PENGEMBANGAN *MIDDLEWARE WEB OF THINGS* SEBAGAI ANTARMUKA AKSES *BATCH* DAN *STREAM* PERANGKAT SENSOR BERBASIS PROTOKOL *BLUETOOTH LOW ENERGY*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Satria Adhi Kharisma
NIM: 155150207111119

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

PENGESAHAN

PENGEMBANGAN *MIDDLEWARE WEB OF THINGS* SEBAGAI *ANTARMUKA* AKSES *BATCH* DAN *STREAM* PERANGKAT SENSOR BERBASIS PROTOKOL *BLUETOOTH LOW ENERGY*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Satria Adhi Kharisma
NIM: 155150207111119

Skripsi ini telah diuji dan dinyatakan lulus pada
4 Desember 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Adhitya Bhawiyuga, S.Kom., M.Sc.	Eko Sakti Pramukantoro, S.Kom., M.Kom.
NIP: 19890720 201803 1 002	NIK: 201102 860805 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, 10 Desember 2019

Satria Adhi Kharisma

NIM: 155150207111119

PRAKATA

Puji syukur penulis panjatkan atas kehadirat Allah, karena berkat rahmat dan karunia-nya, penulis dapat menyelesaikan laporan skripsi yang berjudul "PENGEMBANGAN *MIDDLEWARE WEB OF THINGS* SEBAGAI ANTARMUKA AKSES *BATCH* DAN *STREAM* PERANGKAT SENSOR BERBASIS PROTOKOL *BLUETOOTH LOW ENERGY*" 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 Adhitya Bhawiyuga, S.Kom., M.Sc., dan Bapak Eko Sakti Pramukantoro, S.Kom., M.Kom selaku dosen pembimbing skripsi yang telah sabar dan tekun dalam 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 dukungan 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. Yani, Husein, Kawan F9, dan keluarga besar POROS yang selalu membantu dan memberikan motivasi kepada penulis dalam menyelesaikan penelitian 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, 10 Desember 2019

Penulis

sakharisma@gmail.com

ABSTRAK

Satria Adhi Kharisma, Pengembangan *Middleware Web Of Things* Sebagai Antarmuka *Akses Batch* Dan *Stream* Perangkat Sensor Berbasis Protokol *Bluetooth Low Energy*

Pembimbing: Adhitya Bhawiyuga, S.Kom., M.Sc. dan Eko Sakti Pramukantoro, S.Kom., M.Kom

Bidang kesehatan merupakan salah satu bidang yang menarik dalam penerapan IoT. Namun, perangkat yang digunakan pada penerapan *IoT-based healthcare* sangat heterogen hal ini menjadi tantangan bagi pengembang aplikasi untuk mengatasi perbedaan mekanisme komunikasi pada perangkat medis. Selain itu, aplikasi IoT membutuhkan pengolahan data secara *batch* dan *stream*. Berdasarkan tantangan dan kebutuhan tersebut penulis mengembangkan *middleware* untuk mengatasi tantangan tersebut. *Middleware* terdiri dari tiga komponen *sensor-to-cloud gateway*, *messaging service*, dan *data access interface*. Pada akhir pengembangan dilakukan pengujian kinerja pada *batch interface*, *stream interface*, dan *concurrent user*. Pada pengujian *batch interface* didapatkan nilai *throughput* tertinggi pada penarikan data berukuran 0,5 MB yaitu 2,7 *request/s*. dan *latency* tertinggi pada data berukuran 10 MB yaitu 86,58 detik. