *Busines Layer*: layer ini berfungsi untuk mencakup keseluruhan aplikasi dan manajemen layanan pada IoT. Data yang diperoleh dari sensor dapat ditampilkan dalam bentuk grafik, model bisnis maupun flowchaert sehingga dapat dianalisis lalu mengambil keputusan tentang strategi bisnis yang akan dilakukan.



Gambar 2.1 Layer Pada Internet of Things

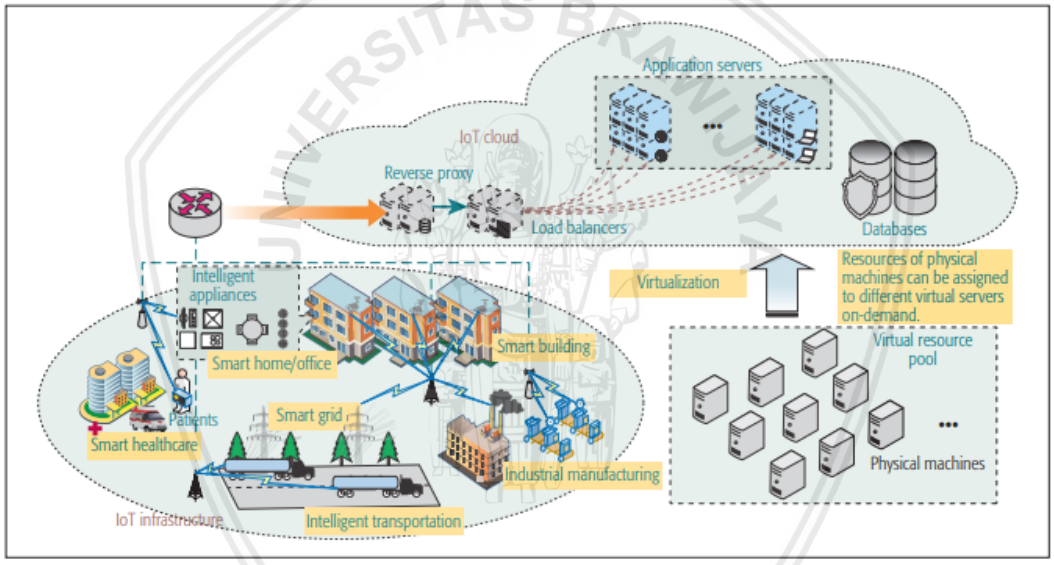
(Kraijak & Tuwanut, 2015)

Tujuan dari adanya *Internet of Things* adalah untuk mempermudah aktifitas dan pekerjaan manusia dengan cara berkomunikasi dengan benda-benda yang saling terkoneksi menggunakan internet. Penerapan *Internet of Things* telah diterapkan di berbagai bidang keilmuan dan industri, seperti bidang ilmu kesehatan, informatika, geografis dan beberapa bidang ilmu lainnya. Cara kerja dari Internet of Things menurut (Kurniawan, et al., 2018) yaitu setiap benda yang

akan menjadi “things” harus memiliki alamat IP atau IP address. IP address merupakan alamat penanda atau identitas pada suatu jaringan yang menyebabkan benda tersebut unik. Setelah itu benda tersebut dapat dipanggil menggunakan alamat IP tersebut menggunakan jaringan internet.

2.2.1.2 *Internet of Things Cloud*

Perkembangan teknologi komunikasi secara nirkabel menghasilkan objek yang dapat terkoneksi dengan internet secara interaktif sehingga menunculkan konsep *internet of things*. Tahun 2020 diperkirakan jumlah dari perangkat *internet of things* yang digunakan akan mencapai 50 miliar unit. Perangkat tersebut akan menghasilkan banyak data dengan berbagai format. Hal tersebut berbanding terbalik dengan perangkat IoT yang memiliki keterbatasan sumber daya karena memiliki ukuran fisik yang kecil dan ukuran penyimpanan yang sedikit. Sehingga IoT *cloud* sangat diperlukan sebagai syarat dalam mendukung banyak perangkat IoT dalam pemrosesan data.



Gambar 2.2 Arsitektur *internet of things*

(Hou, et al., 2016)

Seperti yang dilustrasikan dalam gambar 2.1, IoT *cloud* terdiri dari beberapa komponen yang terdiri dari beberapa *server* dalam melakukan tugas yang berbeda-beda. *Server* tersebut berupa Virtual Machine (VM) yang memanfaatkan teknologi virtualisasi dan independen satu sama lain. VM tersebut dapat memuat server *load balancer/reverse proxy*, *database* dan aplikasi *server* yang dapat dikonfigurasi. Adapun penjelasan dari komponen yang ada dalam IoT cloud adalah sebagai berikut:

1. *Virtual Resource Pool*

   Sumber daya perangkat keras dari mesin fisik (seperti CPU, *memory* dan konektivitas jaringan) tidak sepenuhnya dapat dimanfaatkan, sehingga hal tersebut dapat menjadi pemborosan sumber daya dan masalah skalibilitas

server. Dalam mengatasi permasalahan tersebut dilakukan teknik virtualisasi yang digunakan untuk pemanfaatan sumber daya yang lebih baik dari IoT *cloud*. Melalui teknik virtualisasi, perangkat lunak yang hypervisor dapat berjalan dalam mesin fisik sebagai lapisan abstrak dalam mengatur seluruh sumber daya dan juga menyediakan lingkungan sistem operasi yang independen untuk klien. Dengan memanfaatkan teknik virtualisasi, *Virtual Resource Pool* dapat menghubungkan beberapa mesin fisik yang memuat seluruh sumber daya perangkat keras. Dengan metode tersebut, *server* dapat memperoleh sumber daya yang tepat sesuai kebutuhan.

2. *Application Server*

   Application server sering dianggap sebagai komponen yang penting dalam IoT *cloud* karena bertanggung jawab dalam menawarkan layanan bisnis kepada pelanggan. *Application server* harus menyediakan fasilitas dan lingkungan yang maksimal dalam menjalankan banyak aplikasi bedasarkan protokol yang digunakan dalam aplikasi tersebut. *Application server* dalam *cloud* yang tradisional biasanya berdasarkan pada protokol HTTP. Namun protokol HTTP tidak cocok digunakan dalam IoT *cloud*, karena dibatasi oleh sumber daya, komputasi maupun komunikasi. Sehingga protokol aplikasi yang lebih cocok untuk diterapkan dalam IoT *cloud* salah satunya adalah MQTT. MQTT dirancang untuk menjadi protokol IoT dengan sumber daya yang terbatas. Protokol tersebut menggunakan mekanisme *publish-subscribe* berbasis topik.

3. *Database*

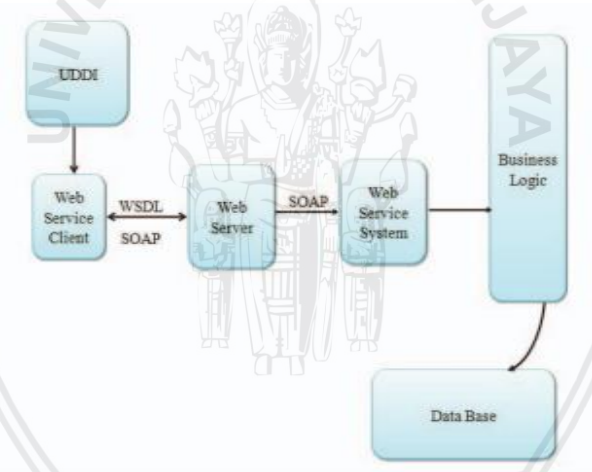
   Penyimpanan data atau *data storage* terbagi menjadi *relational database* (*Structured Query Language*) dan *non relational database* (NoSQL). Kedua jenis *data storage* tersebut adalah opsional yang digunakan dalam IoT *cloud*, sehingga tergantung dengan kebutuhan dari aplikasi. SQL di desain untuk melakukan penyimpanan data ke dalam tabel dua dimensi. Namun kinerja dari SQL akan menyebabkan *bottleneck* untuk aplikasi IoT yang *real-time*, sehingga NoSQL *database* biasanya digunakan untuk menyediakan layanan secara *real-time* untuk *data storage*. NoSQL *database* memungkinkan data untuk secara langsung dapat disimpan ke dalam memori sehingga menyebabkan kecepatan *input-output* (I/O) menjadi meningkat.

4. *Reverse Proxy and Load Balancing*

   Dengan semakin meningkatnya jumlah pengguna dan aplikasi, sehingga *application server* dibutuhkan untuk menangani jutaan permintaan secara bersamaan. Permintaan tersebut tanpa adanya penjadwalan jika *load balancing* tidak digunakan. Hasilnya beberapa *server* akan mengalami kepadatan karena beban yang berlebihan. Oleh karena itu *load balancing* sangat penting diterapkan dalam IoT *cloud* dalam mendistribusikan beban kerja secara merata dalam setiap *server*, serta dapat memanfaatkan sumber daya yang tersedia secara maksimal

2.2.2 Web *Service*

W3C mendeskripsikan Web *service* sebagai perangkat lunak untuk mendukung interaksi yang beroperasi dalam machine to machine di atas jaringan. Web *service* memiliki tampilan antar muka yang dideskripsikan pada format *machine-processable* (WSLD). Menurut kreger (2001) web *service* merupakan sebuah antar muka yang menggambarkan sekumpulan operasi yang dapat diakses melalui jaringan, misalnya jaringan internet, dalam bentuk pesan berupa *Extensible Markup Language* (XML). Sedangkan menurut Manes (2001), web *service* merupakan sebagian informasi atau proses yang dapat diakses oleh siapa saja, kapan saja dan dengan menggunakan piranti apa saja serta tidak terikat dengan sistem operasi atau bahasa pemrograman yang digunakan. Web service terdiri dari kumpulan fungsi dan *method* yang berpusat pada sebuah server yang dipanggil oleh pengguna, walaupun bahasa pemrograman atau platform yang digunakan berbeda, pengguna tetap dapat mengakses method-method tersebut (Martasari & Aminudin, 2010). Web *service* dapat dibangun menggunakan bahasa pemrograman apapun dan platform apapun, karena web *service* memiliki standar format data yang universal untuk berkomunikasi yaitu XML serta JSON dan menggunakan protokol SOAP.



Gambar 2.3 Arsitektur web service

(THIYAGARAJAN & RAVEENDRA, 2017)

Web *service* menyediakan standar komunikasi secara *machine to machine* (M2M) melalui jaringan internet. *Representation State Transfer* (REST) merupakan bagian dari web *sercive* dimana sumber dayanya dapat dimanipulasi menggunakan operasi *stateless* yang beragam. Arsitektur berbasis REST memiliki metode yang umum digunakan yaitu GET, PUT, POST dan DELETE. Selain REST, terdapat bagian web *service* berupa web *service* dinamis yang menggunakan operasi acak seperti SOAP *message*.

Dalam penerapannya pada *Internet of things*, web *service* harus mampu mendukung protokol yang digunakan dalam berkomunikasi. Penggunaan web *service* berbasis REST tidak cukup menjadi acuan dalam proses memberikan layanan kepada *client*. *Client* terkadang memiliki keterbatasan akan sumber daya

baik dari sisi jaringan maupun penyimpanannya yang mengakibatkan sulitnya terhubung dengan web *service* berbasis REST, karena protokol transport yang digunakan yaitu TCP yang berbasiskan connection oriented. Sehingga hadir berbagai protokol yang dapat digunakan dalam web *service*, sebagai contoh MQTT dan CoAP. Kedua protkol tersebut dapat diterapkan menjadi sebuah web *service* berdasarkan pada bahasa pemrograman yang digunakan.

2.2.2.1 *Semantic IoT Web Service*

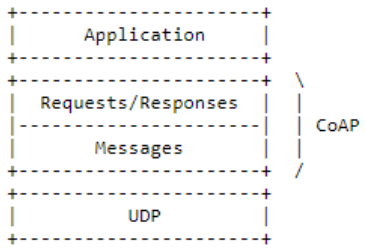
Semantic IoT web service merupakan sebuah web service yang mampu menentukan data storage (MongoDB atau GridFS) melalui identifikasi struktur, format, tipe dan ukuran dari data. Metode yang digunakan dalam menentukan data storage tersebut adalah *semantic filtering*. Data dikirim oleh *client* kepada semantic IoT web service melalui protokol HTTP. Selain itu, semantic IoT web service juga memiliki metode otentikasi dan otorisasi menggunakan JSON Web Token (JWT). Pertama-tama client harus melakukan proses otentikasi terlebih dahulu menggunakan token key dan secret key yang didapat dari pendaftaran device pada IoT *application semantic IoT web service*. Token key dan secret key tersebut kemudian dikirim kepada semantic IoT web service untuk divalidasi. Jika proses validasi berhasil, semantic IoT web service akan mengirimkan JWT token kepada client. JWT Token tersebut kemudian dikirim kembali oleh client bersamaan dengan data dan topik. JWT token, data dan topik yang telah diterima oleh semantic IoT web service selanjutnya akan diproses. JWT token akan digunakan untuk proses otorisasi selama *semantic filtering* diterapkan dalam menyimpan data pada data storage. Data yang telah tersimpan dalam data storage dapat ditampilkan pada IoT application berdasarkan topik (Pramukantoro, et al., 2019).

2.2.3 *Constrained Application Protocol* (CoAP)

Constrained Application Protocol (CoAP) merupakan protokol layer aplikasi yang termasuk ke dalam standar RFC 7252. *Internet Engineering Task Force* (IETF) yang merupakan pengembang dari protokol CoAP memberikan definis dari *Constrained Application Protocol* (CoAP) berupa protokol web transfer yang dikhususkan untuk menangani node yang terbatas dan jaringan yang terbatas. CoAP menyediakan interaksi *request/response* antara endpoint aplikasi, mendukung penemuan layanan dan sumber daya, termasuk konsep utama dari Web seperti URL dan tipe media internet. CoAP dirancang untuk mempermudah interaksi dengan HTTP untuk dapat dintegrasikan dengan Web dengan mendukung beberapa kebutuhan seperti *multicast*, *overhead* yang rendah dan sederhana untuk lingkungan yang terbatas.

Constrained Application Protocol (CoAP) beroperasi pada UDP dengan tujuan untuk menghindari *congestion control* yang kompleks. CoAP mendukung arsitektur *Representational State Transfer* (REST) sehingga menyediakan URL dan metode seperti GET, POST, PUT dan DELETE. CoAP memiliki kelemahan dari sisi sumber daya sehingga CoAP perlu mengoptimalkan panjang datagram untuk

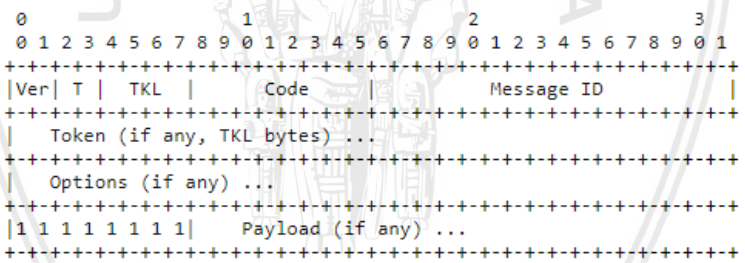
menyediakan transmisi yang handal. Selain itu CoAP juga memiliki fitur untuk melakukan mekanisme retransmisi.



Gambar 2.4 Layer pada CoAP

CoAP memiliki empat tipe pesan yang digunakan untuk melakukan pertukaran data antara client dengan server, yaitu:

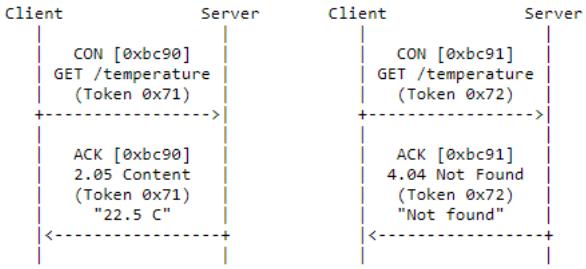
1. *Confirmable* (CON) yang merupakan pesan berisi request dan memerlukan Acknowledgment
2. *Non-Confirmable* (NON) yang merupakan pesan berulang sehingga tidak memerlukan Acknowledgment
3. *Acknowledgment* (ACK) yang merupakan pesan yang berisi response
4. *Reset* (RST) yang merupakan pesan jika pesan CON tidak diterima dengan benar atau terdapat loss connection



Gambar 2.5 Format pesan pada CoAP

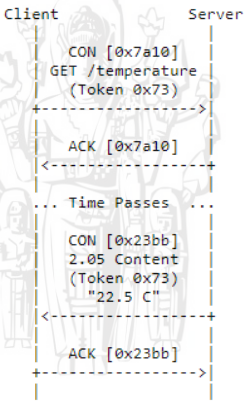
Pesan dalam CoAP akan di *encode* dalam format biner. Adapun bagian dari format pesan dari CoAP yaitu *header* yang memiliki ukuran 4 byte, *Token* yang berfungsi untuk memastikan pesan telah dikirim dan diterima dengan ukuran 0-8 byte. *Option* berfungsi untuk menyimpan informasi *request/response* seperti URI dan tipe media. Sedangkan bagian pesan data yang akan dikirimkan. Setiap pesan dalam CoAP memiliki Message ID dengan ukuruan 16 but yang digunakan untuk mendeteksi duplikasi apabila menggunakan opsi reliability. Apabila pesan yang dikirimkan mendapatkan tanda *Confirmable* (CON) maka akan mendapatkan reliability. Sehingga pengirim akan menunggu *Acknowledgement* (ACK) dari penerima. Pesan ACK akan terus dikirimkan apabila dalam waktu tertentu pengirim tidak mengirimkan pesan ACK padanya. Sehingga jika terjadi kesalahan pada penerima maka ia akan mengirimkan pesan *Reset* (RST). Sedangkan pesan yang tidak membutuhkan reliability dapat dikirimkan sebagai pesan *non-*

Confirmable (NON), sehingga penerima tidak harus mengirimkan pesan ACK kepada pengirim namun masih memungkinkan untuk mengirim pesan RST.



Gambar 2.6 *Piggybacked response*

Adapun beberapa kondisi dalam mekanisme request/response dalam CoAP yaitu *piggybacked response* yang merupakan kondisi dimana saat terjadi request pesan baik CON maupun NON, maka jika memungkinkan pesan akan dikirimkan bersamaan dengan ACK. Namun *piggybacked response* tidak sampai ke tujuan maka client akan mengirimkan pesan request kemabli. Gambar 2.6 menunjukkan contoh dari piggybacked response, yang hanya 1 berhasil sedangkan jika gagal maka akan mengembalikan response berupa 404.



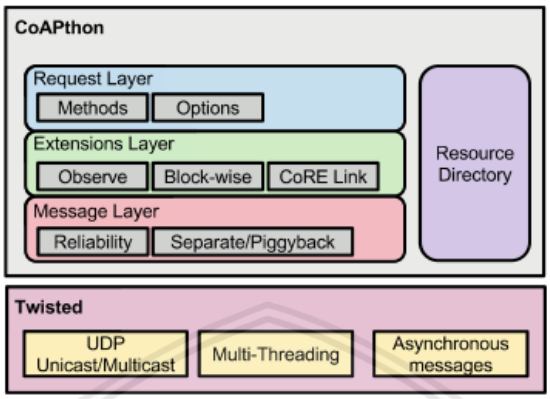
Gambar 2.7 *Separate response*

Model lainnya yaitu *Separate response* yang dimana server tidak dapat merespon CON request secara langsung, maka server akan mengirimkan pesan ACK sehingga client akan berhenti mengirimkan request. Apabila server telah siap, maka server akan mengirimkan pesan CON kepada client dan client akan membalas dengan mengirimkan pesan ACK kepada server. namun jika request yang digunakan adalah NON, maka response yang digunakan juga akan NON.

2.2.4 CoAPthon

Constrained Application Protocol (CoAP) merupakan *framework* aplikasi yang dapat melakukan interaksi dengan *constrained device* dengan berbasis paradigma RESTful. CoAP memiliki potensi menjadi masa depan dari web of things. CoAP mengadopsi mekanisme client/server dari HTTP dan menggunakan interaksi secara request/response. selain itu CoAP juga mendukung penggunaan URL dan menggunakan method stadar seperti GET, POST, PUT dan DELETE. CoAP

merupakan hasil penyederhanaan dari protokol HTTP yang berjalan dalam UDP, memiliki arsitektur jaringan yang simple, serta mendukung implementasi dalam constrained device. Protokol ini juga memiliki fitur yaitu resource observing, block-wise transfer dan CoRE link format dalam mendukung.



Gambar 2.8 Arsitektur CoAPThon (Tanganelli, et al., 2015)

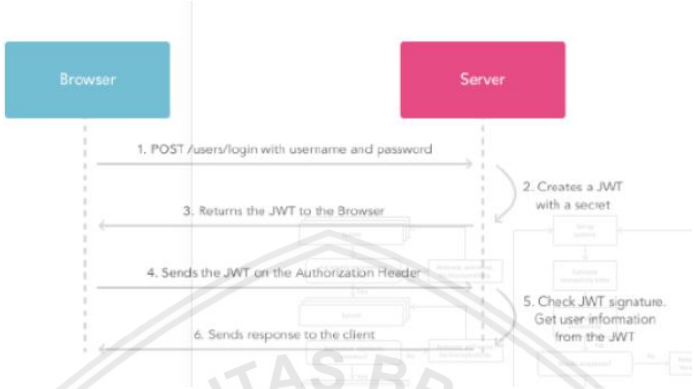
CoAP dapat diimplementasikan salah satunya sebagai sebuah library dengan menggunakan bahasa pemrograman python yang dinamakan CoAPthon. CoAPthon mendukung fitur yang disediakan oleh CoAP yaitu resource observing, block-wise transfer dan CoRE link format dalam mendukung. Library ini didesain untuk menyediakan pengembang aplikasi secara mudah dan simple. Tujuan dari dibuatnya library ini adalah untuk menyediakan pengembangan aplikasi Internet of Things dengan cepat. CoAPthon menggunakan python dalam pengembangannya, dimana python merupakan bahasa pemrograman yang mudah untuk dipahami. Python saat ini juga telah berkembang menjadi bahasa pemrograman yang digunakan untuk pengembangan aplikasi secara mudah, serta telah diadopsi dalam pengembangan dan implementasi pada Internet of Things. Adapun arsitektur dari CoAP yang direfleksikan pada CoAPthon yaitu Message Layer, Request Layer dan Extension Layer.

CoAPthon dapat diimplementasikan dalam framework Twisted. Twisted framework jaringan secara event-driven untuk Python yang digunakan untuk mengimplementasikan beberapa protokol aplikasi seperti HTTP, FTP serta untuk mengembangkan protokol baru yang berbasiskan UDP dan TCP. Selain itu twisted juga memiliki kelebihan dalam mengimplementasikan fungsi dari jaringan tingkat rendah seperti multi-threading, socket UDP, pertukaran pesan secara Asyncronous, serta berguna pada CoAPthon dalam mendukung Multicast (Tanganelli, et al., 2015).

2.2.5 *JSON Web Token* (JWT)

JSON Web Token merupakan salah satu mekanisme otentikasi dan otorisasi yang biasanya diterapkan dalam sebuah web *service*. *JSON Web Token* berupa sebuah token yang berbentuk *string* dan digunakan untuk menjamin integeritas pesan yang dikirim salah satu pihak. Token yang telah dikirimkan dapat diverifikasi menggunakan *secret key* yang telah ditentukan. Struktur dari JWT terdiri dari tiga

bagian yaitu *header*, *payload* dan *signature*. Header biasanya terdiri dari dua bagian yaitu JWT serta algortima *hashing* seperti HMAC SHA-256. Pada *payload* berisi klaim yang merupakan suatu identitas yang biasanya identitas dari pengguna serta *metadata* tambahan. Klaim memiliki tiga jenis yaitu *reserved, public*, dan *private claims*. Terakhir pada bagian ketiga dari JWT terdapat *signature* yang berisi *hash* dari komponen *header*, *payload* dan kunci (Hamid, et al., 2018).



Gambar 2.9 Alur *JSON web token* (Hamid, et al., 2018)

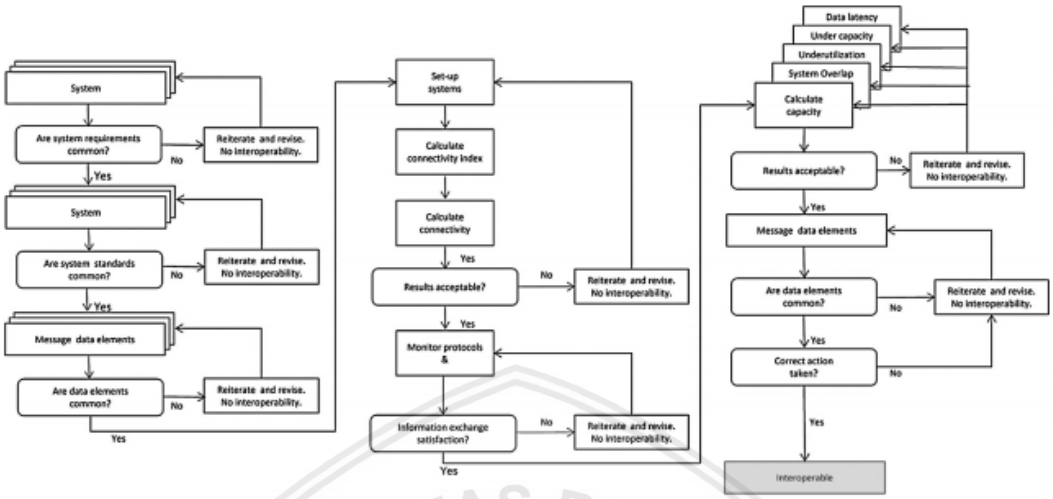
Gambar 2.9 menunjukkan contoh dari alur otentikasi dari *JSON web token* antara *browser* dengan *server*. Ketika pengguna selesai melakukan proses login, pengguna akan mendapatkan Token JWT yang dapat disimpan untuk proses otorisasi.

2.2.6 Pengujian Interoperabilitas

(Rezaei, et al., 2014) memberikan definisi terkait interoperabilitas yaitu kemampuan dua atau lebih sistem untuk saling bertukar data dan dapat saling memproses data tersebut. Interoperabilitas menurut (Desai, et al., 2015) secara umum dapat dibagi menjadi tiga yaitu *Technical Interoperability*, *Syntactical Interoperability* dan *Semantic Interoperability*. *Techincal interoperaility* mengacu pada protokol jaringan yang digunakan pada perangkat IoT untuk terhubung ke perangkat lain seperti ZigBee, Zwave dan Bluetooth. *Syntactical interoperability* mengacu pada model data yang dikirimkan atau protokol pertukaran pesan yang digunakan seperti MQTT, CoAP atau HTTP dan *Semantic interoperability* mengacu pada makna dari sebuah konten atau data yang dikirimkan.

Kebutuhan sistem yang mampu mendukung interoperabilitas merupakan salah satu dampak perkembangan IoT. Kebutuhan tersebut menghasilkan berbagai peneltiian terkait metode pengujian interoperabilitas dalam *internet of things* guna mengukur tingkat interoperabilitas dari sistem yang telah dibuat. Adapun penelitian terkait metode pengujian interoperabilitas telah dilakukan oleh (Rezaei, et al., 2014) dengan membandingkan berbagai metode pengujian interoperabilitas yang telah ada dengan tujuan untuk melihat seberapa luas cakupan interoperabilitas yang diuji. Dari hasil analisis peneliti metode *Interoperability assessment methodology* yang dapat digunakan dalam menguji sistem yang saat ini diimplementasikan. *Interoperability assessment methodology*

menggabungkan pengujian secara deskriptif dan pengujian secara kualitatif. Proses dari *Interoperability assessment methodology* dapat dilihat pada gambar 2.9.



Gambar 2.10 Proses *Interoperability assessment methodology* (Rezaei, et al., 2014)

Gambar 2.9 menunjukkan proses dari *Interoperability assessment methodology* yang terdiri dari sembilan komponen utama pengujian. Adapun sembilan kebutuhan tersebut yakni *Requirements*, *Standards*, *Data elements, Node connectivity, Protocols, Information flow, Data latency, Interpretation, dan Information utilization*.

1. *Requirements*: sistem atau komponen dari sisi interoperabilitas perlu dipertimbangkan dan harus memiliki kebutuhan yang sama.
2. *Standars*: mendefinisikan standar dari pesan yang dikirimkan serta media yang digunakan untuk mengirim pesan tersebut.
3. *Data elements*: setelah standars dan requeirements telah didefinisikan dan diperiksa dan penilaian dari standars dan requirements tersebut telah bernilai positif, maka dapat diklaim bahwa node telah terhubung melalui format yang umum.
4. *Node Connectivity*: node connectivity merupakan sebuah variabel yang bergantung dengan waktu yaitu interval waktu yang belainan dan berulang, sehingga dapat dikatakan sebuah elemen yang sulit untuk diukur. Secara sederhana node connectivity merupakan kemampuan untuk mengirim dan menerima data setiap saat. Perhitungan node connectivity adalah sebagai berikut:

$$C_i = \frac{k}{n \times (n-1)}$$

C = Tingkat konektivitas node (pada saat pengujian)

k = Jumlah koneksi (jalur yang ada)

n = Jumlah node yang ada dalam sistem

5. *Protocols*: Protokol memberikan akses kepada data. *protocols* terbagi menjadi dua bagian yaitu dari sisi pengirim yang akan mengirim data dan sisi penerima yang akan menerima data.
6. *Information Flow*: volume dari data biasanya sebuah fungsi dari operations' tempo dan the area of interest (AOI) yang ditentukan melalui perintah operasional. Dalam information flow terdapat *capasity* yang mengacu pada laju perpindahan data dan overload yang mengacu pada banyaknya data yang harus dipertukarkan dalam proses transimis pada sistem
7. *Data Latency*: *data latency* merupakan waktu yang diperlukan dalam menerima data dari *user*.
8. *Interpretation*: setelah konsistensi dari data yang dikirimkan telah pasti, maka interpretasi data dari masing-masing proses harus diperiksa.
9. *Information untilization*: setelah data telah ditafsirkan dengan benar, selanjutnya adalah melakukan verifikasi terhadap data tersebut.

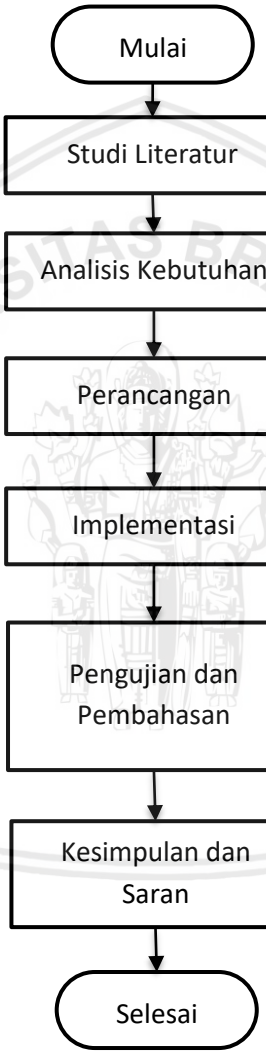
2.2.7 Pengujian Kinerja

Dalam menguji kinerja dari IoT cloud terhadap masing-masing dari API web service diberikan parameter pengujian. Parameter pengujian yang diberikan sesuai dengan pengujian yang telah dilakukan oleh (Kurniawan, et al., 2018) yang tersebut terdiri dari memory usage, CPU usage, runtime dan throughput. Adapun penjelasan dari masing-masing parameter pengujian tersebut adalah sebagai berikut:

1. CPU *usage* merupakan parameter dalam mengambil jumlah persentase dari penggunaan CPU yang digunakan dalam IoT *cloud* terhadap masing-masing API web *service* saat kedua API tersebut melayani *request* dari *client*.
2. Memory *usage* merupakan parameter dalam mengambil konsumsi memori (*byte*) dari IoT *cloud* terhadap penggunaan API web *service* dari masing-masing protokol saat kedua API tersebut melayani *request* dari *client*.
3. *Runtime* merupakan parameter dalam mengambil waktu eksekusi dari masing-masing API web service dalam menyelesaikan proses POST data berdasarkan skenario pengujian yang diberikan.
4. *Throughput* merupakan parameter dalam mengambil kecepatan yang diperlukan masing-masing API dalam menyelesaikan proses POST data berdasarkan skenario yang diberikan. *Throughput* dapat dihitung dengan membagi jumlah operasi dengan *runtime*

BAB 3 METODOLOGI PENELITIAN

Pada bab ini penulis akan menjelaskan langkah-langkah atau metodologi dari penelitian yang saat ini dilakukan. Tahap awal dari metodologi yaitu penulis mengumpulkan teori pendukung penelitian yang disusun dalam studi literatur. Tahapan selanjutnya adalah analisis kebutuhan, dilanjutkan dengan proses perancangan, implementasi, kemudian pengujian dan analisis hasil pengujian. Terakhir adalah dilakukannya penarikan kesimpulan dan saran. Langkah-langkah dalam penelitian ditunjukkan pada gambar 3.1.



Gambar 3.1 Metodologi Penelitian

3.1 Studi Literatur

Pada penelitian yang sedang dikerjakan akan dilakukan studi literatur dengan mencari kajian pustaka dan dasar-dasar teori yang digunakan dalam menunjang penulisan skripsi. Adapun kajian pustaka yang dilakukan bersumber dari penelitian sebelumnya yang berkaitan dengan penelitian yang sedang dikerjakan. Sedangkan pada dasar teori akan dilakukan pencarian informasi

mengenai *Internet of Things* (IoT), *semantic IoT web service*, *Constrained Application Protocol* (CoAP), CoAPthon, interoperabilitas, pengujian interoperabilitas dan pengujian kinerja. Berbagai teori pendukung serta kajian pustaka diperoleh dari buku, jurnal, e-book dan dokumentasi projek pada penelitian sebelumnya.

3.2 Analisis Kebutuhan

Analisis kebutuhan adalah tahapan untuk menganalisa kebutuhan dari sistem. Tujuan dilakukannya analisis kebutuhan adalah untuk melakukan batasan terkait kebutuhan sehingga tidak melakukan analisis kebutuhan yang tidak diperlukan dalam implementasi sistem dan juga sebagai acuan dalam merancang sistem, sehingga menjadi sistematis. Adapaun kebutuhan yang diperlukan pada penelitian ini adalah sebuah lingkungan uji berupa *semantic IoT web service* (Pramukantoro, et al., 2019). Kebutuhan berikutnya yang diperlukan adalah sebuah program untuk implemenasi protokol CoAP yaitu CoAPthon (Tanganelli, et al., 2015).

3.3 Perancangan

Perancangan merupakan tahap persiapan sebelum melakukan implementasi sistem. Adapun perancangan yang dilakukan pada penelitian ini yaitu penjelasan mengenai deskripsi umum sistem, lingkungan penelitian, spesifikasi kebutuhan, kebutuhan data, perancangan topologi jaringan, perancangan penambahan *Constrained Application Protocol* (CoAP) pada *semantic IoT web service*, dan perancangan pengujian. Deskripsi umum sistem akan membandingkan penelitian terdahulu dengan penelitian yang saat ini dilakukan. Lingkungan penelitian akan menjelaskan perangkat keras maupun perangkat lunak yang digunakan dalam implementasi maupun pengujian dari sistem. Spesifikasi kebutuhan akan menjelaskan kebutuhan fungsional dan non fungsional dari sistem. Kebutuhan data akan menjelaskan struktur serta format dari data yang akan digunakan untuk diolah pada sistem. Perancangan topologi jaringan akan menjelaskan komponen yang terhubung melalui jaringan pada sistem. Perancangan penambahan CoAP pada *semantic IoT web service* akan menjelaskan proses membangun API di sisi *client* dan *server* dengan CoAPthon. Terakhir adalah perancangan pengujian yang akan menjelaskan mengenai metode pengujian yang akan dilakukan dengan menambahkan skenario pengujian.

3.4 Implementasi

Proses implementasi adalah sesuai dengan perancangan yang telah dilakukan sebelumnya. Implementasi yang pertama berupa implementasi topologi jaringan, kemudian implementasi dari penambahan CoAP pada *semantic IoT web service* dalam bentuk *Application Programming Interface* (API) menggunakan CoAPthon. Implementasi dari API tersebut akan terbagi menjadi API CoAP di sisi *client* untuk mengirim data serta di sisi *server* untuk menerima data dan meneruskannya kepada *semantic IoT web service*.

3.5 Pengujian dan Analisis Hasil Pengujian

Tahap pengujian adalah tahapan untuk mengetahui kinerja sistem yang telah diimplementasikan. Adapun pengujian yang dilakukan berupa pengujian fungsional, pengujian interoperabilitas dan kinerja. Pengujian fungsional akan menguji kebutuhan dari sistem. Pengujian interoperabilitas akan menguji proses pengiriman data melalui CoAP dan HTTP secara independen yaitu melalui masing-masing protokol dan secara bersama-sama dari *node sensor* kepada *semantic IoT web service*. Sedangkan pengujian kinerja akan menguji kinerja dari sistem berdasarkan parameter *runtime*, *throughput*, CPU *usage* dan *memory usage*.

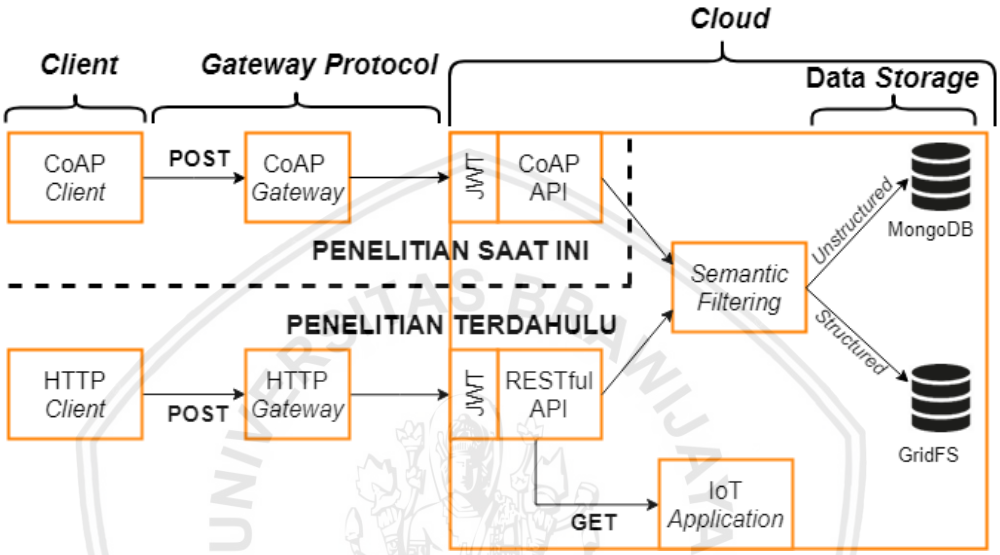
3.6 Kesimpulan dan Saran

Pada bagian ini akan dilakukan pengambilan kesimpulan setelah dilakukannya tahapan perancangan, implementasi, pengujian dan analisis dari sistem. Kesimpulan berasal dari hasil analisis data yang diperoleh dari pengujian. Penarikan kesimpulan dapat diambil dari kelebihan dan kekurangan pada sistem yang telah diimplementasikan. Setelah penarikan kesimpulan lalu diberikan saran dalam rangka memberikan informasi kepada peneliti lain yang ingin mengembangkan penelitian saat ini.

BAB 4 PERANCANGAN

4.1 Deskripsi Umum Sistem

Penelitian ini berfokus pada penambahan *Constrained Application Protocol* (CoAP) sebagai protokol komunikasi pada *semantic IoT web service*. Gambar 4.1 merupakan deskripsi umum sistem terkait penelitian terdahulu dan penelitian yang saat ini akan dilakukan.



Gambar 4.1 Deskripsi umum sistem

Pada penelitian terdahulu, telah dikembangkan *semantic IoT web service* yang mampu menentukan data *storage* (MongoDB atau GridFS) melalui identifikasi struktur maupun tipe dari data yang dikirim oleh *node sensor*. Metode penentuan data *storage* tersebut dinamakan *semantic filtering*. *Node sensor* dapat mengirimkan data kepada *semantic IoT web service* melalui protokol komunikasi HTTP. Selain itu, *semantic IoT web service* juga mampu melakukan mekanisme otentikasi dan otorisasi melalui *JSON Web Token* (JWT). Data yang telah tersimpan dalam data *storage* dapat ditampilkan pada IoT *application* (Pramukantoro, et al., 2019).

Dalam penelitian saat ini, akan ditambahkan *Constrained Application Protocol* (CoAP) pada *semantic IoT web service* yang telah dikembangkan sebelumnya (Pramukantoro, et al., 2019). Implementasi dari CoAP dalam bentuk *Application Programming Interface* (API) dengan menggunakan *library* CoAPthon (Tanganelli, et al., 2015). API CoAP yang akan diimplementasikan terbagi menjadi dua bagian yaitu API CoAP di sisi *client* dan API CoAP di sisi *server*. API CoAP di sisi *client* bertugas dalam mengirim data, sedangkan API CoAP di sisi *server* berada pada *cloud* yang bertugas dalam menerima data dan meneruskannya kepada *semantic IoT web service*. *Semantic IoT web service* nantinya dapat menerima data dari *client* atau *node sensor* melalui protokol komunikasi HTTP serta CoAP.

4.2 Linkungan Penelitian

Lingkungan penelitian digunakan untuk mendukung proses implementasi serta pengujian dari sistem. Lingkungan penelitian terbagi menjadi lingkungan perangkat keras dan perangkat lunak.

4.2.1 Perangkat Keras

Perangkat keras yang dibutuhkan dalam penambahan CoAP pada *semantic IoT web service* dapat dilihat pada tabel 4.1.

Tabel 4.1 Perangkat Keras

No	Perangkat Keras	Deskripsi	Spesifikasi
1	*Virtual Private Server* (VPS)	VPS akan digunakan untuk menjalankan API CoAP server, *semantic IoT web service* dan data *storage*.	OS: Ubuntu server 16.04

4.2.2 Perangkat Lunak

Perangkat lunak yang dibutuhkan dalam penambahan CoAP pada *semantic IoT web service* dapat dilihat pada tabel 4.2.

Tabel 4.2 Perangkat Lunak

No	Perangkat Lunak	Deskripsi
1	CoAPthon (Tanganelli, et al., 2015)	CoAPthon merupakan *library* dalam bahasa pemrograman python yang digunakan untuk implementasi protokol CoAP.
2	Putty	Putty merupakan applikasi *remote console* / terminal yang digunakan untuk meremote VPS dengan port ssh.
3	Tcpdump	Tcpdump digunakan untuk proses *capture packet* pengiriman data dari *client* kepada *cloud*. Hasil dari tcpdump adalah file dengan format .pcap. File tersebut kemdian akan dianalisis dalam aplikasi wireshark.
4	Wireshark	Wireshark digunakan dalam proses analisis hasil *capture packet* dari tcpdump.
5	Studio 3T	Studio 3T digunakan untuk menampilkan data yang telah tersimpan pada data *storage*.

4.3 Analisis Kebutuhan

Analisis kebutuhan mendeskripsikan kebutuhan yang akan digunakan dalam proses implementasi sistem. Kebutuhan sistem dapat dibagi menjadi dua yaitu kebutuhan fungsional dan kebutuhan non-fungsional.

4.3.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan yang harus dipenuhi oleh sistem serta merupakan proses-proses yang dapat dilakukan oleh sistem. Tabel 4.3 mendeskripsikan kebutuhan fungsional pada dari protokol CoAP.

Tabel 4.3 Kebutuhan Fungsional Sistem

No	Kebutuhan Fungsional
1	API *semantic IoT web service* dapat menerima data berbentuk JSON melalui protokol komunikasi CoAP
2	API *semantic IoT web service* dapat menerima data berbentuk gambar melalui protokol komunikasi CoAP
3	API *semantic IoT web service* dapat menerima data berbentuk video melalui protokol komunikasi CoAP
4	API *semantic IoT web service* yang telah ditambah protokol komunikasi CoAP dapat melakukan proses *semantic filtering.*
5	API *semantic IoT web service* yang telah ditambah protokol komunikasi CoAP dapat melakukan otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT).

4.3.2 Kebutuhan Non-fungsional

Kebutuhan non-fungsional merupakan kebutuhan pendukung yang berfungsi dalam membatasi fungsi pada suatu sistem. Tabel 4.4 mendeskripsikan kebutuhan non-fungsional pada sistem.

Tabel 4.4 Kebutuhan Non-Fungsional

No	Kebutuhan Non-fungsional	Deskripsi
1	Kinerja	*Semantic IoT web service* yang telah ditambah CoAP dapat bekerja berdasarkan parameter kinerja penggunaan memori, penggunaan CPU, *runtime* dan *throughput*.

4.4 Kebutuhan Data

Dalam penelitian ini akan digunakan data terstruktur dan data tidak terstruktur seperti pada penelitian sebelumnya (Pramukantoro, et al., 2019). Data

terstruktur yang digunakan adalah data dengan bentuk JSON, sedangkan data yang tidak terstruktur merupakan data yang berbentuk gambar dan video. Gambar 4.2, 4.3 dan 4.4 merupakan struktur dari data berbentuk JSON, gambar dan video yang digunakan.

```
payload = {
      jwtToken = string,
      topic = string,
      data = {
            humidity = integer,
            nitrogen = integer,
            ph = integer
            }
      }
```

Gambar 4.2 Data JSON

```
payload = {
      jwtToken = string,
      topic = string,
      files = pickle (gambar)
      }
```

Gambar 4.3 Data gambar

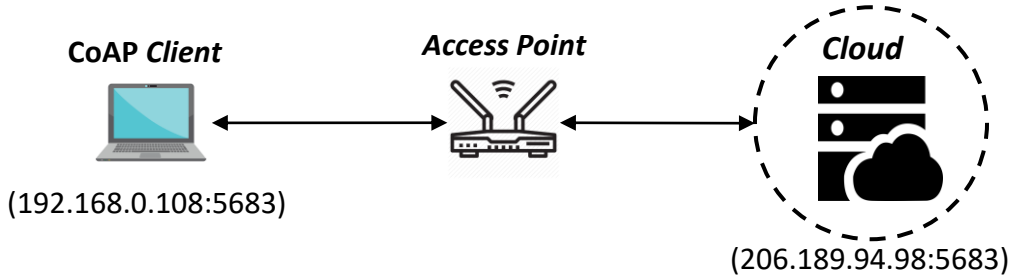
```
payload = {
      jwtToken = string,
      topic = string,
      data = pickle (video)
      }
```

Gambar 4.4 Data Video

Data terstruktur yaitu JSON merupakan data yang didapat dari *node sensor* dalam bentuk angka. Data tidak terstruktur yaitu gambar dan video juga didapat dari *node sensor* dalam bentuk *raw* yang kemudian dirubah ke dalam bentuk *pickle*. Selain data, terdapat komponen JWT Token dan topik. JWT token digunakan dalam proses otorisiasi pada *semantic IoT web service*. Sedangkan komponen topik digunakan sebagai penanda dari data dalam proses penyimpanan pada data *storage*.

4.5 Perancangan Topologi Jaringan

Penambahan protokol CoAP sebagai protokol komunikasi pada *semantic IoT web service* memerlukan topologi jaringan agar dapat memetakan komponen yang terhubung melalui jaringan pada sistem. Adapun perancangan topologi jaringan pada penelitian saat ini dapat dilihat pada gambar 4.5.

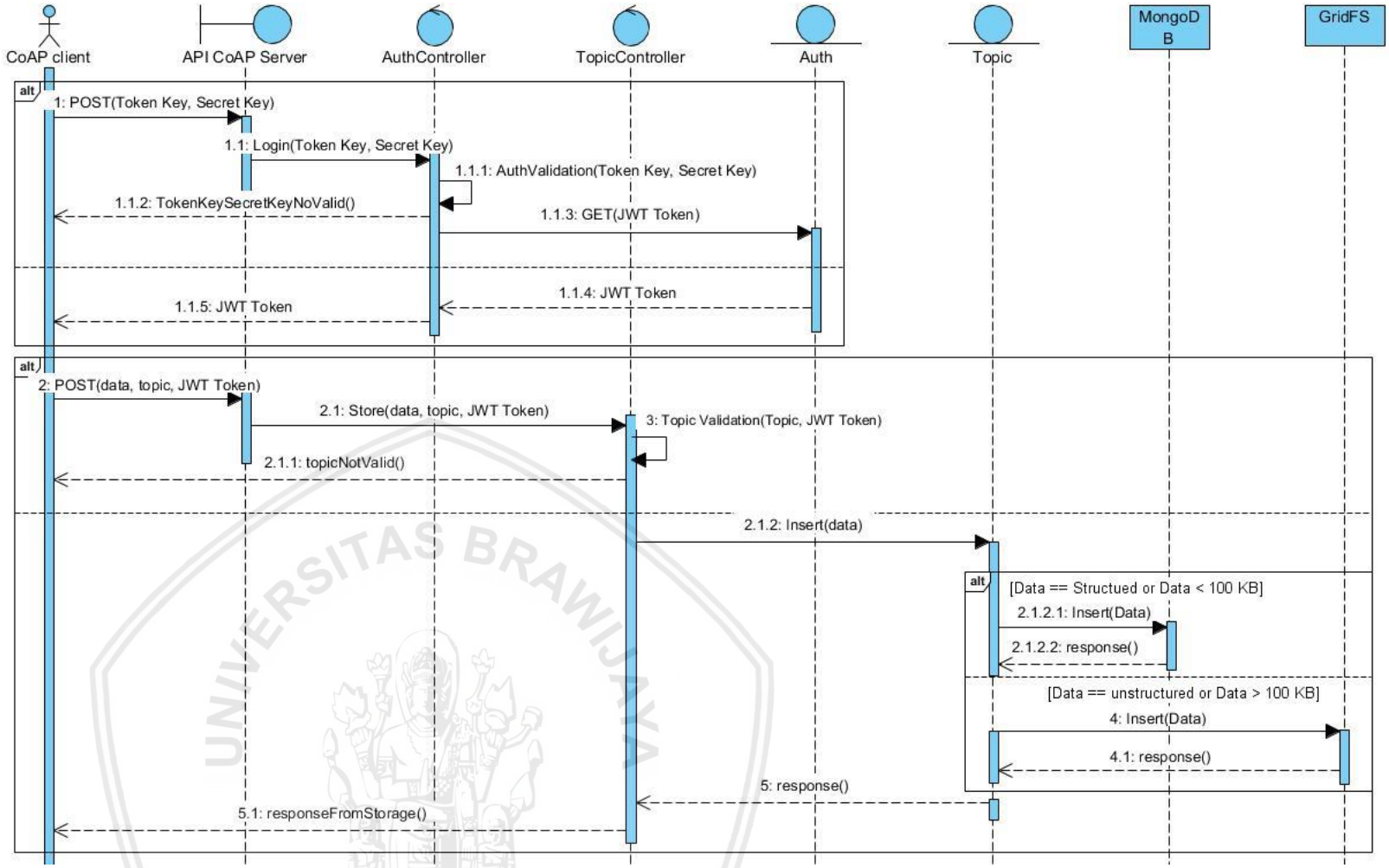


Gambar 4.5 Perancangan topologi jaringan

Dalam gambar 4.5 tersebut terdapat komponen berupa CoAP *client*, *access point* dan *cloud.* Jaringan yang digunakan untuk menghubungkan CoAP *client* dengan *cloud* adalah melalui jaringan *wireless*. CoAP *client* dapat melakukan pengiriman data maupun dapat menampilkan data yang telah tersimpan dalam data *storage*. *Access point* digunakan untuk menghubungkan CoAP *client* dengan internet untuk mengakses *cloud*. *Cloud* dalam penelitian ini merupakan sebuah VPS yang dimana terdapat *semantic IoT web service* dan data *storage*. CoAP *client* akan menggunakan *private* IP sedangkan *cloud* menggunakan *public* IP. Agar CoAP *client* dengan *cloud* dapat saling terhubung melalui protokol komunikasi CoAP, maka diperlukan *port* yang dimana menggunakan *port* 5683.

4.6 Perancangan Penambahan CoAP dengan *Semantic IoT Web Service*

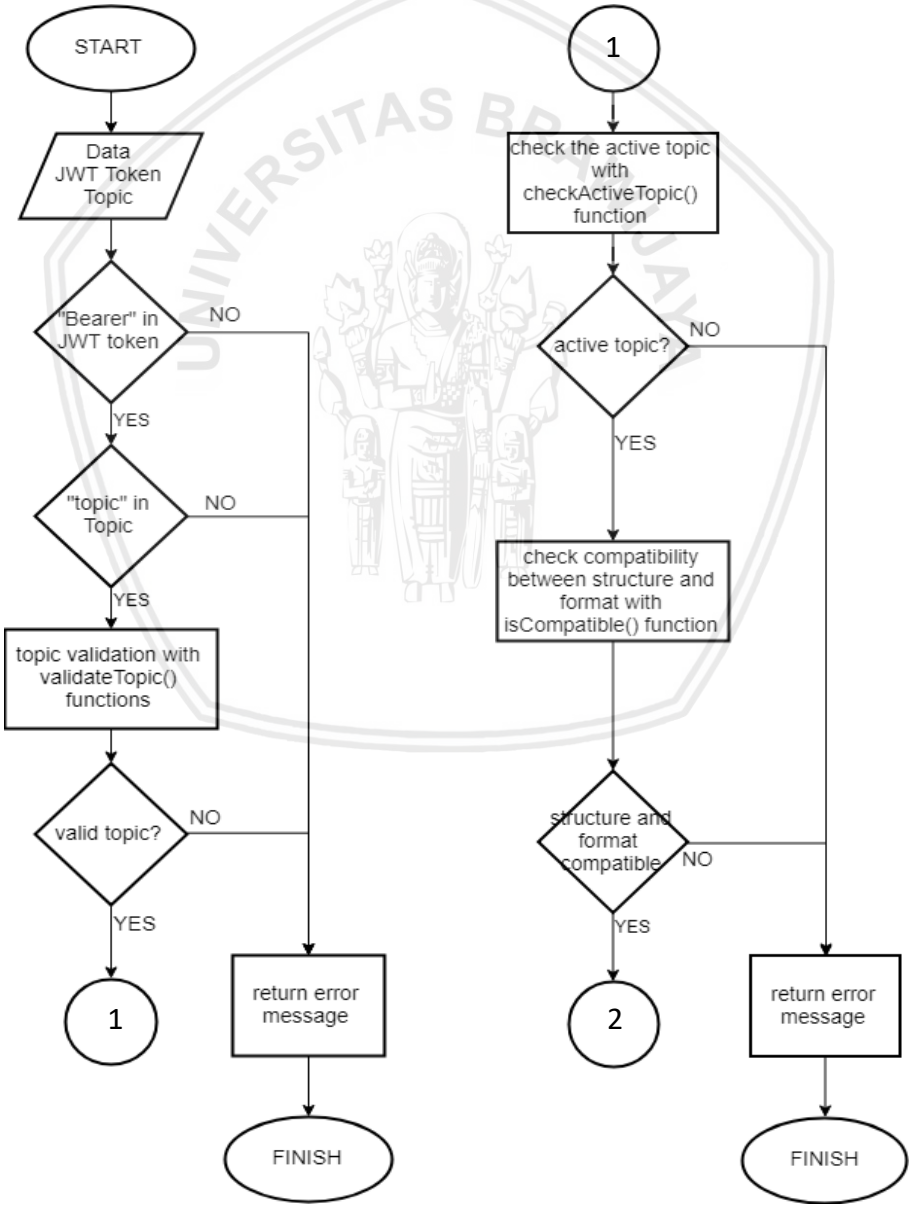
Perancangan penambahan CoAP pada *semantic IoT web service* terbagi menjadi perancangan API CoAP *client* serta API CoAP *server*. Gambar 4.6 menunjukkan *sequence diagram* dari penambahan CoAP pada *semantic IoT web service*. Adapun penjelasan dari *sequence diagram* tersebut yaitu pertama CoAP *client* harus melakukan otentikasi dengan mengirimkan *token key* dan *secret key* kepada *semantic IoT web service* melalui CoAP *server*. *Token key* dan *secret key* didapat dari proses pendaftaran *device* pada IoT *applications* (Pramukantoro, et al., 2019). *Token key* dan *secret key* kemudian akan divalidasi pada *semantic IoT web service* untuk mendapatkan JWT token. Jika valid maka JWT token akan dikirim oleh *semantic IoT web service* kepada CoAP *client* melalui CoAP *server*, namun jika tidak valid maka CoAP *client* akan menerima pesan *error*. JWT token yang telah didapatkan kemudian dikirim kembali bersama dengan data dan topik kepada CoAP *server* lalu diteruskan kepada *semantic IoT web service*. Topik yang telah diterima kemudian divalidasi. Proses validasi tersebut harus menyertakan JWT token sebagai syarat otorisasi. Jika valid, maka dilanjutkan dengan proses penyimpanan data melalui metode *semantic filtering*. Terdapat dua jenis data *storage* yaitu MongoDB untuk menyimpan data terstruktur dan kurang dari 100 KB serta GridFS untuk menyimpan data tidak terstuktur dan lebih dari 100 KB. Setelah proses penyimpanan, *semantic IoT web service* akan mengirimkan respon kepada CoAP *client* baik respon dengan status berhasil maupun dengan status gagal.



Gambar 4.6 *Sequence diagram semantic IoT web service* yang ditambah CoAP

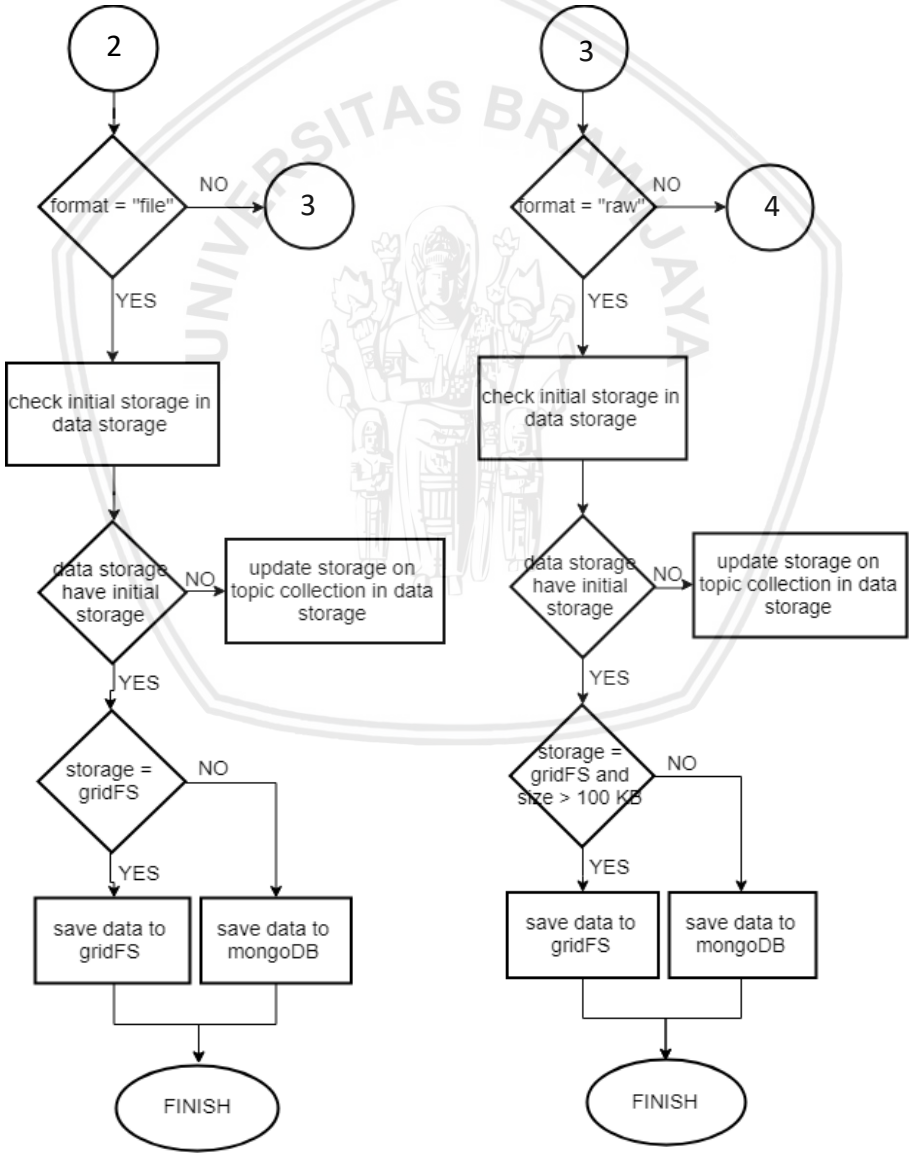
4.6.1 Perancangan Metode *Semantic Filtering*

Semantic filtering merupakan salah satu metode pada *semantic IoT web service* yang digunakan untuk menentukan data *storage* (MongoDB atau GridFS) melalui proses identifikasi tipe, struktur dan topik dari data yang dikirim oleh *client.* Dalam penelitian ini metode *semantic filtering* akan terhubung dangan API CoAP *server*. Data yang diterima dari CoAP *client* selanjutnya akan masuk ke dalam proses *semantic filtering* untuk kemudian disimpan ke dalam data *stroage* yang telah ditentukan. Terdapat dua data *storag*e yang digunakan dalam proses penyimpanan data yaitu MongoDB dan GridFS. MongoDB digunakan untuk menyimpan data terstruktur seperti JSON dan GridFS digunakan untuk menyimpan data yang tidak terstruktur seperi gambar maupun video. Alur proses *semantic filtering* dapat dilihat pada *flowchart* gambar 4.7.



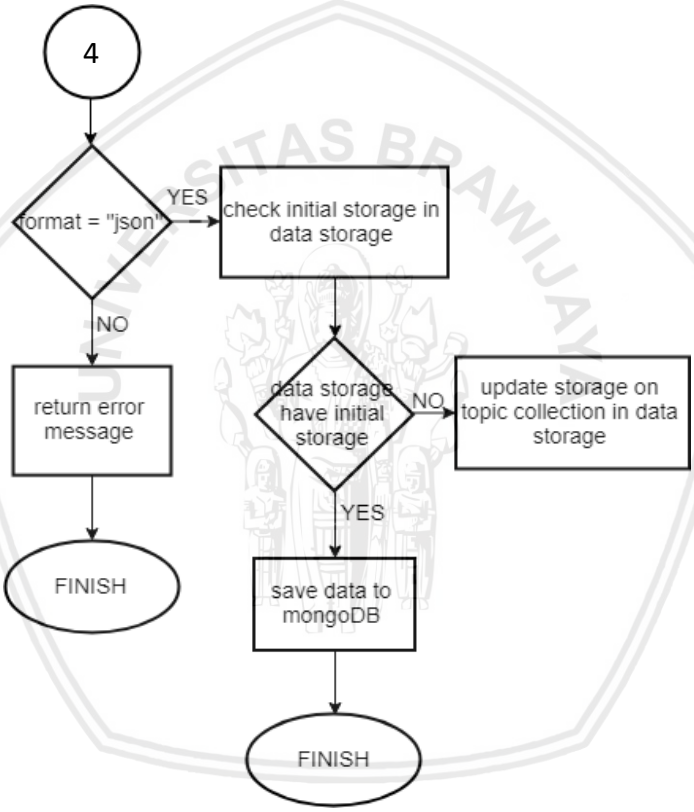
Gambar 4.7 Alur proses *semantic filtering*

Gambar diatas menunjukkan data yang telah diterima dari CoAP *client* akan masuk ke dalam proses *semantic filtering*. Selain data, CoAP *client* juga menyertakan JWT token dan topik. JWT token digunakan untuk proses otorisasi, sedangkan topik digunakan dalam proses penyimpanan data pada data *storage*. Pertama-tama akan dilakukan pengecekan *key* "Bearer" pada JWT token dan pengecekan *key* "topik". Jika kedua *key* tersebut tersedia, maka dilanjutkan dengan proses validasi topik. Apabila topik valid maka maka dilanjutkan dengan pengecekan apakah topik aktif dengan mengambil *value* dari *key* "status" pada data *storage* sesuai dengan topik yang dikirim oleh CoAP *client*. Dilanjutkan dengan pengecekan kompatibilitas dari struktur dan format dari data. Jika selama proses tersebut menghasilkan nilai FALSE, maka proses *semantic filtering* akan berhenti dan *semantic IoT web service* akan mengirimkan respon dengan status *error* kepada CoAP *client* melalui CoAP *server*.



Gambar 4.7 Alur proses *semantic filtering* (Lanjutan)

Gambar diatas merupakan lanjutan dari alur proses *semantic filtering*. Gambar tersebut akan befokus pada proses penyimpanan data pada data *storage*. Pada masing-masing proses penyimpanan tersebut, akan dilakukan pengecekan inisial *value* dari *key* "*storage*" dari data *storage*. *Value* "*storage*" dapat berupa "MongoDB" dan "GridFS" untuk menandakan kemana data yang diterima akan disimpan. Pada data dengan format *file* dan inisial *value* dari *key* "*storage*" adalah "GridFS", maka data akan disimpan pada GridFS. Pada data dengan format *raw* dan inisial *value* dari *key* "*storage*" adalah "GridFS" serta memiliki ukuran lebih dari 100 KB maka data tersebut juga akan disimpan pada GridFS. Selain dari proses seleksi penyimpanan tersebut, data akan disimpan ke dalam MongoDB. Sedangkan untuk data berbentuk JSON akan langsung disimpan ke dalam MongoDB.



Gambar 4.7 Alur proses *semantic filtering* (Lanjutan)

Gambar diatas merupakan lanjutan dari proses penyimpanan data pada data *storage*. Data dengan format JSON akan masuk terlebih dahulu ke dalam proses pencocokan *key value* serta panjang dari data pertama yang telah disimpan. Hal tersebut guna untuk mengetahui konsistensi dari data yang dikirimkan oleh CoAP *client*. Jika data yang dikirimkan sama atau konsisten, maka data tersebut akan disimpan ke dalam MongoDB.

4.6.2 Perancangan Mekanisme Otentikasi dan Otorisasi Menggunakan *JSON Web Token* (JWT)

JSON Web token merupakan salah satu metode yang digunakan pada *semantic IoT web service* untuk proses otentikasi dan otorisasi. *JSON Web Token* (JWT) akan terhubung dengan API CoAP *server*. Proses otentikasi dapat dilakukan dengan mengirimkan *token key* dan *secret key*. *Token key* dan *secret key* didapat jika telah mendaftarkan *device* pada IoT *application* (Pramukantoro, et al., 2019). Adapun alur dari proses otentikasi dapat dilihat pada *flowchart* gambar 4.8.

Gambar 4.8 Alur proses otentikasi menggunakan JWT

Dalam gambar 4.8 terdapat beberapa pengcekan yaitu pengecekan apakah *token key* dan *secret key* yang dikirimkan sama dengan *token key* dan *secret key* yang ada dalam data *storage*, pengecekan apakah *user* pemilik *token key* dan *secret key* aktif, lalu pengecekan apakah *device* dari *token key* dan *secret key* aktif, dan terakhir adalah pengecekan JWT token apakah telah dibuat dan tersimpan dalam data *storage* atau belum dibuat sama sekali. Jika JWT token belum dibuat maka API *JSON Web Token* akan membuat JWT token sesuai dengan struktur

payload JWT yang telah ditentukan dalam untuk kemudian disimpan ke dalam data *storage*. JWT token selanjutnya akan dikirim kembali kepada *client*.

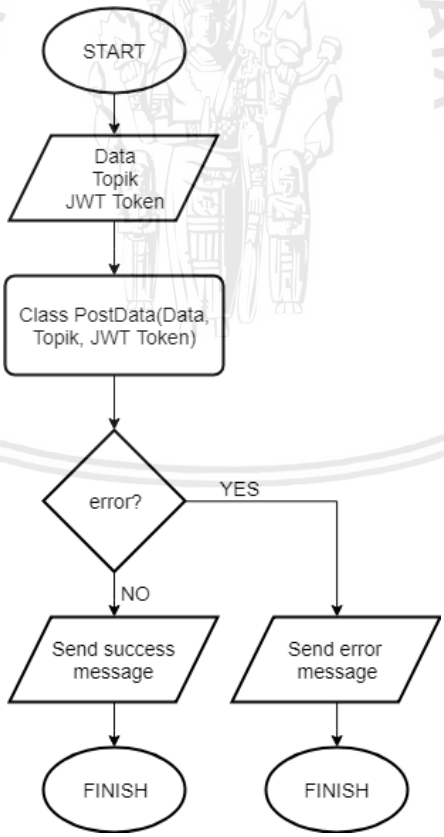
Proses selanjutnya adalah otorisasi yang akan melibatkan penggunaan JWT token. JWT token yang telah diterima CoAP *client* selanjutnya dikirim bersamaan dengan data maupun topik jika melakukan proses POST data kepada *semantic IoT web service*. JWT token yang diterima kemudian akan digunakan sebagai syarat otorisasi dalam menggunakan fungsi-fungsi pada proses *semantic filtering*.

4.6.3 Perancangan API CoAP *Server*

API CoAP *server* berada pada cloud yang akan bertugas meneruskan data yang dikirim oleh CoAP *client* kepada *semantic IoT web service*. Masing-masing komponen dalam API CoAP *server* akan diimplementasikan dalam bentuk kelas. Kelas-kelas tersebut akan terhubung dengan proses *semantic filtering* maupun proses otentikasi dan otorisasi menggunakan JWT pada *semantic IoT web service*.

4.6.3.1 Perancangan API CoAP *Server* Kelas PostData

API kelas PostData digunakan untuk menangani proses POST data yaitu menerima data dari CoAP *client*. Adapun alur proses pada API CoAP *server* kelas PostData dapat dilihat pada gambar 4.9.



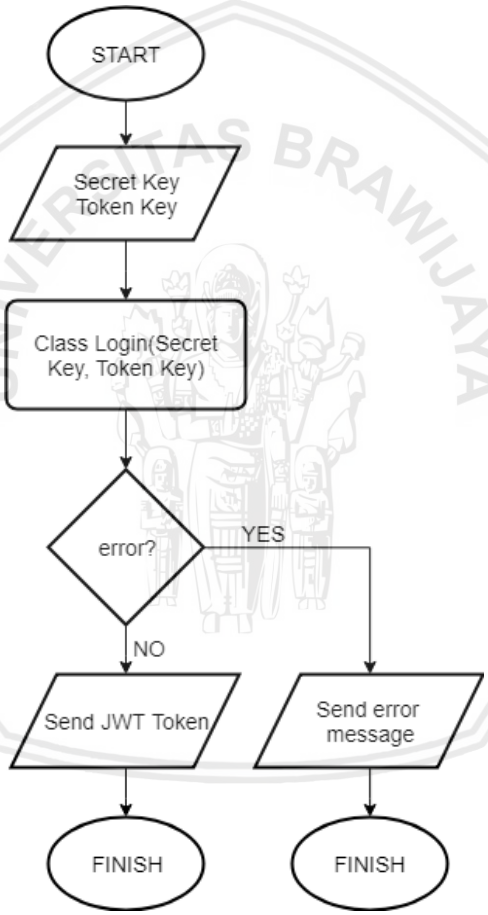
Gambar 4.9 Alur proses API CoAP *server* kelas PostData

Pada gambar 4.9, JWT Token, topik dan data yang dikirim oleh CoAP *client* kemudian akan masuk ke dalam kelas PostData. Dalam kelas PostData akan

dipanggil fungsi hasil implementasi dari metode *semantic filtering*. Jika proses *semantic filtering* berhasil maka data akan dapat disimpan ke dalam data *storage* dan *semantic IoT web service* akan mengirimkan respon dengan status berhasil kepada CoAP *client* melalui CoAP *server*. Sedangkan jika terjadi kegagalan, maka data tidak dapat disimpan dan *semantic IoT web service* akan mengirimkan respon dengan status gagal kepada CoAP *client*.

4.6.3.2 Perancangan API CoAP *Server* Kelas Login

API kelas login digunakan untuk proses otentikasi menggunakan *JSON Web Token*. Dalam API kelas login, akan dipanggil fungsi implementasi dari *JSON Web Token*. Adapun alur proses dari API CoAP *server* kelas login dapat dilihat pada gambar 4.10.

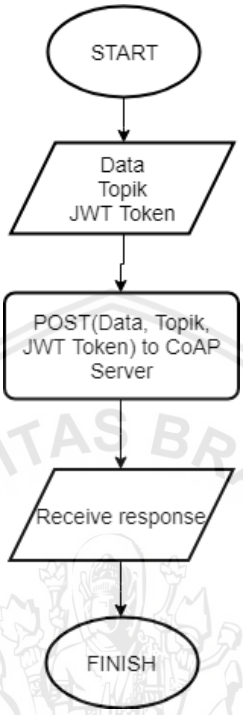


Gambar 4.10 Alur proses API CoAP *server* kelas Login

Dalam gambar 4.10 sebelum data dapat diproses pada kelas lainnya, CoAP *client* harus mengirimkan *token key* dan *secret key* sebagai syarat dalam proses otentikasi menggunakan JWT. Adapun proses dan mekanisme otentikasi telah dijelaskan dalam sub bab perancangan *JSON Web Token*. Khusus untuk mekanisme otorisasi, JWT Token yang telah dikirimkan oleh CoAP *client* kemudian digunakan sebagai syarat dalam menggunakan fungsi *semantic filtering*.

4.6.4 Perancangan API CoAP *Client*

CoAP *client* bertugas melakukan pengiriman atau POST data kepada CoAP *Server*. Adapun alur proses dari API CoAP *client* fungsi POST dapat dilihat pada gambar 4.11.



Gambar 4.11 Alur proses API CoAP *client* fungsi POST

Pada gambar 4.11 proses POST data dilakukan jika proses otentikasi pada *semantic IoT web service* berhasil yang ditandai dengan didapatkannya JWT token. JWT token akan digunakan untuk proses otorisasi pada *semantic filtering*. JWT Token tersebut kemudian dikirimkan bersamaan dengan topik serta data kepada CoAP *server* untuk diteruskan kepada *semantic IoT web service*. CoAP *client* juga akan menerima respon dari *semantic IoT web service* melalui CoAP *server* setiap melakukan proses POST data. Respon tersebut berupa respon dengan status berhasil dan respon dengan status gagal.

4.7 Perancangan Pengujian

Perancangan pengujian memuat mekanisme maupun skenario pengujian yang dilakukan pada sistem. Adapun pengujian yang dilakukan berupa pengujian fungsional, pengujian interoperabilitas dan pengujian kinerja.

4.7.1 Perancangan Pengujian Fungsional

Perancangan pengujian fungsional akan menguji kebutuhan fungsional sistem. Untuk mempermudah melakukan pengujian fungsional, sehingga dibuat skenario pengujian fungsional pada tabel 4.5.

Tabel 4.5 Skenario pengujian fungsional

Kode	Fungsi	Skenario Pengujian
CoAP_001	API *semantic IoT web service* dapat menerima data berbentuk JSON melalui protokol komunikasi CoAP	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud*. 2. Menjalankan kode textPost.py yang berisi data dalam bentuk JSON, JWT Token dan topik. Kode postText.py merupakan kode API CoAP di sisi *client.* 3. Menampilkan data yang dikirim melalui *log console* terminal *cloud.*
CoAP_002	API *semantic IoT web service* dapat menerima data berbentuk gambar melalui protokol komunikasi CoAP	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud*. 2. Menjalankan kode imagePost.py yang berisi data dalam bentuk gambar, JWT Token dan topik. Kode imagePost.py merupakan kode API CoAP di sisi *client.* 3. Menampilkan data yang dikirim melalui *log console* terminal *cloud.*
CoAP_003	API *semantic IoT web service* dapat menerima data berbentuk video melalui protokol komunikasi CoAP	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud.* 2. Menjalankan kode videoPost.py yang berisi data dalam bentuk video, JWT Token dan topik. Kode videoPost.py merupakan kode API CoAP di sisi *client.* 3. Menampilkan data yang dikirim melalui *log console* terminal *cloud*
CoAP_004	API *semantic IoT web service* yang telah ditambah protokol komununikasi CoAP dapat melakukan proses *semantic filtering.*	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud*. 2. Melakukan pengiriman data menggunakan API CoAP *client*. 3. Menampilkan status proses *semantic filtering* dari data yang telah tersimpan pada data *storage* dan *console* terminal CoAP *client*.

Tabel 4.5 Skenario pengujian fungsional (lanjutan)

CoAP_005	API *semantic IoT web service* yang telah ditambah protokol komunikasi CoAP dapat melakukan otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT).	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud*. 2. API CoAP *client* melakukan pengiriman *token key* dan *secret key* yang benar untuk proses otentikasi 3. API CoAP *client* melakukan pengiriman *token key* dan *secret key* yang salah untuk proses otentikasi 4. Menampilkan respon pada *console* terminal API CoAP *client* 5. API CoAP *client* melakukan pengiriman JWT token yang benar untuk proses otorisasi 6. API CoAP *client* melakukan pengiriman JWT token yang salah untuk proses otorisasi 7. Menampilkan respon pada *console* terminal API CoAP *client*

4.7.2 Perancangan Pengujian Interoperabilitas

Interoperabilitas merupakan kemampuan pertukaran dan pemrosesan informasi atau data antara dua buah sistem atau lebih (Desai, et al., 2015). Pengujian interoperabilitas bertujuan untuk mengetahui tingkat interoperabilitas dari *semantic IoT web service* yang terdapat protokol komunikasi HTTP dan CoAP. Metode pengujian yang dilakukan berdasarkan pada *interoperability assessment methodology* (Rezaei, et al., 2014) yang terdiri dari sembilan komponen pengujian. Namun khusus dalam pengujian pada penelitian ini akan diambil komponen "*Protocol*", "*Intepretation*" dan "*Information Untilization*".

"Interpretation" dan "*information untilization"* akan menjadi satu yaitu pengujian integritas data. Sedangkan pada "*protocol"* akan diuji proses pengiriman data secara independen melalui masing-masing protokol komunikasi CoAP dan HTTP serta pengiriman data secara bersama-sama kepada *semantic IoT web service*. Bentuk data yang digunakan dalam pengujian ini adalah JSON, *file* gambar dan *file* video. Adapun skenario pengujian interoperabilitas dapat dilihat pada tabel 4.6.

Tabel 4.6 Skenario pengujian interoperabilitas

Kode Pengujian	Parameter	Keterangan
PI_001	Pengiriman data	Melakukan proses pengiriman data melalui protokol komunikasi CoAP dan HTTP secara bersama-sama kepada *semantic IoT web service* serta melakukan pengiriman data secara independen melalui masing-masing protokol komunikasi CoAP serta HTTP kepada *semantic IoT web service*.
PI_002	*Monitoring* pengiriman data	Selama proses pengiriman data akan dilakukan *monitoring* menggunakan tcpdump untuk kemudian dianalisis pada wireshark.
PI_003	Pengecekan model data yang dikirim dengan yang diterima	Melakukan validasi model data yang dikirim oleh *client* dengan model data yang diterima pada *semantic IoT web service*. Akan terdapat dua model data yaitu model data yang dikirim melalui CoAP dan dikirim melalui HTTP.
PI_004	Data yang telah diterima dapat diproses dengan benar	Melakukan verifikasi terhadap data yang diterima pada *semantic IoT web service* apakah telah tersimpan dalam data *storage* dan dapat ditampilkan dalam IoT *applications*.

4.7.3 Perancangan Pengujian Kinerja

Pengujian kinerja dilakukan dengan mengirimkan data melalui masing-masing protokol secara bersamaan berdasarkan skenario pengujian yaitu 50, 100, 150, 200 dan 250. Data yang dikirim dalam pengujian ini adalah data berbentuk JSON. Parameter yang akan diobservasi yaitu *runtime*, *throughput*, penggunaan CPU dan penggunaan memori. Penguijian ini akan menggunakan *library* psutil dari python yang dikhususkan dalam mengambil data penggunaan memori dan penggunaan CPU. Sedangkan dalam mengambil data *runtime* dan *throughput* menggunakan library *time*. *Pseudocode* 4.6, 4.7 dan 4.8 merupakan kode progran yang digunakan untuk menguji parameter penggunaan CPU, penggunaan memori serta *runtime* dan *throughput*. Hasil dari proses pengujian akan dibahas pada bab pengujian dan pembahasan hasil pengujian. Untuk mempermudah proses pengujian maka dibuat skenario pengujian yang ditampilkan pada tabel 4.7.

Pseudocode 4.1 Program pengujian *CPU usage*

```
cpu.py
1   SET psutil
2   DO WHILE:
3     SET mem : get CPU usage from semantic IoT web service
4     DO PRINT mem
5     DO wait 1 second
```

Pseudocode 4.1 merupakan kode yang digunakan untuk mengambil data penggunaan CPU dari *semantic IoT web service* pada *cloud* saat menerima data dari CoAP *client* melalui protokol CoAP.

Pseudocode 4.2 Program pengujian *memory usage*

```
memory.py
1   SET psutil
2   DO WHILE:
3     SET mem : get memory usage from semantic IoT web service
4     DO PRINT mem
5     DO wait 1 second
```

Pseudocode 4.2 merupakan kode yang digunakan untuk mengambil data penggunaan memori dari *semantic IoT web service* pada *cloud* saat menerima data dari CoAP *client* melalui protokol CoAP.

Pseudocode 4.3 Program pengujian *runtime* dan *throughput*

```
runtimethroughput.py
1   SET time
2
3   SET start_time : current time
4   main()
5   .
6   .
7   SET runtime : current time - start_time
8   SET throughput = data / runtime
9   DO PRINT runtime
10  DO PRINT throughput
```

Pseudocode 4.3 merupakan kode yang digunakan untuk mengambil data *runtime* dan *throughput*. Kode tersebut dijalankan bersamaan saat CoAP *client* melakukan POST data kepada *semantic IoT web service*. Perhitungan *runtime* dalam kode tersebut adalah waktu yang dibutuhkan CoAP *client* dalam proses pengiriman data. Sedangkan untuk menghitung *throughput* adalah banyaknya operasi pengiriman data dibagi dengan *runtime*.

Tabel 4.7 Skenario pengujian kinerja

No	Parameter Pengujian	Skenario Pengujian
1	Penggunaan memori	1. Menjalankan kode memoryusage.py yang merupakan kode program untuk mengambil data penggunaan memori setiap 1 detik sekali. Kode tersebut dijalankan pada *cloud*. Nilai penggunaan memori yang diambil kemudian disimpan kedalan sebuah file .txt. 2. Melakukan POST data kepada *semantic IoT web service* melalui protokol CoAP dan HTTP secara bersamaan dengan skenario pengujian sebesar 50, 100, 150, 200 dan 250 data secara bergantian. 3. Mencari Q1, Median, Q3, nilai minimum, dan maksimum dari masing-masing hasil skenario pengujian untuk kemudian dirubah menjadi grafik *whisker plot*
2	Penggunaan CPU	1. Menjalankan kode cpuusage.py yang merupakan kode program untuk mengambil data penggunaan CPU setiap 1 detik sekali. Kode tersebut akan dijalankan pada *cloud*. Nilai penggunaan CPU yang diambil kemudian disimpan kedalan sebuah file .txt. 2. Melakukan POST data kepada *semantic IoT web service* melalui protokol CoAP dan HTTP secara bersamaan dengan skenario pengujian sebesar 50, 100, 150, 200 dan 250 data secara bergantian. 3. Mencari Q1, Median, Q3, nilai minimum, dan maksimum dari masing-masing hasil skenario pengujian untuk kemudian dirubah menjadi grafik *whisker plot*

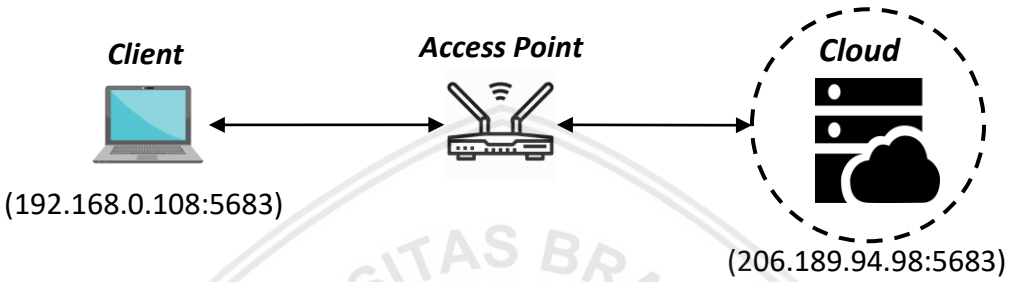
Tabel 4.7 Skenario pengujian kinerja (lanjutan)

3	*Runtime* dan *Throughput*	1. Menjalankan kode masing-masing *client* yang didalamnya terdapat baris kode untuk mengambil data *runtime* dan *throughput*. *Runtime* diperoleh dari keseluruhan waktu pengiriman data. Sedangkan *throughput* diperoleh dari *runtime* yang dibagi jumlah pengiriman data atau masing-masing skenario pengujian. 2. Melakukan POST data kepada *semantic IoT web service* melalui protokol CoAP dan HTTP dengan skenario sebesar 50, 100, 150, 200 dan 250 data secara bergantian. 3. Hasil *runtime* dan *throughput* akan ditampikan dalam *console* CoAP *client*. 4. POST data dari masing-masing skenario pengujian adalah sebanyak 10x per protokol komunikasi, sehingga akan didapatkan nilai *runtime* dan *throughput* masing-masing protokol sebanyak 10 data. 10 data CoAP dan 10 data HTTP tersebut kemudian dicari rata-ratanya. 5. Mencari Q1, Median, Q3, nilai minimum, dan maksimum dari masing-masing hasil skenario pengujian untuk kemudian dirubah menjadi grafik *whisker plot*.

BAB 5 IMPLEMENTASI

Implementasi merupakan tahapan penting dalam penelitian. Implementasi dilakukan berdasarkan pada perancangan yang telah dilakukan sebelumnya yaitu perancangan topologi jaringan dan perancangan penambahan CoAP dengan *semantic IoT web service*.

5.1 Implementasi Topologi Jaringan



Gambar 5.1 Implementasi topologi jaringan

Gambar 5.1 merupakan implementasi topologi jaringan dari penambahan CoAP pada *semantic IoT web service*. Sesuai dengan yang telah dijelaskan dalam bab perancangan, *client* menggunakan *private* IP yaitu 192.168.0.108, sedangkan pada *cloud* menggunakan *public* IP yaitu 206.189.94.98. Agar kedua komponen tersebut dapat berkomunikasi melalui protokol CoAP, maka port yang digunakan adalah 5683.

```
iotapps@IoTApps: ~/coap
(coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 206.189.94.98:5683
```

Gambar 5.2 Menjalankan program *semantic IoT web service*

```
iotapps@IoTApps:~$ netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 IoTApps:5683            0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
```

Gambar 5.3 Netstat pada *cloud data storage*

API CoAP *server* kemudian dijalankan pada *cloud*. Untuk membuktikan bahwa API tersebut telah berjalan dapat dilihat pada gambar 5.2 dan 5.3. Gambar 5.2 menunjukkan API CoAP *server* yang telah "*listening*" serta berjalan pada port 5683 sehingga siap untuk menerima data dari *client*. Gambar 5.3 perintah netstat yang merupakan bukti dari API CoAP *server* siap terhubung atau menerima data dari CoAP *client*.

5.2 Implementasi Penambahan CoAP pada *Semantic IoT Web Service*

Implementasi penambahan CoAP pada *semantic IoT web service* dalam bentuk *Application Programming Interface* (API) mengguanakn *library* CoAPthon. Khusus pada API CoAP di sisi *server*, akan diimplementasikan beberapa kelas yang kemudian akan terhubung dengan metode *semantic filtering* serta metode otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT) dari *semantic IoT web service* penelitian sebelumnya (Pramukantoro, et al., 2019).

5.2.1 Implementasi Metode *Semantic Filtering*

Semantic filtering merupakan salah satu metode dari *semantic IoT web service*. Karena metode *semantic filtering* akan terintegrasi dengan API CoAP server, maka metode ini akan diimplementasikan kembali dalam fungsi *store*. Implementasi *semantic filtering* dapat dilihat dalam *pseudocode* 5.1.

Pseudocode 5.1 Fungsi store

```
funciton: store
1   DEFINE function store
2     SET payloads = data from CoAP client
3     SET token = get "jwt token" from payloads
4
5     DO check "Bearer in token
6     DO check "topic" in payloads
7
8     SET topic = get "topic" value from payloads
9
10    DO function validateTopic(topic, token)
11
12    IF validateTopic = TRUE
13      DO function checkActiveTopic(topic, token)
14
15      IF checkActiveTopic = FALSE
16        DO return payload = error
17
18      SET structure = function getAtributeTopic(topic, token)
19      SET formats = function checkFormat(payloads)
20
21      DO function isCompatible(structure, format)
22
23      IF isCompatible = FALSE
24        DO return payload = error
25
26      IF formats = "file"
27        SET temp = temporary file that contain payloads
28          DO function checkFile(temp)
29          IF checkFile = TRUE
30            DO save file to GridFS
31          ELSE
32            DO save file to MongoDB
33
34      ELSE IF formats = "raw"
35        SET size = GET size from raw
36        IF size more than 100 KB
37          DO save raw to GridFS
```

Pseudocode 5.1 Fungsi store (Lanjutan)

```
funciton: store
38          ELSE
39            DO save raw to MongoDB
40
41        ELSE IF formats = "JSON"
42          DO check payloads consistency
43          IF payload consistent
44            DO save JSON to MongoDB
45          ELSE
46             DO return payload = error
47
48        ELSE
49          DO return payload = error
50
51      ELSE
52        DO return payload = error
```

Implementasi dari fungsi *semantic filtering* bernama *store*. Adapaun penjelasan dari *pseudocode* 5.1 di masing-masing baris adalah sebagai berikut:

- **Baris 1**: Deklarasi fungsi dengan nama *store*
- **Baris 2 – 3**: Inisialisasi variabel *payloads* dan variabel token. Variabel *payloads* merupakan variabel yang berisi data JWT Token, topik serta data yang berbentuk JSON. Sedangkan variabel token merupakan variabel yang berisi JWT token yang diambil dari variabel *payloads*.
- **Baris 5 – 6**: Pengecekan apakah komponen "Bearer" terdapat didalam *payload* token dan variabel "topic" terdapat dalam data yang dikirimkan. Jika salah satu atau kedua proses pengecekan tersebut bernilai FALSE maka akan mengembalikakan pesan *error*.
- **Baris 8**: mengambil isi *topic* dari variabel *payload*
- **Baris 10 – 12**: Proses validasi *topic* dengan fungsi validateTopic() yang berisi parameter topik dan JWT token. Proses validasi tersebut akan menghasilkan nilai TRUE atau FALSE. Jika TRUE maka akan dilanjutkan dengan proses berikutnya, jika FALSE maka akan mengembalikan pesan *error*.
- **Baris 13 – 16**: Pengecekan topik diberikan apakah aktif atau tidak menggunakan fungsi checkActiveTopic(), Proses tersebut akan mengembalikan nilai TRUE jika berhasil dan akan mengembalikan nilai FALSE jika gagal serta mengembalikan pesan *error*.
- **Baris 18 – 24**: Dalam baris tersebut terdapat inisialisasi variabel *structure* dan *formats*. Variabel *structure* berisi struktur dari topik yang dikirimkan untuk kemudian dibandingkan dengan struktur dari topik yang didaftarkan dalam data *storage*. Topik sebelumnya harus terdaftar dalam *IoT applications*. Proses pendaftaran tersebut harus menentukan *structure* dari topik yang didaftarakan. Sedangkan variabel *formats* berisi fungsi checkFormat() untuk mengetahui format dari data yang dikirimkan CoAP

client. Kedua variabel tersebut kemudian dimasukkan sebagai parameter dalam fungsi isCompatible(). Dalam fungsi isCompatible() terdapat pengecekan dari struktur serta format yang diberikan. Jika format dari data adalah JSON dan struktur adalah *structured* maka akan mengembalikan nilaii TRUE, jika format dari data adalah *file*, *raw* dan JSON dan struktur adalah *non-structured* maka akan mengembalikan nilai TRUE, selain itu maka akan mengembalikan nilai FALSE. Jika hasil pengecekan pada fungsi isCompatible() adalah FALSE, maka akan mengembalikan pesan *error*

- **Baris 26 – 32**: Baris tersebut merupakan baris yang digunakan untuk melakukan penyimpanan data yang memiliki format *file*. Didalam baris tersebut terdapat pengecekan data yang berbentuk *pickle* untuk dikenali sistem pada fungsi checkFile(). Jika data telah dikenali maka dimulai proses penyimpanan data pada data *storage* dengan melakukan inisial "storage" yang digunakan untuk menyimpan. Proses inisial hanya berlaku jika *collection* topik di dalam data *storage* tidak memiliki *key* dan *value* "storage" sehingga *collection* tersebut akan di perbaharui dengan menambahkan inisial "storage". Dalam kasus ini yaitu penyimpanan data dengan format *file*, maka dilakukan inisial *key* "storage" dengan value "gridFS" yang artinya data berbentuk *file* yang dikenali oleh sistem akan disimpan ke dalam GridFS. Jika parameter *storage* sudah ada maka akan langsung menuju proses penyimpanan data ke dalam data *storage*. Jika data tidak dikenali sistem, maka secara otomatis akan disimpan ke dalam Mongodb. Proses tersebut sama dengan proses sebelumnya yaitu akan melakukan inisial terlebih dahulu terhadap *collection* topik yang ada dalam data *storage* jika belum ditemukan *key value* dari *storage*.
- **Baris 34 – 39**: Baris tersebut di khususkan untuk penyimpanan data dengan format *raw*. Format *raw* merupakan format yang tidak teridentifikasi oleh sistem. Sama halnya dalam proses penyimpanan data berbentuk *file*, sebelum data dapat disimpan, maka akan dilakukan proses inisial "storage" terlebih dahulu jika *key value "storage"* belum ada dalam *collections topics* dalam data *storage*. Dalam proses inisial akan terdapat seleksi kembali dengan menyeleksi ukuran dari data yang dikirimkan. Jika ukuran data lebih dari 100 KB, maka inisial adalah "gridFS", selain itu adalah "mongodb". Selanjutnya adalah proses penyimpanan data dengan bentuk raw dan memiliki ukuran lebih dari 100 KB ke dalam GridFS. Jika ukuran data kurang dari 100 KB maka akan disimpan ke dalam MongoDB.
- **Baris 41 – 46**: Baris tersebut merupakan baris yang digunakan untuk penyimpanan data dengan format JSON. Sama dengan proses sebelumnya, pertama-tama akan dilakukan proses inisial *storage* jika belum ada terdapat *key value storage* dalam *collections topics*. Inisal *"storage"* untuk data dengan fomat JSON adalah "MongoDB". Selanjutnya adalah pengecekan konsistensi data yang dikirimkan oleh client dengan membandingakan data yang dikirim dengan data pertama JSON yang sebelumnya telah tersimpan dalam data *storage*. Proses ini dilakukan

untuk memastikan konsistensi data. Jika data kedua dan seterusnya sama maka dilakukan penyimpanan ke dalam data *storage* MongoDB. Jika pada proses tersebut terjadi *error*, maka akan proses tersebut akan mengembalikan nilai *error*.

Adapun contoh dari hasil implementasi dari metode *semantic filtering* dapat dilihat pada gambar 5.4.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}

{"message": "Finished Saving", "format": "json", "storage": "mongodb",
"size": 66}
```

Gambar 5.4 Hasil *semantic filtering* data JSON

Gambar 5.4 merupakan contoh hasil respon dari CoAP *server* pada CoAP *client* setelah melakukan pengiriman data JSON melalui proses *semantic filtering*. Dalam gambar tersebut, format data yang dikirimkan adalah JSON dan data *storage* yang digunakan sebagai media penyimpanan data adalah MongoDB.

5.2.2 Implementasi Mekanisme Otentikasi dan Otorisasi Menggunakan *JSON Web Token* (JWT)

JSON Web Token merupakan mekanisme otentikasi dan otorisasi pada semantic IoT web service. Karena metode JWT akan terintegrasi dengan API CoAP *server*, maka metode ini akan diimplementasikan kembali dalam fungsi *login*.. Adapun implementasi otentikasi menggunakan *JSON Web Token* dapat dilihat dari *pseudocode* 5.2

Pseudocode 5.2 Otentikasi menggunakan *JSON Web Token*

```
function: login
1   DEFINE funtion login with parameter token and secret key
2     DO checkDevice(token, secret)
3     IF checkDevice = TRUE
4       DO checkUser(token, secret)
5       IF checkUser = TRUE
6         DO deviceStatus(token, secret)
7         IF deviceStatus = TRUE
8           SET deviceID = find device ID in data storage
9           SET userID = find user ID in data storag
10          SET devices = find device in data storage
11             IF "jwt_active" not in devices
12               DO function createJWT
13             ELSE
14               DO RETURN JWT Token
```

Pseudocode 5.2 Otentikasi menggunakan *JSON Web Token* (Lanjutan)

```
function: login
15          ELSE
16            RETURN payload = error
17        ELSE
18          RETURN payload = error
19      ELSE
20        RETURN payload = error
```

Pseudocode 5.2 merupakan implementasi mekanisme otentikasi menggunakan *JSON Web Token* dengan nama login. Penjelasan dari *pseudocode* 5.2 adalah sebagai berikut:

- **Baris 1**: deklarasi nama fungsi login yang didalamnya terdapat parameter berupa *token key* dan *secret key*
- **Baris 2 – 3**: proses validasi *device* pada data *storage* menggunakan *token key* dan *secret key* yang dikirimkan menggunakan fungsi checkDevice
- **Baris 4 – 5**: proses pengecekan apakah status dari user aktif pada data *storage* dengan menggunakan *token key* dan *secret key.* proses tersebut menggunakan fungsi checkUser
- **Baris 6 – 7**: proses pengecekan apakah status dari *device* aktif pada data *storage* dengan menggunakan *token key* dan *secret key*. proses tersebut menggunakan fungsi deviceStatus
- **Baris 8 – 14**: proses untuk melakukan pengecekan apakah JWT token telah dibuat dan telah tersimpan dalam data *storage*. jika JWT token belum dibuat maka akan dijalakan fungsi createJWT. Fungsi createJWT akan membuatkan jwt token berdasarkan *payload* yang telah ditentukan yang kemudian akan disimpan ke dalam data *storage*. jika jwt token telah dibuat dan tersimpan maka fungsi login akan mengembalikan jwt token kepada *client*.

Tahap otorisasi dilakukan dalam fungsi *store*. Adapun contoh dari hasil metode otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT) dapat dilihat pada gambar 5.5.

```
rezananda@REZANANDA:/mnt/c/Users/Reza      Nanda/OneDrive/Documents/web
service/RestCoAP$                  python                  textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message":          "login          successfully",          "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
```

Gambar 5.5 Hasil proses otentikasi dan otorisasi menggunakan JWT

```
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}

{"message": "Finished Saving", "format": "json", "storage": "mongodb",
"size": 66}
```

Gambar 5.5 Hasil proses otentikasi dan otorisasi menggunakan JWT (Lanjutan)

Gambar 5.5 merupakan hasil proses otentikasi dan otorisasi menggunakan JWT. Gambar dengan *highlight* berwarna hijau merupakan hasil proses otentikasi berhasil yaitu didapatkannya JWT token pada CoAP *client*. Sedangkan pada *highlight* berwarna kuning merupakan hasil proses otorisasi berhasil, dibuktikan dengan data yang telah berhasil disimpan pada data *storage* yang melalui proses *semantic filtering*.

5.2.3 Implementasi API CoAP *Server*

API CoAP *server* bertugas untuk menerima data dan meneruskannya kepada fungsi otentikasi dan otorisasi serta fungsi *semantic filtering*. Terdapat dua kelas utama dalam API CoAP *server* yaitu kelas untuk otentikasi dan kelas untuk POST data.

5.2.3.1 Implementasi API CoAP *Server* Kelas PostData

API CoAP *server* kelas postData merupakan kelas yang digunakan untuk menerima data yang dikirimkan oleh CoAP *client*. Pada API ini akan dipanggil fungsi "*store*" yang merupakan implementasi dari metode *semantic filtering*. Data yang telah diterima kemudian dialihkan pada fungsi *store* untuk dilakukan proses *semantic filtering*. Adapun implementasi dari kelas PostData dapat dilihat pada *pseudocode* 5.3.

Pseudocode 5.3 API CoAP *server* Kelas PostData

```
Class: PostData
1   DEFINE class PostData:
2     DEFINE function rander_post_advance()
3       SET data = get data from CoAP client
4
5       SET stores = function store(data)
6
7         IF error in stores:
8           Payload = error
9           DO return payload
10        ELSE
11          Payload = succeed
12          DO return payload
13    DEFINE function render_GET_advanced()
14       SET response = payload
15       DO send response to CoAP client
```

Penjelasan dari pseudocode 5.3 tersebut adalah sebagai berikut:

- **Baris 1**: Definisi kelas PostData

- **Baris 2**: Definisi fungsi rander_post_advance(). Fungsi tersebut digunakan untuk menerima data dari CoAP *client*.
- **Baris 3**: Menerima data dari CoAP *client*
- **Baris 5 – 12**: Melakukan proses *semantic filtering* dengan memanggil fungsi *store*. Jika pada proses *semantic filtering* terjadi *error*, maka akan proses tersebut akan mengembalikan pesan *error*, jika berhasil akan mengembalikan pesan yang berisi pesan berhasil.
- **Baris 13 – 15**: Definisi fungsi rander_get_advance(). Fungsi tersebut digunakan untuk mengirimkan data yang diterima dari fungsi rander_post_advance() kepada CoAP *client*.'

Adapun contoh dari kelas postData dalam menerima data dari CoAP *client* dapat dilihat pada gambar 5.6.

```
(coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 0.0.0.0:5683
2019-04-07 06:51:03,974 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28489,
POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,974 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28489, POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,989 - Thread-5    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 06:51:03,989 - Thread-5    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CREATED-kM, [] No payload
2019-04-07 06:51:03,989 - Thread-5    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28489,
CREATED-kM, [] No payload
2019-04-07 06:51:04,036 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28490,
GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,037 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28490, GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,038 - Thread-7    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 06:51:04,038 - Thread-7    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28490,
CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 06:51:04,082 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28491,
POST-tY, [Uri-Path: postdata, ] {"topic": "jsontopic...495 bytes
2019-04-07 06:51:04,082 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28491, POST-tY, [Uri-Path: postdata, ] {"topic": "jsontopic...495 bytes
2019-04-07 06:51:04,111 - Thread-9    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 06:51:04,111 - Thread-9    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CREATED-tY, [] No payload
2019-04-07 06:51:04,112 - Thread-9    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28491,
CREATED-tY, [] No payload
```

Gambar 5.6 Implementasi API CoAP *server* kelas PostData

```
2019-04-07 06:51:04,152 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28492,
GET-None, [Uri-Path: postdata, ] No payload
2019-04-07 06:51:04,152 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28492, GET-None, [Uri-Path: postdata, ] No payload
2019-04-07 06:51:04,153 - Thread-11  - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CONTENT-None, [] {"message": "Finishe...82 bytes
2019-04-07 06:51:04,153 - Thread-11  - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28492,
CONTENT-None, [] {"message": "Finishe...82 bytes
```

Gambar 5.6 Implementasi API CoAP *server* kelas PostData (Lanjutan)

Gambar 5.6 merupakan implementasi dai API CoAP *server* kelas PostData. Konsep *threading* diterapkan dalam CoAP *server* tersebut untuk proses penerimaan data dari CoAP *client*. *Highlight* berwarna kuning pada gambar tersebut merupakan contoh proses penerimaan data dalam bentuk JSON. Data tersebut kemudian akan masuk ke dalam proses *semantic filtering*. Setelah data tersimpan dalam data *storage*, API CoAP *server* kelas PostData juga akan mengirimkan respon kepada CoAP *client,* seperti pada *highlight* berwarna biru pada gambar 5.6. Respon tersebut dapat berupa respon dengan status *error* atau status berhasil.

5.2.3.2 Implementasi API CoAP *Server* Kelas Login

Kelas Login merupakan kelas pada API CoAP *server* yang bertugas melakukan otentikasi terhadap CoAP *client*. Fungsi login akan dipanggil dalam kelas ini yang akan menangani proses otentikasi lebih lanjut. Adapun implmenetasi dari kelas Login dapat dilihat pada pseudocode 5.4.

Pseudocode 5.4 Kelas Login

```
Class: Login
1   DEFINE class Login:
2     DEFINE function rander_post_advance()
3       SET data = get data from CoAP client
4       SET token = get token key from data
5       SET secret = get secret key from data
6
7       SET logins = function login(secret, token)
8
9       IF error in logins:
10        Payload = error
11        RETURN payload
12      ELSE
13        Payload = succeed
14        RETURN payload
15    DEFINE function render_GET_advanced()
16      SET response = payload
17      SEND response to CoAP client
```

Penjelasan dari *pseudocode* 5.4 adalah sebagai berikut:

- **Baris 1**: Definisi kelas PostData
- **Baris 2**: Definisi fungsi rander_post_advance(). Fungsi tersebut digunakan untuk menerima token key dan secret key dari CoAP *client*.
- **Baris 3**: Menerima token key dan secret key dari CoAP *client*
- **Baris 4 – 5**: definisi variabel token dan secret yang berisi token key dan secret key.
- **Baris 7 – 14**: Melakukan proses otentikasi menggunakan fungsi login dengan parameter berupa token key dan secret key. Jika pada proses otentikasi terjadi *error*, maka akan proses tersebut akan mengembalikan pesan *error*, jika berhasil akan mengembalikan JWT token kepada CoAP *client*.
- **Baris 15 – 17**: Definisi fungsi rander_get_advance(). Fungsi tersebut digunakan untuk mengirimkan JWT Token atau pesan *error* yang diterima dari fungsi rander_post_advance() kepada CoAP *client*.

Adapun contoh dari kelas Login dalam melakukan otentikasi dan otorisasi dapat dilihat pada gambar 5.7.

```
(coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 0.0.0.0:5683
2019-04-07 06:51:03,974 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28489,
POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,974 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28489, POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,989 - Thread-5 - coapthon.server.coap - DEBUG -
Notify
2019-04-07 06:51:03,989 - Thread-5 - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CREATED-kM, [] No payload
2019-04-07 06:51:03,989 - Thread-5 - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28489,
CREATED-kM, [] No payload
2019-04-07 06:51:04,036 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28490,
GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,037 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28490, GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,038 - Thread-7 - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 06:51:04,038 - Thread-7 - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28490,
CONTENT-None, [] {"message": "login s...448 bytes
```

Gambar 5.7 Implementasi API CoAP *server* kelas Login

Gambar 5.7 merupakan hasil implementasi dari API CoAP *server* kelas login. *Highlight* berwarna kuning tersebut merupakan proses otentikasi menggunakan JWT. Konsep *threading* diterapkan dalam CoAP *server* kelas login untuk dalam menerima *token key* dan *secret key* dari CoAP *client* serta

mengirimkan JWT token kepada CoAP *client*. Proses otentikasi berlangsung pada fungsi login yang dipanggil pada kelas tersebut. Jika proses otentikasi tersebut berhasil, CoAP *server* kelas login tersebut akan mengirimkan JWT token kepada CoAP *client*. Namun jika proses otentikasi gagal, akan dikirimkan pesan *error* kepada CoAP *client*.

5.2.4 Implementasi API CoAP *Client*

API *Client* digunakan untuk melakukan otentikasi dan pengiriman data kepada CoAP *server*. Adapun implementasi dari API CoAP *client* fungsi POST dapat dapat dilihat pada *pseudocode* 5.5.

Pseudocode 5.5 API CoAP *Client* Fungsi POST

```
1   SET host CoAP server
2   SET port CoAP server
3   SET path postData
4   SET path login
5
6   SET login data = get token key and secret key
7
8   DO POST(login path, login data) to CoAP server
9
10  DO GET(response) from CoAP server as JWT Token
11
12  IF GET(response) = error
13    DO print (response)
14    STOP
15
16  WHILE
17    SET post data = JWT Token, topic, payload
18
19    DO POST(postData path, post data) to CoAP server
20
21    DO GET(response) from CoAP server as payload
22
23    IF GET(response) = error
24      DO print (response)
25      STOP
```

Pembahasan dari *pseudocode* 5.5 tersebut adalah sebagai berikut:

- **Baris 1 – 4**: Insialisasi variable *host*, *port*, *path* “login” dan *path* postData. Variavel *Host* yang berisi IP dari VPS, variabel *port* yang berisi *port* dari CoAP yaitu 5683, variabel *path* “/postData” digunakan untuk *routing* proses POST data kepada CoAP *server* serta *path* “/login” digunakan untuk *routing* dalam proses otentikasi terhadap CoAP *server*.
- **Baris 6 – 14**: Inisialisasi *token key* dan *secret key* untuk kemudian dikirim dankepada CoAP *server* dengan *path* berupa “/login” dan menerima respon dari CoAP *server* berupa JWT token. Proses tersebut merupakan proses otentikasi sebelum CoAP *client* dapat mengambil data dari CoAP *server*. Jika proses otentikasi gagal maka CoAP *server* akan mengirimkan pesan *error* dan secara otomatis API CoAP *client* akan berhenti.

- **Baris 16 – 25**: Proses POST data yang mula-mula melakukan inisialisasi komponen JWT Token, topik serta data. Ketiga komponen tersebut kemudian di POST kepada CoAP *server*. Jika proses POST gagal maka CoAP *server* akan mengirimkan pesan *error* dan secara otomatis API CoAP *client* akan berhenti

Adapun contoh proses pengiriman data pada CoAP *client* beserta respon yang diterima dapat dilihat pada gambar 5.8

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50
```

Gambar 5.8Implementasi API CoAP *client*

```
Attention! if your topic has spaces, please add "_" (uderscore)
{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}
{"message": "Finished Saving", "format": "json", "storage": "mongodb",
"size": 66}
```

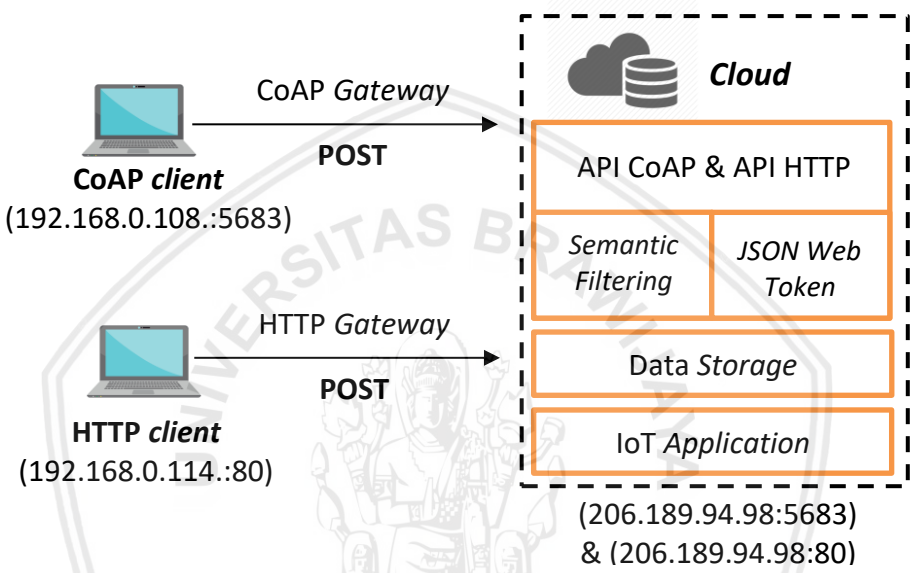
Gambar 5.8 Implementasi API CoAP *client* (Lanjutan)

Gambar 5.8 merupakan implementasi dari API CoAP *client* yang bertugas untuk melakukan pengiriman data kepada CoAP *server*. Sebelum data dapat dikrim, CoAP *client* harus melakukan otentikasi terlebih dahulu menggunakan *token key* dan *secret key* kepada fungsi JWT. Jika proses otentikasi berhasil, CoAP *server* lalu meneruskan respon dari *semantic IoT web service* dengan mengirimkan JWT token kepada CoAP *client* seperti pada gambar 5.8 yang di *highlight* berwarna kuning. Sebaliknya jika proses otentikasi gagal, fungsi JWT akan mengirimkan pesan *error*. JWT token tersebut kemudian dikirimkan bersamaan dengan data dan topik sebagai syarat proses otorisasi pada fungsi *semantic filtering*. Jika proses *semantic filtering* berhasil, maka CoAP *server* akan meneruskan respon dengan status berhasil kepada CoAP *client* sesuai dengan gambar 5.6 yang di *highlight* berwarna biru. Namun jika proses penyimpanan gagal, maka CoAP *client* akan mengirimkan respon dengan status gagal.

BAB 6 PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

Pada bab pengujian akan dilakukan pengujian berdasarkan perancangan pengujian serta skenario pengujian yang telah dideskripsikan pada bab perancangan. Adapun pengujian yang dilakukan yaitu pengujian fungsional, pengujian interoperabilitas dan pengujian kinerja yang meliputi penggunaan memori, penggunaan CPU, *throughput* dan *runtime*.

6.1 Lingkungan Pengujian



Gambar 6.1 Lingkungan pengujian

Gambar 6.1 merupakan lingkungan pengujian yang digunakan untuk menguji sistem. Seperti yang telah dijelaskan dalam perancangan pengujian, pengujian yang dilakukan adalah pengujian secara fungsional, pengujian interoperabilitas dan pengujian kinerja. Dalam gambar 6.1 terdapat tiga komponen yaitu dua *client* yang akan mengirimkan data melalui protokol CoAP dan HTTP, serta *cloud* yang merupakan sebuah VPS. *Client* menggunakan *private* IP yang berbeda serta *port* yang berbeda yaitu 5683 untuk CoAP dan 80 untuk HTTP. Masing-masing *client* akan mengirimkan data melalui media *wireless* dan dengan data *dummy* sesuai dengan kebutuhan pengujian.

Pada sisi *cloud*, terdapat komponen berupa API *semantic IoT web service*, API CoAP server, data *storage* serta IoT *applications*. API *semantic IoT web service* akan menerima data melalui protokol HTTP dan API CoAP server akan menerima data melalui protokol CoAP. Selain itu, akan dilakukan juga proses *semantic filtering* dan otentikasi serta orotisasi menggunakan JWT. *Cloud* yang digunakan saat ini merupakan sebuah VPS yang memiliki *public* IP serta akan melayani pengiriman data melalui *port* 5683 dan 80.

6.2 Pengujian Fungsional

Pengujian fungsional dilakukan berdasarkan skenario pengujian fungsional yang telah dideskripsikan pada bab perancangan.

6.2.1 API *Semantic IoT Web Service* dapat Menerima Data Berbentuk JSON Melalui CoAP

Pengujian fungsional yang pertama yaitu API *semantic IoT web service* dapat menerima data berbentuk JSON melalui protokol CoAP. Proses dan skenario dari pengujian tersebut dapat dilihat pada tabel 6.1.

Tabel 6.1 Proses pengujian menerima data JSON melalui CoAP

Kode	CoAP_001
Nama Fungsi	API *semantic IoT web service* dapat menerima data berbentuk JSON melalui protokol komunikasi CoAP.
Prosedur Pengujian	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud*. 2. Menjalankan kode textPost.py yang berisi data dalam bentuk JSON, JWT Token dan topik. Kode postText.py merupakan kode API CoAP di sisi *client.* 3. Menampilkan data yang dikirim melalui *log console* terminal *cloud.*
Hasil yang Diharapkan	API *semantic IoT web service* berhasil menerima data berbentuk JSON melalui protokol komunikasi CoAP

Hasil dari pengujian API *semantic IoT web service* dapat menerima data berbentuk JSON melalui CoAP dapat dilihat pada gambar 6.2.

```
(coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 0.0.0.0:5683
2019-04-07 06:51:03,974 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28489,
POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,974 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28489, POST-kM, [Uri-Path: login, ] {"token": "e6bd4133a...99 bytes
2019-04-07 06:51:03,989 - Thread-5    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 06:51:03,989 - Thread-5    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CREATED-kM, [] No payload
2019-04-07 06:51:03,989 - Thread-5    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28489,
CREATED-kM, [] No payload
2019-04-07 06:51:04,036 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28490,
GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,037 - MainThread - coapthon.layers.messagelayer -
```

Gambar 6.2 *Log* pengiriman data JSON

```
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28490, GET-None, [Uri-Path: login, ] No payload
2019-04-07 06:51:04,038 - Thread-7    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 06:51:04,038 - Thread-7    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28490,
CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 06:51:04,082 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28491,
POST-tY, [Uri-Path: postdata, ] {"topic": "jsontopic...495 bytes
2019-04-07 06:51:04,082 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28491, POST-tY, [Uri-Path: postdata, ] {"topic": "jsontopic...495 bytes
2019-04-07 06:51:04,111 - Thread-9    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 06:51:04,111 - Thread-9    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CREATED-tY, [] No payload
2019-04-07 06:51:04,112 - Thread-9    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28491,
CREATED-tY, [] No payload
2019-04-07 06:51:04,152 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 53253), To None, CON-28492,
GET-None, [Uri-Path: postdata, ] No payload
2019-04-07 06:51:04,152 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 53253), To None, CON-
28492, GET-None, [Uri-Path: postdata, ] No payload
2019-04-07 06:51:04,153 - Thread-11   - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 53253), None-
None, CONTENT-None, [] {"message": "Finishe...82 bytes
2019-04-07 06:51:04,153 - Thread-11   - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 53253), ACK-28492,
CONTENT-None, [] {"message": "Finishe...82 bytes
^CServer Shutdown
2019-04-07 06:51:12,344 - MainThread - coapthon.server.coap - INFO - Stop
server
```

Gambar 6.2 *Log* pengiriman data JSON (Lanjutan)

Gambar 6.2 yang di *highlight* berwarna kuning merupakan gambar pada *log* dari API CoAP *server* dalam menerima data berbentuk JSON lalu diteruskan kepada *semantic IoT web service*. Dalam pengujian ini, *semantic IoT web service* berhasil menerima data berbentuk JSON melalui CoAP.

6.2.2 API *Semantic IoT Web Service* dapat Menerima Data Berbentuk Gambar Melalui CoAP

Pengujian fungsional yang kedua yaitu API *semantic IoT web service* dapat menerima data berbentuk gambar melalui CoAP. Proses dan skenario dari pengujian tersebut dapat dilihat pada tabel 6.2.

Tabel 6.2 Proses pengujian menerima data gambar melalui CoAP

Kode	CoAP_002
Nama Fungsi	API *semantic IoT web service* dapat menerima data berbentuk gambar melalui protokol komunikasi CoAP
Prosedur Pengujian	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP server pada cloud

Tabel 6.2 Proses pengujian menerima data gambar melalui CoAP (Lanjutan)

	2. Menjalankan kode imagePost.py yang berisi data dalam bentuk gambar, JWT Token dan topik. Kode imagePost.py merupakan kode API CoAP di sisi client. 3. Menampilkan data yang dikirim melalui log console terminal cloud.
Hasil yang Diharapkan	API *semantic IoT web service* berhasil menerima data berbentuk gambar melalui protokol komunikasi CoAP

Hasil dari pengujian API *semantic IoT web service* dapat menerima data berbentuk gambar melalui CoAP dapat dilihat pada gambar 6.3.

```
(coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 0.0.0.0:5683
2019-04-07 07:09:09,492 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 28175), To None, CON-36322,
POST-tW, [Uri-Path: login, ] {"token": "7ed387a27...99 bytes
2019-04-07 07:09:09,492 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 28175), To None, CON-
36322, POST-tW, [Uri-Path: login, ] {"token": "7ed387a27...99 bytes
2019-04-07 07:09:09,509 - Thread-5    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 07:09:09,510 - Thread-5    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 28175), None-
None, CREATED-tW, [] No payload
2019-04-07 07:09:09,510 - Thread-5    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 28175), ACK-36322,
CREATED-tW, [] No payload
2019-04-07 07:09:09,551 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 28175), To None, CON-36323,
GET-None, [Uri-Path: login, ] No payload
2019-04-07 07:09:09,552 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 28175), To None, CON-
36323, GET-None, [Uri-Path: login, ] No payload
2019-04-07 07:09:09,553 - Thread-7    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 28175), None-
None, CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 07:09:09,553 - Thread-7    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 28175), ACK-36323,
CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 07:09:09,696 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 28175), To None, CON-36324,
POST-fz, [Uri-Path: postdata, Block1: 14, ] {"topic": "imagetopi...1024
bytes
2019-04-07 07:09:09,696 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 28175), To None, CON-
36324, POST-fz, [Uri-Path: postdata, Block1: 14, ] {"topic":
"imagetopi...1024 bytes
2019-04-07 07:09:09,696 - Thread-9    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 28175), None-
None, CONTINUE-fz, [Block1: 14, ] No payload
2019-04-07 07:09:09,697 - Thread-9    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 28175), ACK-36324,
CONTINUE-fz, [Block1: 14, ] No payload
```

Gambar 6.3 *Log* pengiriman data gambar

```
2019-04-07 07:09:09,735 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 28175), To None, CON-36325,
POST-fz, [Uri-Path: postdata, Block1: 30, ] \x00\\x00\\x00\\x00\...1024
bytes
2019-04-07 07:09:09,736 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 28175), To None, CON-
36325, POST-fz, [Uri-Path: postdata, Block1: 30, ]
\x00\\x00\\x00\\x00\...1024 bytes
2019-04-07 07:09:09,736 - Thread-11 - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 28175), None-
None, CONTINUE-fz, [Block1: 30, ] No payload
2019-04-07 07:09:09,736 - Thread-11 - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 28175), ACK-36325,
CONTINUE-fz, [Block1: 30, ] No payload
2019-04-07 07:09:09,780 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 28175), To None, CON-36326,
POST-fz, [Uri-Path: postdata, Block1: 46, ] x02\\x03\\x11\\x04\\...1024
bytes
2019-04-07 07:09:09,781 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 28175), To None, CON-
36326, POST-fz, [Uri-Path: postdata, Block1: 46, ]
x02\\x03\\x11\\x04\\...1024 bytes
```

Gambar 6.3 Log pengiriman data gambar (Lanjutan)

Gambar 6.3 yang di *highlight* berwarna kuning merupakan gambar pada *log* dari API CoAP *server* dalam menerima data berbentuk gambar lalu diteruskan kepada *semantic IoT web service*. Data gambar yang dikirim oleh CoAP *client* merupakan data berbentuk *pickle*. Dalam pengujian tersebut, *semantic IoT web service* berhasil menerima data berbentuk gambar melalui CoAP.

6.2.3 API *Semantic IoT Web Service* dapat Menerima Data Berbentuk Video Melalui CoAP

Pengujian fungsional yang ketiga yaitu API *semantic IoT web service* dapat menerima data berbentuk video melalui CoAP. Proses dan skenario dari pengujian tersebut dapat dilihat pada tabel 6.3.

Tabel 6.3 Proses pengujian menerima data video melalui CoAP

Kode	CoAP_003
Nama Fungsi	API *semantic IoT web service* dapat menerima data berbentuk video melalui protokol komunikasi CoAP
Prosedur Pengujian	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP server pada cloud. 2. Menjalankan kode videoPost.py yang berisi data dalam bentuk video, JWT Token dan topik. Kode videoPost.py merupakan kode API CoAP di sisi client. 3. Menampilkan data yang dikirim melalui log console terminal cloud.
Hasil yang Diharapkan	API *semantic IoT web service* berhasil menerima data berbentuk video melalui protokol komunikasi CoAP

Hasil dari pengujian API *semantic IoT web service* dapat menerima data berbentuk video melalui CoAP dapat dilihat pada gambar 6.4

```
((coap) iotapps@IoTApps:~/coap$ python webservice.py
server listening at 0.0.0.0:5683
2019-04-07 08:35:43,051 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 60304), To None, CON-29812,
POST-bl, [Uri-Path: login, ] {"token": "7ed387a27...99 bytes
2019-04-07 08:35:43,052 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 60304), To None, CON-
29812, POST-bl, [Uri-Path: login, ] {"token": "7ed387a27...99 bytes
2019-04-07 08:35:43,064 - Thread-5    - coapthon.server.coap - DEBUG -
Notify
2019-04-07 08:35:43,064 - Thread-5    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 60304), None-
None, CREATED-bl, [] No payload
2019-04-07 08:35:43,064 - Thread-5    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 60304), ACK-29812,
CREATED-bl, [] No payload
2019-04-07 08:35:43,132 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 60304), To None, CON-29813,
GET-None, [Uri-Path: login, ] No payload
2019-04-07 08:35:43,133 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 60304), To None, CON-
29813, GET-None, [Uri-Path: login, ] No payload
2019-04-07 08:35:43,134 - Thread-7    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 60304), None-
None, CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 08:35:43,135 - Thread-7    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 60304), ACK-29813,
CONTENT-None, [] {"message": "login s...448 bytes
2019-04-07 08:35:43,253 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 60304), To None, CON-29814,
POST-rl, [Uri-Path: postdata, Block1: 14, ] {"topic": "videotopi...1024
bytes
2019-04-07 08:35:43,253 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 60304), To None, CON-
29814, POST-rl, [Uri-Path: postdata, Block1: 14, ] {"topic":
"videotopi...1024 bytes
2019-04-07 08:35:43,254 - Thread-9    - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 60304), None-
None, CONTINUE-rl, [Block1: 14, ] No payload
2019-04-07 08:35:43,255 - Thread-9    - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 60304), ACK-29814,
CONTINUE-rl, [Block1: 14, ] No payload
2019-04-07 08:35:43,328 - MainThread - coapthon.server.coap - DEBUG -
receive_datagram - From ('202.80.212.194', 60304), To None, CON-29815,
POST-rl, [Uri-Path: postdata, Block1: 30, ] \\xb9\\x18\\xeb\\xd6...1024
bytes
2019-04-07 08:35:43,328 - MainThread - coapthon.layers.messagelayer -
DEBUG - receive_request - From ('202.80.212.194', 60304), To None, CON-
29815,    POST-rl,    [Uri-Path:    postdata,    Block1:    30,    ]
\\xb9\\x18\\xeb\\xd6...1024 bytes
2019-04-07 08:35:43,329 - Thread-11   - coapthon.layers.messagelayer -
DEBUG - send_response - From None, To ('202.80.212.194', 60304), None-
None, CONTINUE-rl, [Block1: 30, ] No payload
2019-04-07 08:35:43,329 - Thread-11   - coapthon.server.coap - DEBUG -
send_datagram - From None, To ('202.80.212.194', 60304), ACK-29815,
CONTINUE-rl, [Block1: 30, ] No payload
```

Gambar 6.4 *Log* pengiriman data video

Gambar 6.4 yang di *highlight* berwarna kuning merupakan gambar pada *log* dari API CoAP *server* dalam menerima data berbentuk video lalu diteruskan

kepada *semantic IoT web service*. Data video yang dikirim oleh *client* merupakan data berbentuk *pickle*. Dalam pengujian tersebut, *semantic IoT web service* berhasil menerima data berbentuk video melalui CoAP.

6.2.4 API *Semantic IoT Web Service* dapat Melakukan Proses *Semantic Filtering*

Pengujian fungsional yang keempat yaitu API *semantic IoT web service* yang telah ditambah CoAP dapat menentukan data *storage* melalui proses *semantic filtering*. Proses dan skenario dari pengujian tersebut dapat dilihat pada tabel 6.4.

Tabel 6.4 Proses pengujian *semantic filtering* pada API *semantic IoT web service*

Kode	CoAP_004
Nama Fungsi	API *semantic IoT web service* yang telah ditambah protokol komununikasi CoAP dapat melakukan proses *semantic filtering.*
Prosedur Pengujian	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP server pada cloud. 2. Melakukan pengiriman data menggunakan API CoAP client. 3. Menampilkan status proses semantic filtering dari data yang telah tersimpan pada data storage dan console terminal CoAP client.
Hasil yang Diharapkan	API *semantic IoT web service* yang telah ditambah protokol komununikasi CoAP berhasil melakukan proses *semantic filtering.*

Hasil dari pengujian API *semantic IoT web service* dapat menentukan data *storage* melalui metode *semantic filtering* dapat dilihat pada gambar dibawah ini.

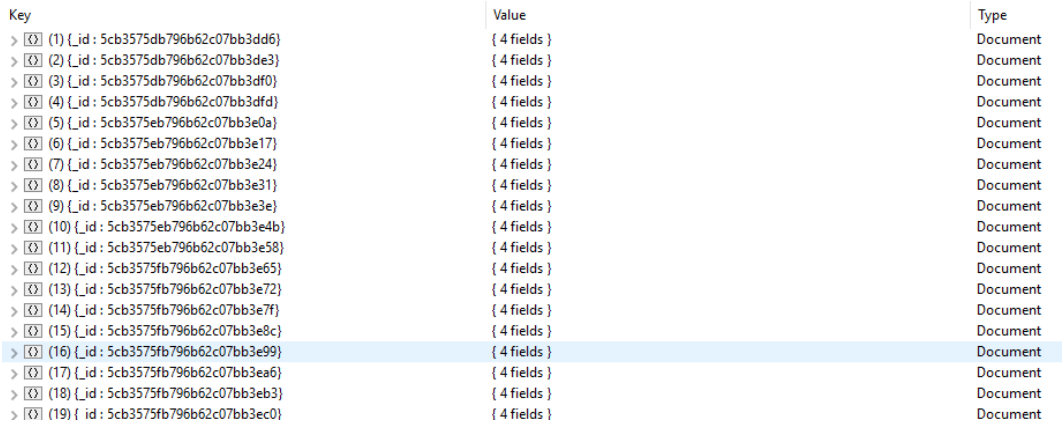
```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}

{"message": "Finished Saving", "format": "json", "storage": "mongodb",
"size": 66}
```

Gambar 6.5 *Console* CoAP *client*



Key	Value	Type
(1) {_id : 5cb3575db796b62c07bb3dd6}	{ 4 fields }	Document
(2) {_id : 5cb3575db796b62c07bb3de3}	{ 4 fields }	Document
(3) {_id : 5cb3575db796b62c07bb3df0}	{ 4 fields }	Document
(4) {_id : 5cb3575db796b62c07bb3dfd}	{ 4 fields }	Document
(5) {_id : 5cb3575eb796b62c07bb3e0a}	{ 4 fields }	Document
(6) {_id : 5cb3575eb796b62c07bb3e17}	{ 4 fields }	Document
(7) {_id : 5cb3575eb796b62c07bb3e24}	{ 4 fields }	Document
(8) {_id : 5cb3575eb796b62c07bb3e31}	{ 4 fields }	Document
(9) {_id : 5cb3575eb796b62c07bb3e3e}	{ 4 fields }	Document
(10) {_id : 5cb3575eb796b62c07bb3e4b}	{ 4 fields }	Document
(11) {_id : 5cb3575eb796b62c07bb3e58}	{ 4 fields }	Document
(12) {_id : 5cb3575fb796b62c07bb3e65}	{ 4 fields }	Document
(13) {_id : 5cb3575fb796b62c07bb3e72}	{ 4 fields }	Document
(14) {_id : 5cb3575fb796b62c07bb3e7f}	{ 4 fields }	Document
(15) {_id : 5cb3575fb796b62c07bb3e8c}	{ 4 fields }	Document
(16) {_id : 5cb3575fb796b62c07bb3e99}	{ 4 fields }	Document
(17) {_id : 5cb3575fb796b62c07bb3ea6}	{ 4 fields }	Document
(18) {_id : 5cb3575fb796b62c07bb3eb3}	{ 4 fields }	Document
(19) {_id : 5cb3575fb796b62c07bb3ec0}	{ 4 fields }	Document

Gambar 6.6 Data *storage* MongoDB

Gambar 6.5 dan 6.6 merupakan proses penyimpanan data berbentuk JSON yang disimpan pada mongoDB. Gambar 6.5 merupakan gambar *console* CoAP *client*. Khusus yang di *highlight* berwarna kuning menampilkan respon pada CoAP *server* yang telah melakukan proses *semantic filtering* dengan menyimpan data tersebut pada mongoDB. Gambar 6.6 menampilkan data JSON telah tersimpan pada data *storage* MongoDB sesuai dengan proses *semantic filtering*. Dalam pengujian ini, API *semantic IoT web service* yang telah ditambah CoAP berhasil melakukan identifikasi dan penyimpanan terhadap data JSON yang diterima menggunakan metode *semantic filtering.*

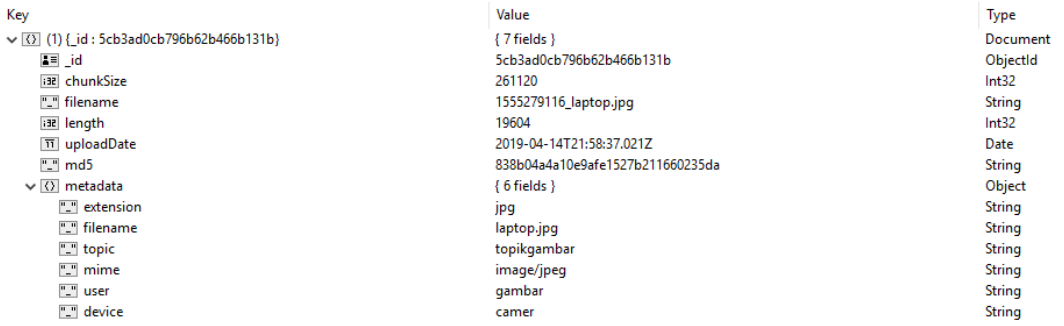
Adapun proses *semantic filtering* terhadap data berbentuk gambar dapat dilihat pada gambar 6.7 dan 6.8.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python imagePost.py
d459be6a681f8192388ef2adb63b944e1a455129
17f9d337be930bf9ed07cb18c4f19913 topikgambar

{"message": "Create JWT Token, login successfully", "jwtToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1Y2IzYWM3NGFkYjA2NDA0Y
2E1YTg2MzYiLCJpc3MiOiJodHRwOi8vYXBpLmlvdGFwcHMuYmVsYWphcmRpc2luaS5jb20v
dXNlci9sb2dpbiIsImp0aSI6IjdiN2EyZjFiNTlkMjQyMmFiZTAxZGFmYTRlMjgxZmVmIiw
iZGV2aWNlIjoiNWNiM2FjYjBhZGIwNjQwNjQ0NmE3ZDc2IiwiaWF0IjoxNTU1Mjc5MTExLC
JuYmYiOjE1NTUyNzkxMTF9.gsF7oxfzZ1yzQkfUscO4pCrNIcOz26uCSEHSLFaNVQc"}

{"storage": "gridfs", "message": "Finished Saving", "format": "file",
"mime": "image/jpeg", "size": 19604}
```

Gambar 6.7 *Console* CoAP *client*



Key	Value	Type
(1) {_id : 5cb3ad0cb796b62b466b131b}	{ 7 fields }	Document
_id	5cb3ad0cb796b62b466b131b	ObjectId
chunkSize	261120	Int32
filename	1555279116_laptop.jpg	String
length	19604	Int32
uploadDate	2019-04-14T21:58:37.021Z	Date
md5	838b04a4a10e9afe1527b211660235da	String
metadata	{ 6 fields }	Object
extension	jpg	String
filename	laptop.jpg	String
topic	topikgambar	String
mime	image/jpeg	String
user	gambar	String
device	camer	String

Gambar 6.8 Data *storage* GridFS

Gambar 6.7 dan 6.8 merupakan proses penyimpanan data berbentuk gambar yang disimpan pada GridFS. Gambar 6.7 merupakan *console* dari CoAP *client*. Khusus yang di *highlight* berwarna kuning menampilkan respon dari CoAP *server* yang telah melakukan proses *semantic filtering* dengan menyimpan data gambar tersebut pada GridFS. Gambar 6.8 menampilkan data gambar telah tersimpan pada data *storage* GridFS sesuai dengan proses *semantic filtering*. Dalam pengujian ini, API *semantic IoT web service* yang telah ditambah CoAP berhasil melakukan identifikasi dan penyimpanan terhadap data gambar yang diterima menggunakan metode *semantic filtering.*

Adapun hasil proses *semantic filtering* terhadap data dalam bentuk video dapat dilihat pada gambar 6.9 dan 6.10.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python videoPost.py
d459be6a681f8192388ef2adb63b944e1a455129
17f9d337be930bf9ed07cb18c4f19913 topikvideo

{"message": "login successfully", "jwtToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1Y2IzYWM3NGFkYjA2NDA0Y
2E1YTg2MzYiLCJpc3MiOiJodHRwOi8vYXBpLmlvdGFwcHMuYmVsYWphcmRpc2luaS5jb20v
dXNlci9sb2dpbiIsImp0aSI6IjdiN2EyZjFiNTlkMjQyMmFiZTAxZGFmYTRlMjgxZmVmIiw
iZGV2aWNlIjoiNWNiM2FjYjBhZGIwNjQwNjQ0NmE3ZDc2IiwiaWF0IjoxNTU1Mjc5MTExLC
JuYmYiOjE1NTUyNzkxMTF9.gsF7oxfzZ1yzQkfUscO4pCrNIcOz26uCSEHSLFaNVQc"}

{"storage": "gridfs", "message": "Finished Saving", "format": "file",
"mime": "application/octet-stream", "size": 178743}
```

Gambar 6.9 *Console* CoAP *client*

(1) {_id : 5cb3ae39b796b62c32525968}	{ 7 fields }	Document
_id	5cb3ae39b796b62c32525968	ObjectId
chunkSize	261120	Int32
filename	1555279417_video.MP4	String
length	178743	Int32
uploadDate	2019-04-14T22:03:37.997Z	Date
md5	1b92191f1a9c3b4abb825c23fef6d96b	String
metadata	{ 6 fields }	Object

Gambar 6.10 Data *storage* GridFS

Gambar 6.9 dan 6.10 merupakan proses penyimpanan data berbentuk video yang disimpan pada GridFS. Gambar 6.9 merupakan *console* CoAP *client*. Khusus yang di *highlight* berwarna kuning menampilkan respon pada CoAP *server* yang telah melakukan proses *semantic filtering* dengan menyimpan data video tersebut pada GridFS. Gambar 6.10 menampilkan data video telah tersimpan pada data *storage* GridFS sesuai dengan proses *semantic filtering*. Dalam pengujian ini, API *semantic IoT web service* yang telah ditambah CoAP berhasil melakukan identifikasi dan penyimpanan terhadap data video yang diterima menggunakan metode *semantic filtering.*

6.2.5 API *Semantic IoT Web Service* dapat Otentikasi dan Otorisasi Menggunakan JWT

Pengujian fungsional yang kelima yaitu API *semantic IoT web service* yang telah ditambah CoAP dapat melakukan otentikasi dan otorisasi menggunakan

JSON Web Token (JWT). Proses dan skenario dari pengujian tersebut dapat dilihat pada tabel 6.5.

Tabel 6.5 Proses pengujian otentikasi dan otorisasi menggunakan JWT

Kode	CoAP_005
Nama Fungsi	API *semantic IoT web service* yang telah ditambah protokol komunikasi CoAP dapat melakukan otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT).
Prosedur Pengujian	1. Menjalankan kode webservice.py yang merupakan kode dari API CoAP *server* pada *cloud* 2. *Client* melakukan pengiriman *token key* dan *secret key* yang benar untuk proses otentikasi 3. *Client* melakukan pengiriman *token key* dan *secret key* yang salah untuk proses otentikasi 4. Menampilkan respon pada *console* terminal CoAP *client* 5. *Client* melakukan pengiriman JWT token yang benar untuk proses otorisasi 6. *Client* melakukan pengiriman JWT token yang salah untuk proses otorisasi
Hasil yang Diharapkan	API *semantic IoT web service* yang telah ditambah protokol komunikasi CoAP berhasil melakukan otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT).

Hasil dari pengujian *semantic IoT web service* yang telah ditambah CoAP dapat melakukan otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT) dapat dilihat pada gambar dibawah.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}
```

Gambar 6.11 Proses otentikasi berhasil

Gambar 6.11 merupakan hasil dari proses otentikasi pada API CoAP *server* kelas Login. Dalam proses tersebut, CoAP *client* berhasil melakukan proses otentikasi dengan didapatkannya JWT token.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee5450 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"error": "Your token or secret key is incorrect"}
```

Gambar 6.12 Proses otentikasi gagal

Gambar 6.12 merupakan hasil dari proses otentikasi pada API CoAP *server* kelas Login. Dalam proses tersebut, otentikasi gagal dilakukan karena *token key* atau *secret key* yang dikirimkan salah, sehingga CoAP *client* akan menerima pesan *error*.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}

{"message": "Finished Saving", "format": "json", "storage": "mongodb",
"size": 66}
```

Gambar 6.13 Proses otorisasi berhasil

Gambar 6.13 merupakan gambar respon dari API CoAP *server* kelas postData dimana proses otorisasi berhasil dilakukan, hal tersebut dibuktikan dengan data yang dikirim oleh CoAP *client* telah berhasil untuk disimpan pada data *storage* yang disertai dengan pengiriman status penyimpanan.

```
rezananda@REZANANDA:/mnt/c/Users/Reza Nanda/OneDrive/Documents/web
service/RestCoAP$ python textPost.py
f01db494c0ba5b7736f1e12693b2fc91b6b6d8be
e05b41e6dd53f462a2bf0b79eee54502 coaphttp50

Attention! if your topic has spaces, please add "_" (uderscore)

{"message": "login successfully", "jwtToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkuaW90Y
XBwcy5iZWxhamFyZGlzaW5pLmNvbVwvdXNlclwvbG9naW4iLCJpYXQiOjE1NTUyNjIzMDcs
Im5iZiI6MTU1NTI2MjMwNywianRpIjoiVFJEdXNJZDdHM2tETXVBUSIsInN1YiI6IjVjYjM
0NDRlYWRiMDY0N2ZjYjA1YWUzMyIsInBydiI6Ijg3ZTBhZjFlZjlmZDE1ODEyZmRlYzk3MT
UzYTE0ZTBiMDQ3NTQ2YWEiLCJkZXZpY2UiOiI1Y2IzNDRkYmFkYjA2NDAyZWQzMjQ2YTIif
Q.urt_gdEE0_vOn2lRB68KKAi0QGk1S41zxaEPG8Yt2jI"}

{"error": "Not Authorized to write on this topic!"}
```

Gambar 6.14 Proses otorisasi gagal

Gambar 6.14 merupakan gambar respon dari API CoAP *server* kelas postData yang terjadi kegagalan dalam proses otorisasi yang dibuktikan dengan terkirimnya pesan *error* kepada CoAP *client*. Hasil pengujian otentikasi dan otorisasi yang telah dilakukan menunjukkan *semantic IoT web service* yang telah

ditambah CoAP berhasil melakukan mekanisme otentikasi dan otorisasi menggunakan *JSON Web Token* (JWT)

Dari keseluruhan proses pengujian fungsional yang telah dilakukan menunjukkan seluruh kebutuhan dari *semantic IoT web service* yang telah ditambah CoAP dapat berjalan sesuai dengan fungsinya.

6.3 Pengujian Interoperabilitas

Pengujian interoperabilitas kali ini akan berfokus pada pengiriman data menggunakan protokol komunikasi CoAP dan HTTP secara bersama-sama, serta secara independen melalui masing-masing protokol komunikasi. Pengujian pertama yaitu pengujian pengiriman data (PI_001). Pengujian ini juga akan bersamaan dengan pengujian kedua yaitu *monotoring* pengiriman data menggunakan tcpdump (PI_002). Hasil *capture packet* tersebut kemudian akan dianalisis melalui aplikasi wireshark. Hasil pengujian pengiriman data menggunakan CoAP dan HTTP secara independen dapat dilihat pada tabel 6.11, 6.12, 6.13, 6.14. 6.15 dan 6.16. Sedangkan hasil pengujian pengiriman data secara bersama-sama melalui CoAP dan HTTP dapat dilihat pada tabel 6.17, 6.18 dan 6.19.

Tabel 6.11 dan 6.12 merupakan hasil *capture packet* pengiriman data berbentuk JSON secara independen melalui CoAP dan HTTP. Data JSON yang dikirim masing-masing sejumlah tiga data. Pada *capture packet* tabel 6.11 melalui CoAP terdapat empat *packet* yang didapatkan yaitu packet proses otentikasi dan *packet* proses pengiriman data berbentuk JSON. Data yang dikirim melalui CoAP lalu diterima pada sisi *server* dengan menggunakan tipe pesan *Confirmable* (CON). *Server* selanjutnya akan memberikan balasan dengan mengirimkan respon status penyimpanan serta dengan menyertakan tipe pesan *Acknowledgment* (ACK). Sedangkan pada *capture packet* tabel 6.12 melalui HTTP terdapat empat *packet* yang didapatkan yaitu satu *packet* proses otentikasi dan tiga *packet* proses pengiriman data berbentuk JSON.

Tabel 6.13 merupakan sebagian dari hasil *capture packet* pengiriman data berbentuk gambar melalui CoAP dan tabel 6.14 merupakan keseluruhan *capture packet* pengiriman data gambar pada HTTP. Pada tabel 6.13 terdapat banyak *packet* yang didapatkan. Banyaknya *packet* yang didapatkan tersebut merupakan hasil dari mekanisme *block-wise transfer* dari CoAP yang akan memecah data menjadi beberapa *block* per *byte*. Data yang dikirim melalui CoAP lalu diterima pada sisi *server* dengan menggunakan tipe pesan *Confirmable* (CON). *Server* selanjutnya akan memberikan balasan dengan mengirimkan respon status penyimpanan serta dengan menyertakan tipe pesan *Acknowledgment* (ACK). Sedangkan pada tabel 6.14 terdapat dua *packet* yang didapatkan masing-masing packet proses otentikasi dan packet proses pengiriman data berbentuk gambar.

Tabel 6.15 merupakan sebagian dari hasil *capture packet* pengiriman data berbentuk video melalui CoAP dan tabel 6.16 merupakan keseluruhan *capture packet* pengiriman data video pada HTTP. Pada tabel 6.15 terdapat banyak *packet*

yang didapatkan. Banyaknya *packet* yang didapatkan tersebut merupakan hasil dari mekanisme *block-wise transfer* dari CoAP yang akan memecah data menjadi beberapa *block* per *byte*. Data yang dikirim melalui CoAP lalu diterima pada sisi *server* dengan menggunakan tipe pesan *Confirmable* (CON). *Server* selanjutnya akan memberikan balasan dengan mengirimkan respon status penyimpanan serta dengan menyertakan tipe pesan *Acknowledgment* (ACK). Sedangkan pada tabel 6.16 terdapat dua *packet* yang didapatkan masing-masing packet proses otentikasi dan packet proses pengiriman data berbentuk video.

Tabel 6.17 merupakan *capture packet* pengiriman data berbentuk JSON secara bersama-sama melalui CoAP dan HTTP. *Packet* dengan *highlight* berwarna kuning khusus untuk *packet* pengiriman data melalui HTTP, sedangkan yang tidak di *highlight* merupakan packet pengiriman data melalui CoAP. Data JSON yang dikirimkan melalui masing-masing protokol adalah sebanyak 3 data. Pada hasil *capture packet*, terdapat 12 *packet* yang terdiri dari *packet* untuk proses otentikasi, *packet* untuk pengiriman data serta *packet* respon kepada *client*. Dari hasil *packet capture* yang dilakukan, seluruh proses pengiriman serta penerimaan data JSON dilakukan pada satu waktu yang sama. Hal tersebut menandakan pengiriman data JSON melalui HTTP dan CoAP secara bersama-sama berhasil dilakukan.

Tabel 6.18 merupakan *capture packet* pengiriman data berbentuk gambar secara bersama-sama melalui CoAP dan HTTP. Khusus untuk pengiriman data gambar melalui CoAP, hanya diambil beberapa *packet* sebagai *sample*. *Packet* dengan *highlight* berwarna kuning khusus untuk *packet* pengiriman data melalui HTTP, sedangkan yang tidak di *highlight* merupakan packet pengiriman data melalui CoAP. Pada hasil *capture packet* terdapat dua *packet* HTTP yang merupakan proses otentikasi dan pengiriman data gambar. Sedangkan pada pengiriman data gambar melalui CoAP terdapat banyak *packet*. Banyaknya *packet* yang tertangkap tersebut merupakan pecahan data gambar yang telah dirubah menjadi bentuk *pickle* yang kemudian dikirim menjadi beberapa *block* per *bytes*. Mekanisme tersebut dalam CoAP dinamakan *block-wise transfer*. Dari hasil *packet capture* yang dilakukan, proses pengiriman serta penerimaan data berbentuk gambar dilakukan pada satu waktu yang sama. Hal tersebut menandakan pengiriman data berbentuk gambar melalui HTTP dan CoAP secara bersama-sama berhasil dilakukan.

Tabel 6.17 merupakan *capture packet* pengiriman data berbentuk video secara bersama-sama melalui CoAP dan HTTP. Dalam proses tersebut terdapat *packet* yang dikirimkan melalui masing-masing protokol. Khusus pada *packet* pengiriman data melalui CoAP hanya diambil sebagaian sebagai *sample*. *Packet* dengan *highlight* berwarna kuning khusus untuk *packet* pengiriman data melalui HTTP, sedangkan yang tidak di *highlight* merupakan packet pengiriman data melalui CoAP. Pada pengiriman data video melalui HTTP terdapat dua packet yaitu *packet* untuk proses otentikasi dan *packet* pengiriman data. Sedangkan pada *packet* pengiriman data video melalui CoAP, terdapat banyak *packet*. Banyaknya *packet* yang tertangkap tersebut merupakan pecahan data video yang telah

dirubah menjadi bentuk *pickle* yang kemudian dikirim menjadi beberapa *block* per *bytes*. Mekanisme tersebut dalam CoAP dinamakan *block-wise transfer*. Pada hasil *capture packet*, pengiriman serta penerimaan data berbentuk video melalui masing-masing protokol dilakukan dalam satu waktu yang sama. Hal tersebut menandakan bahwa pengiriman data berbentuk video menggunakan CoAP dan HTTP secara bersama sama berhasil dilakukan.

Pengujian selanjutnya yaitu (PI_003) dan (PI_004) akan berfokus pada integritas dari data yang dikirimkan melalui CoAP dan HTTP..

6.3.1 Integritas Pengiriman Data JSON

Contoh dari data yang akan dikirimkan oleh CoAP *client* dan HTTP *client* kepada *semantic IoT web service* masing-masing dapat dilihat pada gambar 6.15 dan 6.16.

```
'data' : {
    'humidity' : 80,
    'nitrogen' : 75,
    'ph' : 5,
    'protocol': 'CoAP',
    }
```

Gambar 6.15 Data JSON CoAP

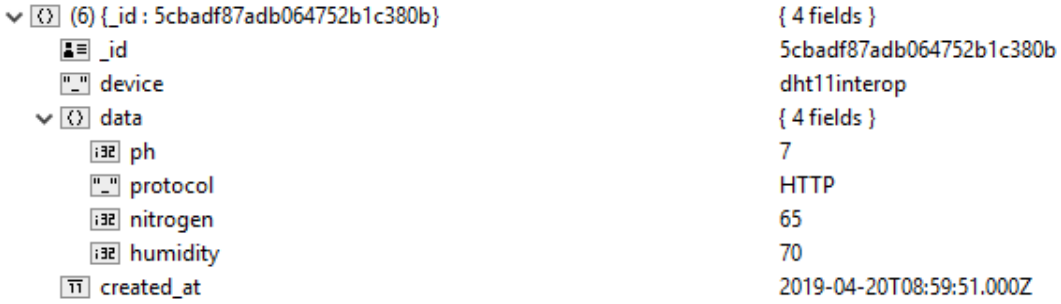
```
iot = {
        'humidity':70,
        'nitrogen':65,
        'ph':7,
        'protocol': 'HTTP',
    }
```

Gambar 6.16 Data JSON HTTP

Gambar 6.15 dan 6.16 merupakan contoh data JSON yang dikirim oleh *client* kepada *semantic IoT web service*. Data tersebut berupa *humidity*, nitrogen dan ph yang berbentuk angka. Seluruh data tersebut kemudian dikirim kepada *semantic IoT web service* untuk disimpan pada data *storage*. Gambar 6.17 dan 6.18 merupakan data dari CoAP *client* dan HTTP *client* yang berhasil disimpan ke dalam data *storage*.

(1) {_id : 5cbadf7fb796b67ec64f1d8d}	{ 4 fields }
_id	5cbadf7fb796b67ec64f1d8d
device	dht11interop
created_at	2019-04-20T08:59:43.527Z
data	{ 4 fields }
ph	5
protocol	CoAP
nitrogen	75
humidity	80

Gambar 6.17 Data JSON pada data *storage* (CoAP)



Gambar 6.18 Data JSON pada data *storage* (HTTP)

Data berbentuk JSON sesuai dengan hasil proses *semantic filtering* kemudian disimpan pada mongoDB. Data yang telah tersimpan dalam data *storage* dapat ditampilkan melalui IoT *applications*. Adapun hasil dari data yang ditampilkan dalam IoT *applications* dapat dilihat pada gambar 6.19 dan 6.20.

Ph	Protocol	Nitrogen	Humidity	Device	Created At
5	CoAP	75	80	dht11interop	Sat 20-Apr-2019 15:59:43

Gambar 6.19 Data JSON pada IoT application (CoAP)

Ph	Protocol	Nitrogen	Humidity	Device	Created At
7	HTTP	65	70	dht11interop	Sat 20-Apr-2019 15:59:44

Gambar 6.20 Data JSON pada IoT application (HTTP)

Data pada gambar 6.19 dan 6.20 masing-masing merupakan data dari CoAP *client* dan HTTP *client* yang berhasil disimpan dalam data *storage* serta dapat ditampilkan dalam IoT *application*. Kedua data yang ditampilkan tersebut memiliki nilai yang sama dengan data yang dikirimkan oleh *client* serta tersimpan dalam data *storage*. Hal tersebut menandakan bahwa data JSON yang dikirim kepada *semantic IoT web service* hingga ditampilkan pada IoT *applications* melalui masing-masing protokol memiliki integritas yang sama.

6.3.2 Integritas Pengiriman data Gambar

Pada pengiriman data berbentuk gambar, dilakukan perubahan bentuk dari data gambar yang akan dikirim. Khusus pada pengiriman melalui protokol CoAP, data gambar yang dikirim harus dirubah menjadi data dalam bentuk *pickle*. Sedangkan pada pengiriman melalui protokol HTTP, data gambar yang dikirim berbentuk *raw*. Gambar 6.21 merupakan data gambar yang akan dikirim kepada *semantic IoT web service* melalui masing-masing protokol.

Gambar 6.21 Data gambar

Data gambar tersebut kemudian dikirim melalui CoAP dan HTTP kepada *semantic IoT web service*. Data gambar kemudian diterima pada *semantic IoT web service* untuk masuk ke dalam proses *semantic filtering* dan disimpan ke dalam data *storage*. Sesuai dengan proses *semantic filtering*, data berbentuk *file* yang dikenali oleh sistem disimpan pada GridFS. Gambar 6.22 dan 6.23 menunjukkan data gambar yang telah disimpan pada data *storage* GridFS.

(2) {_id : 5cbae660b796b60261fde202}	{ 7 fields }
_id	5cbae660b796b60261fde202
chunkSize	261120
filename	1555752544_laptop.jpg
length	19604
uploadDate	2019-04-20T09:29:04.998Z
md5	838b04a4a10e9afe1527b211660235da
metadata	{ 6 fields }
extension	jpg
filename	laptop.jpg
topic	gambarinterop
mime	image/jpeg
user	interoperabilitas
device	kamerainterop

Gambar 6.22 Data gambar pada data *stroage* (CoAP)

(1) {_id : 5cbae65eadb064763a08bacb}	{ 7 fields }
_id	5cbae65eadb064763a08bacb
chunkSize	261120
filename	1555752542_laptop.jpg
uploadDate	2019-04-20T09:29:02.436Z
metadata	{ 6 fields }
filename	laptop.jpg
mime	image/jpeg
extension	jpeg
user	interoperabilitas
device	kamerainterop
topic	gambarinterop
length	19604
md5	838b04a4a10e9afe1527b211660235da

Gambar 6.23 Data gambar pada data *stroage* (HTTP)

Data gambar yang tersimpan pada data *storage* merupakan data berbentuk *binary*. Data yang telah tersimpan dalam data *storage* dapat ditampilkan melalui IoT *application*. Gambar 6.24 dan 6.25 menunjukkan hasil tampilan IoT *application* yang memuat data gambar yang telah dikirim melalui protokol CoAP dan HTTP.


```
{
  "_id": {
    "$oid": "5cbae65eadb064763a08bacb"
  },
```


Gambar 6.24 Data gambar pada *IoT application* (CoAP)

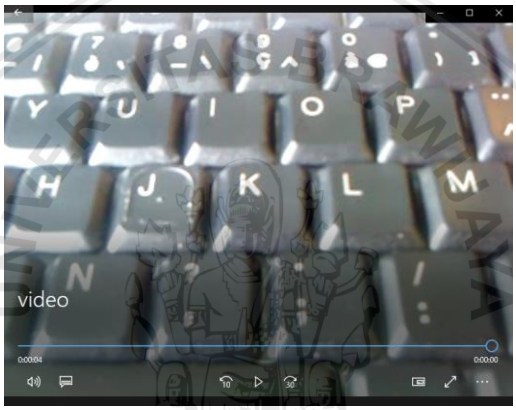

```
{
  "_id": {
    "$oid": "5cbae660b796b60261fde202"
  },
```


Gambar 6.25 Data gambar pada *IoT application* (HTTP)

Data gambar yang ditampilkan pada gambar 6.24 dan 6.25 masing-masing merupakan data dari CoAP *client* dan HTTP *client* yang berhasil disimpan dalam data *storage* serta dapat ditampilkan dalam IoT *application*. Kedua data yang ditampilkan tersebut memiliki gambar atau hasil yang sama dengan data gambar yang dikirimkan. Hal tersebut menandakan bahwa data berbentuk gambar yang dikirim kepada *semantic IoT web service* hingga ditampilkan pada IoT *applications* melalui masing-masing protokol memiliki integritas yang sama.

6.3.3 Integritas Pengiriman Data Video

Sama halnya dengan pengiriman data berbentuk gambar, pada pengiriman data berbentuk video, data harus dirubah terlebih dahulu menjadi *pickle* untuk CoAP, sedangkan untuk HTTP hanya dirubah menjadi bentuk *raw*. Gambar 6.26 menunjukkan data video yang akan dikirim kepada *semantic IoT web service* melalui masing-masing prtokol.



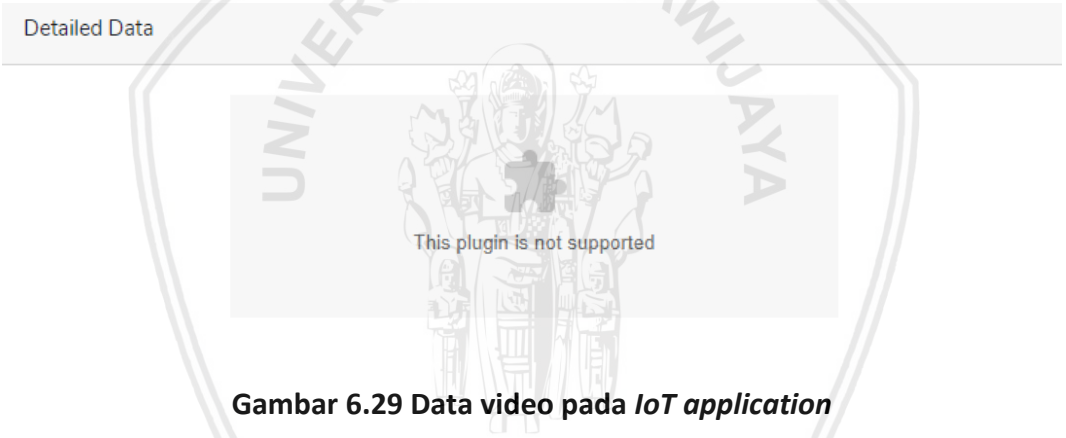
Gambar 6.26 Data video

Data tersebut kemudian dikirim kepada *semantic IoT web service*. Sesuai dengan proses *semantic filtering*, data yang dikenali oleh sistem akan disimpan ke dalam data *storage* GridFS. Adapun data video yang telah disimpan pada GridFS dapat dilihat pada gambar 6.27 dan 6.28.

Gambar 6.27 Data video pada data *stroage* (CoAP)

Gambar 6.28 Data video pada data *stroage* (HTTP)

Data yang tersimpan dalam data *storage* merupakan data berbentuk *binary*. Namun data video yang telah tersimpan dalam data *storage* tidak dapat ditampilkan ke dalam IoT *application* karena keterbatasan *plugin*. IoT *application* hanya menampilkan data video seperti gambar 6.29.



Gambar 6.29 Data video pada *IoT application*

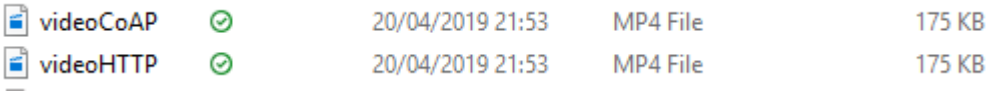
Untuk membuktikan integritas dari data video, data video yang telah tersimpan dalam data *storage* kemudian akan ditulis kembali ke dalam sebuah *file*. Adapun *source code* dalam mengambil serta menulis data video pada data storage dapat dilihat pada tabel 6.6.

Tabel 6.6 *Source code write* data video

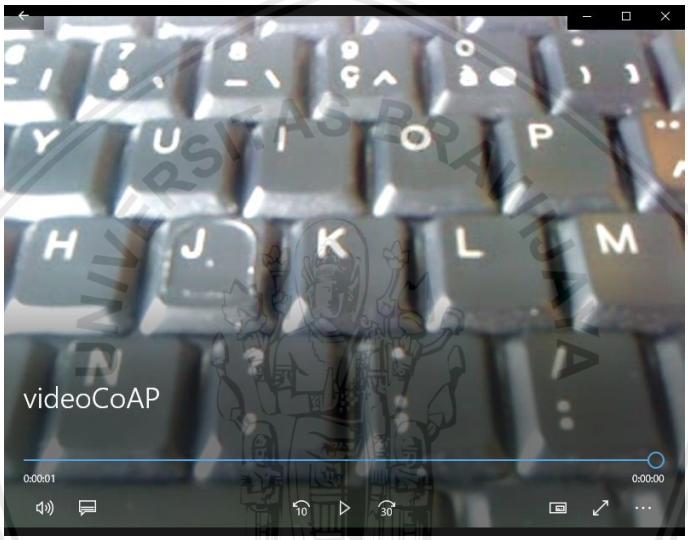
No	WriteVideo.py
1	`import pymongo`
2	`import gridfs`
3	
4	`from pymongo import MongoClient`
5	`from bson.objectid import ObjectId`
6	
7	`client = MongoClient('mongodb://localhost:27017/')`
8	`db = client.gambar`
9	`topic = db['topikvideo'] //topik yang dipilih`
10	`a = topic.chunks.find({"_id" :ObjectId("5cb3b8e9adb06404cb33b819")})`
11	`//_id merupaka ide dari chunks Data video`
12	
13	`data = open("pidio.MP4", 'wb') //proses writew data`
14	`b = []`
15	`for i in a:`

```
16      b.append(i)
data.write(b[0]['data'])
```

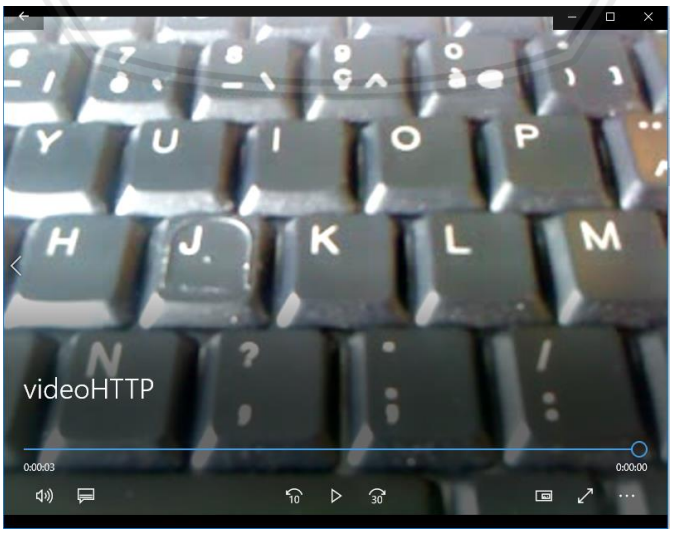
Source code tersebut dijalankan pada *cloud* agar langsung dapat langsung mengambil data video yang telah tersimpan dalam data *storage*. Hasil dari data video yang telah berhasil ditulis dapat dilihat pada gambar 6.30. Hasil dari data video tersebut dapat diputar seperti pada gambar 6.31 dan 6.32.



Gambar 6.30 Data video yang berhasil di tulis



Gambar 6.31 Data video CoAP berhasil diputar



Gambar 6.32 Data video HTTP berhasil diputar

Data video yang ditampilkan pada gambar 6.31 dan 6.32 merupakan data video yang telah tersimpan dalam GridFS dan ditulis kembali ke dalam *file*. Data tersebut juga dapat diputar atau dimainkan sama seperti data yang dikirimkan sebelumnya. Hal tersebut menandakan bahwa data berbentuk video yang dikirim kepada *semantic IoT web service* melalui masing-masing protokol hingga ditulis kembali dalam sebuah *file* memiliki integritas yang sama.

Seluruh proses pengujian interoperabilitas berhasil dilakukan. Data dapat dikirim secara independen serta secara bersama-sama melalui CoAP dan HTTP kepada *semantic IoT web service*. Hal tersebut dibuktikan dengan hasil *capture packet* yang ditampilkan wireshark. Dari sisi integritas data, seluruh data yang dikirim oleh *client* melalui masing-masing protokol kemudian ditampilkan kembali pada IoT *application* memiliki integritas data yang sama. Hal tersebut menandakan bahwa *semantic IoT web service* yang saat ini telah ditambah protokol CoAP mampu mengatasi permasalahan interoperabilitas, khususnya *syntactical interoperability*.

6.4 Pengujian Kinerja

Pengujian kinerja mencakup parameter penggunaan memori, penggunaan CPU, *runtime* dan *throughput*. Data yang dikirim oleh *client* melalui masing-masing protokol adalah sebesar 50, 100, 150, 200, dan 250 data. Adapun skenario pengujian kinerja telah dijelaskan dalam sub bab perancangan pengujian kinerja. Data yang digunakan dalam proses pengujian ini adalah data berbentuk JSON. Kedua protokol akan bersama-sama melakukan pengiriman data kepada *semantic IoT web service* untuk kemudian dicari Q1, median, Q2, nilai maximum dan nilai minimum. Data tersebut kemudian akan dirubah kedalam bentuk *whisker plot* lalu dianalisis.

6.4.1 Penggunaan Memori

Penggunaan memori dari *cloud* dalam melakukan pengiriman data melalui protokol CoAP dan HTTP secara bersamaan kepada *semantic IoT web service* dapat dilihat pada tabel 6.7.

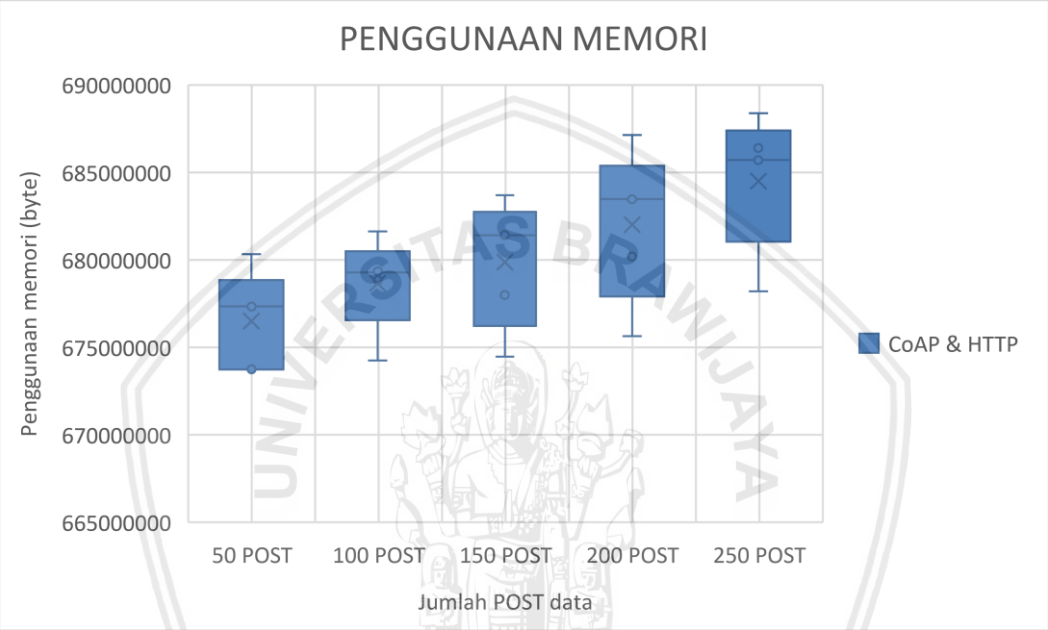
Tabel 6.7 Hasil pengujian penggunaan memori

	Q1	MEDIAN	Q3	MAX	MIN
50 POST	673743872	677339136	677371904	680329216	673742848
100 POST	678841344	679276544	679350272	681623552	674258944
150 POST	677990400	681410560	681807872	683692032	674463744
200 POST	680194048	683474944	683630592	687140864	675635200
250 POST	683871232	685701120	686390272	688377856	678203392

Dalam tabel tersebut terdapar komponen berupa Q1, median, Q3, nilai terkecil dan nilai terbesar dari masing-masing skenario pengujian. Data dalam tabel tersebut kemudian dirubah menjadi bentuk *whisker plot* untuk

mempermudah proses analisis. Adapun *whisker plot* dari penggunaan memori ditunjukkan pada gambar 6.35.

Gambar 6.33 merupakan *whisker plot* penggunaan memori dalam proses pengiriman data melalui CoAP dan HTTP secara bersama-sama kepada *semantic IoT web service*. Pada *whisker plot* tersebut, semakin tinggi data yang dikirim, berbanding lurus dengan penggunaan memori dari *cloud*. Penggunaan memori tertinggi pada pengiriman data sejumlah 250 yaitu 685701120 bytes. Sedangkan penggunaan memori terendah pada pengiriman data sejumlah 50 yaitu 677339136 bytes.



Gambar 6.33 Hasil penggunaan memori

6.4.2 Penggunaan CPU

Penggunaan CPU dari *cloud* dalam melakukan pengiriman data melalui protokol CoAP dan HTTP secara bersamaan kepada *semantic IoT web service* dapat dilihat pada tabel 6.8.

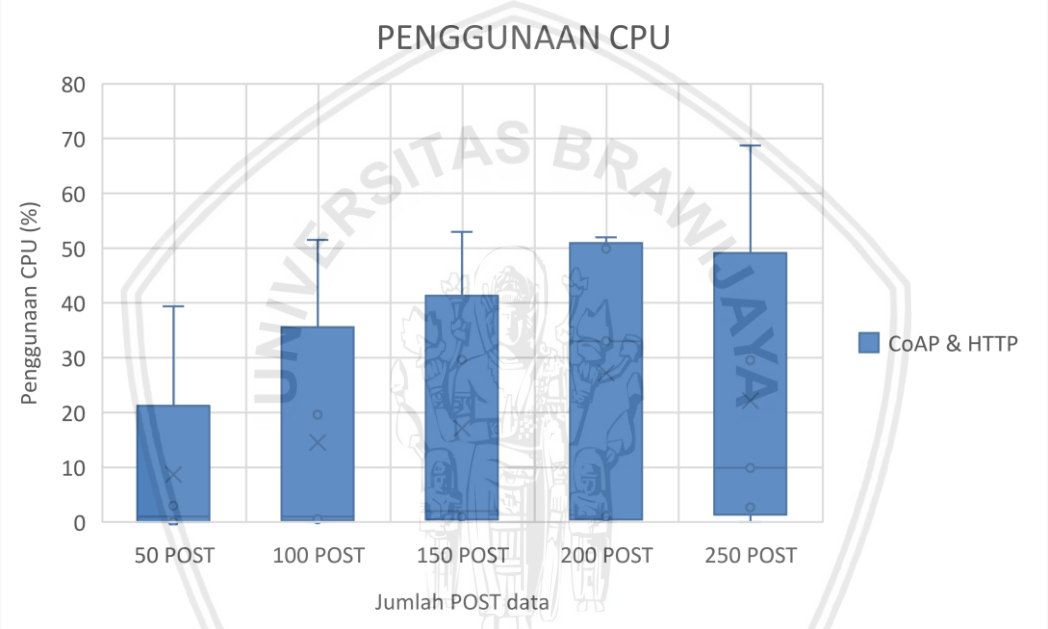
Tabel 6.8 Hasil penggunaan CPU

	Q1	MEDIAN	Q3	MAX	MIN
50 POST	0	1	3	39,4	0
100 POST	0,5	1	19,6	51,5	0
150 POST	1	2	29,65	53	0
200 POST	1	33	49,875	52	0
250 POST	2,675	9,9	29,55	68,7	0

Dalam tabel tersebut terdapar komponen berupa Q1, median, Q3, nilai terkecil dan nilai terbesar dari masing-masing skenario pengujian. Data dalam tabel tersebut kemudian dirubah menjadi bentuk *whisker plot* untuk

mempermudah proses analisis. Adapun *whisker plot* dari penggunaan memori ditunjukkan pada gambar 6.34.

Gambar 6.34 merupakan *whisker plot* penggunaan CPU dalam proses pengiriman data melalui CoAP dan HTTP secara bersama-sama kepada *semantic IoT web service*. Pada *whisker plot* tersebut, secara umum semakin tinggi data yang dikirim maka berbanding lurus pada penggunaan CPU. Namun pada pengiriman data sejumlah 200 terjadi peningkatan penggunaan CPU yang signifikan yaitu sebesar 33%. Salah satu faktor yang mempengaruhui hal tersebut adalah kecepatan pengiriman data yang menyebabkan *semantic IoT web serice* memerlukan CPU yang besar dalam menerima data tersebut.



Gambar 6.34 Hasil penggunaan CPU

6.4.3 *Runtime*

Runtime dari proses pengiriman data melalui protokol CoAP dan HTTP secara bersamaan kepada *semantic IoT web service* dapat dilihat pada tabel 6.9.

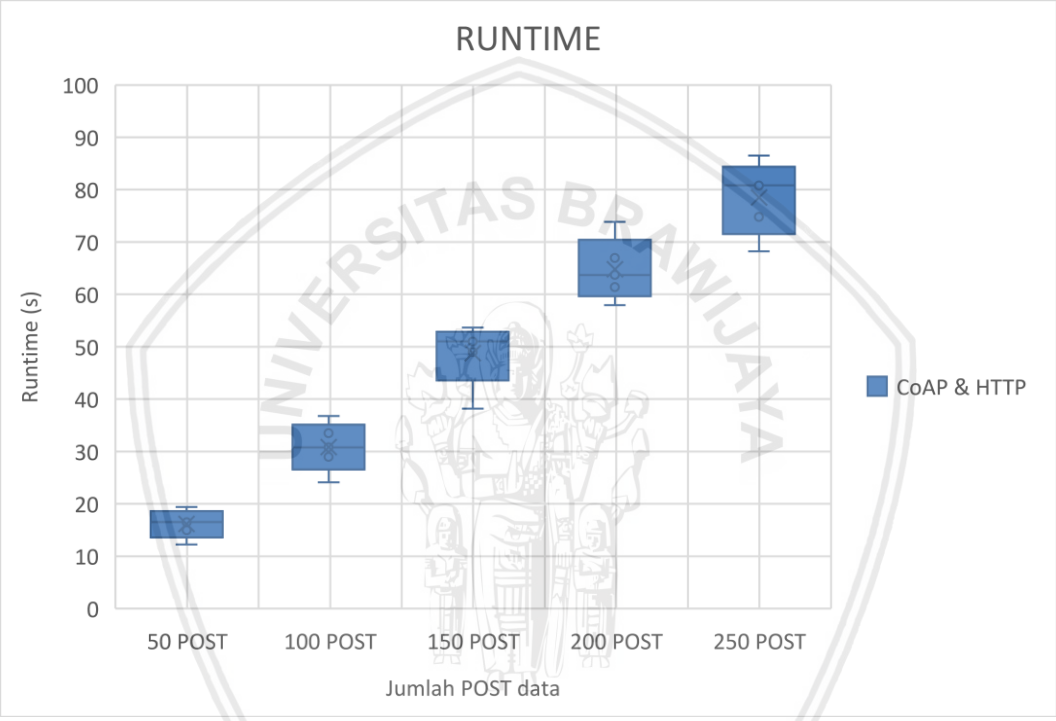
Tabel 6.9 Hasil *runtime*

	Q1	MEDIAN	Q3	MAX	MIN
50 POST	14,93561	16,51749	17,844	19,402	12,217
100 POST	28,9607	30,7888	33,52038	36,745	24,097
150 POST	48,95046	51,00309	52,0599	53,631	38,191
200 POST	61,40408	63,72053	66,99663	73,879	57,942
250 POST	74,78483	80,79981	82,22116	86,542	68,229

Dalam tabel 6.9 terdapat komponen berupa Q1, median, Q3, nilai terkecil dan nilai terbesar. Data yang didapat pada tabel tersebut merupakan rata-rata dari

runtime pada CoAP dan runtime pada HTTP pada masing-masing skenario pengujian. Data tersebut kemudian dirubah menjadi bentuk whisker plot untuk mempermudah proses analisis. Adapun whisker plot dari parameter *runtime* ditunjukkan pada gambar 6.35

Gambar 6.35 merupakan *whisker plot runtime* dalam proses pengiriman data kepada *semantic IoT web service* melalui protokol CoAP dan HTTP. Dalam *whisker plot* tersebut, semakin tinggi jumlah data yang dikirim, maka berbanding lurus dengan runtime. *Runtime* terlama pada pengiriman data sejumlah 250 yaitu 80,79981 seconds, sedangkan runtime tercepat pada pengiriman data sejumlah 50 yaitu 16,51749.



Gambar 6.35 Hasil pengujian runtime

6.4.4 Throughput

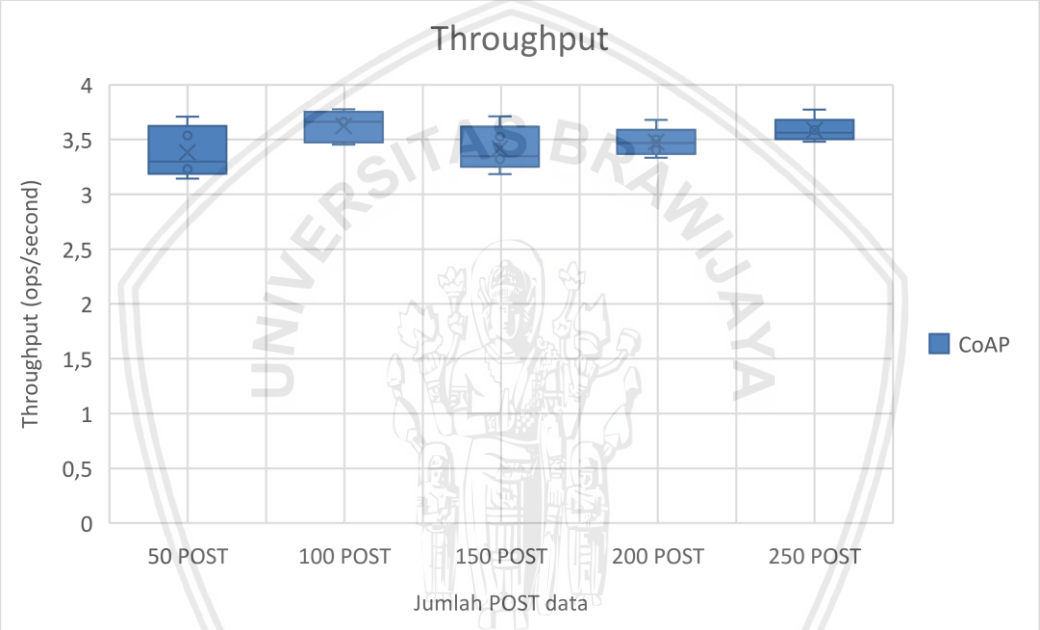
Throughput dari proses pengiriman data melalui protokol CoAP dan HTTP secara bersamaan kepada *semantic IoT web service* dapat dilihat pada tabel 6.10.

Tabel 6.10 Hasil *throughput*

	Q1	MEDIAN	Q3	MAX	MIN
50 POST	3,230207	3,298164	3,537377	3,707645	3,143596
100 POST	3,49613	3,661289	3,730021	3,777543	3,454437
150 POST	3,31997	3,349084	3,527149	3,711719	3,185139
200 POST	3,402356	3,469871	3,499706	3,680257	3,334507
250 POST	3,524864	3,562371	3,587028	3,774212	3,480148

Dalam tabel 6.10 terdapat komponen berupa Q1, median, Q3, nilai terkecil dan nilai terbesar. Data yang didapat pada tabel tersebut merupakan rata-rata dari *throughput* pada CoAP dan *throughput* pada HTTP pada masing-masing skenario pengujian. Data tersebut kemudian dirubah menjadi bentuk *whisker plot* untuk mempermudah proses analisis. Adapun *whisker plot* dari parameter *throughput* ditunjukkan pada gambar 6.36

Gambar 6.36 merupakan *whisker plot throughput* dari pengiriman data kepada *semantic IoT web service* melalui protokol CoAP dan HTTP. Dalam whisker plot tersebut, nilai *throughput* yang didapat beragam. Kenaikan signifikan terjadi pada pengiriman data sejumlah 100 yaitu 3,661289 ops/sec, sedangkan *throughput* terendah pada pengiriman data sejumlah 50 yaitu 3,298163713 ops/sec.



Gambar 6.36 Hasil pengujian *throughput*

Tabel 6.11 *Packet* pengiriman data JSON melalui CoAP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
13	4.514.187	182.1.77.223	206.189.94.98	CoAP	154	4.514.187.000	CON, MID:59253, POST, TKN:4e 71, /login
24	4.537.188	206.189.94.98	182.1.77.223	CoAP	48	4.537.188.000	ACK, MID:59253, 2.01 Created, TKN:4e 71, /login
35	4.623.173	182.1.77.223	206.189.94.98	CoAP	56	4.623.173.000	CON, MID:59254, GET, /login
44	4.626.553	206.189.94.98	182.1.77.223	CoAP	445	4.626.553.000	ACK, MID:59254, 2.05 Content
53	4.724.336	182.1.77.223	206.189.94.98	CoAP	527	4.724.336.000	CON, MID:59255, POST, TKN:73 4b, /postdata
68	4.763.909	206.189.94.98	182.1.77.223	CoAP	48	4.763.909.000	ACK, MID:59255, 2.01 Created, TKN:73 4b, /postdata
75	4.834.188	182.1.77.223	206.189.94.98	CoAP	56	4.834.188.000	CON, MID:59256, GET, /postdata
84	4.837.081	206.189.94.98	182.1.77.223	CoAP	129	4.837.081.000	ACK, MID:59256, 2.05 Content
96	5.914.072	182.1.77.223	206.189.94.98	CoAP	527	5.914.072.000	CON, MID:59257, POST, TKN:58 53, /postdata
114	5.950.057	206.189.94.98	182.1.77.223	CoAP	48	5.950.057.000	ACK, MID:59257, 2.01 Created, TKN:58 53, /postdata
121	6.034.252	182.1.77.223	206.189.94.98	CoAP	56	6.034.252.000	CON, MID:59258, GET, /postdata
130	6.037.010	206.189.94.98	182.1.77.223	CoAP	129	6.037.010.000	ACK, MID:59258, 2.05 Content
143	7.153.092	182.1.77.223	206.189.94.98	CoAP	527	7.153.092.000	CON, MID:59259, POST, TKN:46 4c, /postdata
158	7.201.299	206.189.94.98	182.1.77.223	CoAP	48	7.201.299.000	ACK, MID:59259, 2.01 Created, TKN:46 4c, /postdata
165	7.303.088	182.1.77.223	206.189.94.98	CoAP	56	7.303.088.000	CON, MID:59260, GET, /postdata
174	7.305.787	206.189.94.98	182.1.77.223	CoAP	129	7.305.787.000	ACK, MID:59260, 2.05 Content

Tabel 6.12 *Packet* pengiriman data JSON melalui HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
17	5.174.372	182.1.65.236	206.189.94.98	HTTP	152	5.174.372.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
19	5.198.351	206.189.94.98	182.1.65.236	HTTP	725	5.198.351.000	HTTP/1.1 200 OK (application/json)
30	5.639.058	182.1.64.199	206.189.94.98	HTTP	170	5.639.058.000	POST /topic/jsonhttptopic HTTP/1.1 (application/x-www-form-urlencoded)
32	5.697.139	206.189.94.98	182.1.64.199	HTTP	482	5.697.139.000	HTTP/1.1 200 OK (application/json)
44	8.992.395	182.1.64.199	206.189.94.98	HTTP	170	8.992.395.000	POST /topic/jsonhttptopic HTTP/1.1 (application/x-www-form-urlencoded)
46	9.021.746	206.189.94.98	182.1.64.199	HTTP	482	9.021.746.000	HTTP/1.1 200 OK (application/json)
56	12.314.677	182.1.65.198	206.189.94.98	HTTP	170	12.314.677.000	POST /topic/jsonhttptopic HTTP/1.1 (application/x-www-form-urlencoded)
58	12.339.558	206.189.94.98	182.1.65.198	HTTP	482	12.339.558.000	HTTP/1.1 200 OK (application/json)

Tabel 6.13 *Packet* pengiriman data gambar melalui CoAP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
70	19.530.429	182.1.77.223	206.189.94.98	CoAP	154	19.530.429.000	CON, MID:37919, POST, TKN:48 63, /login
81	19.555.652	206.189.94.98	182.1.77.223	CoAP	48	19.555.652.000	ACK, MID:37919, 2.01 Created, TKN:48 63, /login
92	19.641.204	182.1.77.223	206.189.94.98	CoAP	56	19.641.204.000	CON, MID:37920, GET, /login
101	19.645.473	206.189.94.98	182.1.77.223	CoAP	463	19.645.473.000	ACK, MID:37920, 2.05 Content
110	19.914.456	182.1.77.223	206.189.94.98	CoAP	1085	19.914.456.000	CON, MID:37921, POST, TKN:68 71, Block #0, /postdata

Tabel 6.13 *Packet* pengiriman data gambar melalui CoAP (Lanjutan)

119	19.920.133	206.189.94.98	182.1.77.223	CoAP	51	19.920.133.000	ACK, MID:37921, 2.31 Continue, TKN:68 71, Block #0, /postdata
128	20.011.229	182.1.77.223	206.189.94.98	CoAP	1085	20.011.229.000	CON, MID:37922, POST, TKN:68 71, Block #1, /postdata
137	20.016.936	206.189.94.98	182.1.77.223	CoAP	51	20.016.936.000	ACK, MID:37922, 2.31 Continue, TKN:68 71, Block #1, /postdata
146	20.091.422	182.1.77.223	206.189.94.98	CoAP	1085	20.091.422.000	CON, MID:37923, POST, TKN:68 71, Block #2, /postdata
155	20.096.833	206.189.94.98	182.1.77.223	CoAP	51	20.096.833.000	ACK, MID:37923, 2.31 Continue, TKN:68 71, Block #2, /postdata

Tabel 6.14 *Packet* pengiriman data gambar melalui HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
15	4.602.557	182.1.65.236	206.189.94.98	HTTP	152	4.602.557.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
17	4.623.490	206.189.94.98	182.1.65.236	HTTP	729	4.623.490.000	HTTP/1.1 200 OK (application/json)
52	4.985.191	182.1.65.236	206.189.94.98	HTTP	929	4.985.191.000	POST /topic/imagecoaptopic HTTP/1.1
54	5.025.555	206.189.94.98	182.1.65.236	HTTP	486	5.025.555.000	HTTP/1.1 200 OK (application/json)

Tabel 6.15 *Packet* pengiriman data video melalui CoAP

No.	**Time**	**Source**	**Destination**	**Protocol**	**Length**	**Time since reference or first frame**	**Info**
8	3.893.568	182.1.77.223	206.189.94.98	CoAP	154	3.893.568.000	CON, MID:10722, POST, TKN:66 6d, /login
19	3.916.471	206.189.94.98	182.1.77.223	CoAP	48	3.916.471.000	ACK, MID:10722, 2.01 Created, TKN:66 6d, /login
30	4.012.797	182.1.77.223	206.189.94.98	CoAP	56	4.012.797.000	CON, MID:10723, GET, /login

Tabel 6.15 *Packet* pengiriman data video melalui CoAP (Lanjutan)

39	4.016.407	206.189.94.98	182.1.77.223	CoAP	445	4.016.407.000	ACK, MID:10723, 2.05 Content
48	4.203.616	182.1.77.223	206.189.94.98	CoAP	1085	4.203.616.000	CON, MID:10724, POST, TKN:4e 5a, Block #0, /postdata
57	4.206.092	206.189.94.98	182.1.77.223	CoAP	51	4.206.092.000	ACK, MID:10724, 2.31 Continue, TKN:4e 5a, Block #0, /postdata
66	4.325.669	182.1.77.223	206.189.94.98	CoAP	1085	4.325.669.000	CON, MID:10725, POST, TKN:4e 5a, Block #1, /postdata
75	4.331.461	206.189.94.98	182.1.77.223	CoAP	51	4.331.461.000	ACK, MID:10725, 2.31 Continue, TKN:4e 5a, Block #1, /postdata
84	4.479.788	182.1.77.223	206.189.94.98	CoAP	1085	4.479.788.000	CON, MID:10726, POST, TKN:4e 5a, Block #2, /postdata
93	4.483.776	206.189.94.98	182.1.77.223	CoAP	51	4.483.776.000	ACK, MID:10726, 2.31 Continue, TKN:4e 5a, Block #2, /postdata
102	4.609.961	182.1.77.223	206.189.94.98	CoAP	1085	4.609.961.000	CON, MID:10727, POST, TKN:4e 5a, Block #3, /postdata
111	4.613.260	206.189.94.98	182.1.77.223	CoAP	51	4.613.260.000	ACK, MID:10727, 2.31 Continue, TKN:4e 5a, Block #3, /postdata
120	4.735.813	182.1.77.223	206.189.94.98	CoAP	1085	4.735.813.000	CON, MID:10728, POST, TKN:4e 5a, Block #4, /postdata

Tabel 6.16 *Packet* pengiriman data video melalui HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
14	2.687.278	182.1.82.108	206.189.94.98	HTTP	152	2.687.278.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
16	2.710.867	206.189.94.98	182.1.82.108	HTTP	727	2.710.867.000	HTTP/1.1 200 OK (application/json)
255	4.256.555	182.1.65.236	206.189.94.98	HTTP	1836	4.256.555.000	POST /topic/videocoaptopic HTTP/1.1
261	4.290.044	206.189.94.98	182.1.65.236	HTTP	487	4.290.044.000	HTTP/1.1 200 OK (application/json)

Tabel 6.17 *Packet* pengiriman data JSON melalui CoAP dan HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
15	14.122.694	202.80.212.194	206.189.94.98	CoAP	154	14.122.694.000	CON, MID:12447, POST, TKN:6e 49, /login
25	14.508.079	202.80.212.194	206.189.94.98	CoAP	56	14.508.079.000	CON, MID:12448, GET, /login
38	14.566.617	202.80.212.194	206.189.94.98	CoAP	575	14.566.617.000	CON, MID:12449, POST, TKN:4a 4b, /postdata
56	14.670.926	202.80.212.194	206.189.94.98	CoAP	56	14.670.926.000	CON, MID:12450, GET, /postdata
72	15.495.791	202.80.212.194	206.189.94.98	HTTP	392	15.495.791.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
82	15.929.462	202.80.212.194	206.189.94.98	HTTP	841	15.929.462.000	POST /topic/jsoninterop HTTP/1.1 (application/x-www-form-urlencoded)
89	17.741.267	202.80.212.194	206.189.94.98	CoAP	575	17.741.267.000	CON, MID:12451, POST, TKN:46 6e, /postdata
103	18.957.015	202.80.212.194	206.189.94.98	CoAP	56	18.957.015.000	CON, MID:12452, GET, /postdata
110	19.330.790	202.80.212.194	206.189.94.98	HTTP	841	19.330.790.000	POST /topic/jsoninterop HTTP/1.1 (application/x-www-form-urlencoded)
123	22.029.212	202.80.212.194	206.189.94.98	CoAP	575	22.029.212.000	CON, MID:12453, POST, TKN:5a 73, /postdata
139	22.112.680	202.80.212.194	206.189.94.98	CoAP	56	22.112.680.000	CON, MID:12454, GET, /postdata
156	22.479.625	202.80.212.194	206.189.94.98	HTTP	841	22.479.625.000	POST /topic/jsoninterop HTTP/1.1 (application/x-www-form-urlencoded)

Tabel 6.18 *Packet* pengiriman data gambar melalui CoAP dan HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
235	13.482.008	202.80.212.194	206.189.94.98	CoAP	1085	13.482.008.000	CON, MID:24369, POST, TKN:6e 6b, Block #15, /postdata
245	13.564.649	202.80.212.194	206.189.94.98	CoAP	1086	13.564.649.000	CON, MID:24370, POST, TKN:6e 6b, Block #16, /postdata
264	13.611.977	202.80.212.194	206.189.94.98	CoAP	1086	13.611.977.000	CON, MID:24371, POST, TKN:6e 6b, Block #17, /postdata
278	13.626.346	202.80.212.194	206.189.94.98	HTTP	392	13.626.346.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
285	13.674.128	202.80.212.194	206.189.94.98	CoAP	1086	13.674.128.000	CON, MID:24372, POST, TKN:6e 6b, Block #18, /postdata
299	13.733.175	202.80.212.194	206.189.94.98	CoAP	1086	13.733.175.000	CON, MID:24373, POST, TKN:6e 6b, Block #19, /postdata
313	13.780.631	202.80.212.194	206.189.94.98	CoAP	1086	13.780.631.000	CON, MID:24374, POST, TKN:6e 6b, Block #20, /postdata
354	13.842.011	202.80.212.194	206.189.94.98	HTTP	920	13.842.011.000	POST /topic/gambarinterop HTTP/1.1
356	13.847.413	202.80.212.194	206.189.94.98	CoAP	1086	13.847.413.000	CON, MID:24375, POST, TKN:6e 6b, Block #21, /postdata

Tabel 6.19 *Packet* pengiriman data video melalui CoAP dan HTTP

No.	Time	Source	Destination	Protocol	Length	Time since reference or first frame	Info
2199	32.325.501	202.80.212.194	206.189.94.98	CoAP	1086	32.325.501.000	CON, MID:48098, POST, TKN:49 4e, Block #163, /postdata
2214	32.362.920	202.80.212.194	206.189.94.98	CoAP	1086	32.362.920.000	CON, MID:48099, POST, TKN:49 4e, Block #164, /postdata

Tabel 6.19 *Packet* pengiriman data video melalui CoAP dan HTTP (Lanjutan)

2228	32.401.626	202.80.212.194	206.189.94.98	CoAP	1086	32.401.626.000	CON, MID:48100, POST, TKN:49 4e, Block #165, /postdata
2242	32.442.621	202.80.212.194	206.189.94.98	HTTP	392	32.442.621.000	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)
2244	32.443.728	202.80.212.194	206.189.94.98	CoAP	1086	32.443.728.000	CON, MID:48101, POST, TKN:49 4e, Block #166, /postdata
2255	32.487.900	202.80.212.194	206.189.94.98	CoAP	1086	32.487.900.000	CON, MID:48102, POST, TKN:49 4e, Block #167, /postdata
2267	32.524.720	202.80.212.194	206.189.94.98	CoAP	1086	32.524.720.000	CON, MID:48103, POST, TKN:49 4e, Block #168, /postdata
2280	32.561.432	202.80.212.194	206.189.94.98	CoAP	1086	32.561.432.000	CON, MID:48104, POST, TKN:49 4e, Block #169, /postdata
2299	32.600.915	202.80.212.194	206.189.94.98	CoAP	1086	32.600.915.000	CON, MID:48105, POST, TKN:49 4e, Block #170, /postdata
2330	32.660.898	202.80.212.194	206.189.94.98	CoAP	1086	32.660.898.000	CON, MID:48106, POST, TKN:49 4e, Block #171, /postdata
2381	32.741.853	202.80.212.194	206.189.94.98	CoAP	1086	32.741.853.000	CON, MID:48107, POST, TKN:49 4e, Block #172, /postdata
2451	32.863.665	202.80.212.194	206.189.94.98	CoAP	1086	32.863.665.000	CON, MID:48108, POST, TKN:49 4e, Block #173, /postdata
2515	32.957.377	202.80.212.194	206.189.94.98	CoAP	1086	32.957.377.000	CON, MID:48109, POST, TKN:49 4e, Block #174, /postdata
2583	33.042.893	202.80.212.194	206.189.94.98	HTTP	458	33.042.893.000	POST /topic/videointerop HTTP/1.1
2585	33.048.857	202.80.212.194	206.189.94.98	CoAP	1086	33.048.857.000	CON, MID:48110, POST, TKN:49 4e, Block #175, /postdata
2598	33.087.660	202.80.212.194	206.189.94.98	CoAP	1086	33.087.660.000	CON, MID:48111, POST, TKN:49 4e, Block #176, /postdata

BAB 7 PENUTUP

Pada bab ini akan dijelaskan mengenai kesimpulan yang didapat dari seluruh proses, mulai dari perancangan, implementasi dan pengujian. Selain itu diberikan pula saran sebagai rujukan pengembangan dari penelitian yang saat ini dilakukan.

7.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi dan pengujian yang dilakukan, adapun kesimpulan yang diambil yaitu:

1. *Constrained Application Protocol* (CoAP) berhasil ditambahkan pada *semantic IoT web service* melalui implementasi *application programming interface* (API) menggunakan CoAPthon. Implementasi dari API tersebut terbagi dua bagian yaitu API CoAP *client* untuk mengirim data serta API CoAP *server* untuk menerima data dan meneruskannya kepada *semantic IoT web service*. Selain itu, *semantic IoT web service* berhasil menerima data berbentuk JSON, gambar dan video melalui protokol komunikasi CoAP.
2. *Semantic IoT web service* yang telah ditambah CoAP mampu mengatasi permasalahan interoperabilitas terutama *syntactical interoperability* dengan menyediakan protokol komunikasi melalui CoAP dan HTTP kepada *node sensor*. CoAP dan HTTP dapat bekerja secara independen maupun bersama-sama dalam proses pengiriman data kepada *semantic IoT web service*. Dari sisi integritas data, data yang dikirim oleh *node sensor* memiliki struktur yang sama dengan data yang disimpan dalam data *storage* maupun yang ditampilkan dalam IoT *application*.
3. Kinerja *semantic IoT web service* yang telah ditambah CoAP yang ditinjau dari proses POST data adalah sebagai berikut:
    a. Penggunaan memori tertinggi sebesar 685701120 bytes dan terendah sebesar 677339136 bytes.
    b. Penggunaan CPU tertinggi sebesar 33% dan terendah sebesar 1%.
    c. Waktu terlama yang dibutuhkan dalam proses pengiriman data adalah sebesar 80,79981 seconds, sedangkan tercepat adalah sebesar 16,51749 seconds.
    d. *Throughput* terendah yang dihasilkan adalah sebesar 3,661289 ops/sec dan tertinggi sebesar 3,298163713 ops/sec.

7.2 Saran

Berdasarkan kesimpulan yang telah diberikan oleh peneliti, adapun saran yang diberikan dengan harapan dapat dilanjutkan untuk penelitian selanjutnya yaitu:

1. Perlu ditambahkan kembali protokol komunikasi pada *semantic IoT web service* sehingga mampu menerima data dari *node sensor* melalui banyak protokol komunikasi.
2. Perlu ditambahkan enkripsi terhadap data yang dikirim dari *node sensor* kepada *semantic IoT web service* demi menjaga aspek *confidentiality* terhadap data tersebut.

DAFTAR REFERENSI

THIYAGARAJAN, D. M. & RAVEENDRA, D. C., 2017. ROLE OF WEB SERVICE IN INTERNET OF THINGS. *IEEE,* pp. 268-270.

Chander, R. V., Mukherjee, S. & Elias, S., 2017. An applications interoperability model for heterogeneous internet of things environments. *ScienceDirect,* pp. 1-10.

Dalipi, E. et al., 2016. EC-IoT: An Easy Configuration Framework for Constrained IoT Devices. *IEEE,* pp. 159-164.

Desai, P., Sheth, A. & Anantharam, P., 2015. Semantic Gateway as a Service architecture for IoT Interoperability. *IEEE,* pp. 313-319.

Guoqiang, S., Yanming, C., Chao, Z. & Yanxu, Z., 2013. Design and Implementation of a Smart IoT Gateway. *IEEE.*

Hamid, T. W., Pramukantoro, E. S. & Siregar, R. A., 2018. Pengembangan Web Service Pada IoT Data StorageDengan Pendekatan Semantik dan Penambahan Mekanisme Autentikasi. *JPTIK,* pp. 6824-6827.

Hou, L. et al., 2016. Internet ofThings Cloud: Architecture and Implementation. *IEEE,* pp. 32-39.

Kurniawan, A. K., Pramukantoro, E. S. & Trisnawan, P. H., 2018. Perbandingan Kinerja Cassandra dan MongoDB Sebagai Backend IoT Data Storage. *JPTIIK,* Volume III, pp. 364-371.

Madhu, C., Vijayakumar, P. & Gao, X. Z., 2017. RESOURCE CONSTRAINED IOT ENVIRONMENTS: A SURVEY. *Journal of Advance Research in Dynamic and Control Systems,* pp. 445-457.

Pramukantoro, E. S., Bakhtiar, F. A. & Bhawiyuga, A., 2019. A Semantic RESTful API for Heterogeneous IoT Data Storage. pp. 1-2.

Rezaei, R., Chiew, T. K., Lee, S. P. & Aliee, Z. S., 2014. Interoperability evaluation models: A systematic review. *ScienceDirect,* pp. 1-23.

Ruta, M. et al., 2017. A CoAP-based framework for collaborative sensing in the Semantic Web of Things. *ScienceDirect,* pp. 1047-1052.

Shah, S. H. & Yaqoob, I., 2016. A Survey: Internet of Things (IOT) Technologies, Applications and Challenges. *IEEE,* pp. 381-385.

Tanganelli, G., Vallati, C. & Mingozzi, E., 2015. CoAPthon: Easy Development of CoAP-based IoT. *IEEE.*

Wiryawan, Y. F., Kartikasari, P. D. & Data, M., 2018. Implementasi Constrained Application Protocol(CoAP) pada Sistem Pengamatan Kelembaban Tanah. *JPTIK,* Volume II, pp. 2480-2487.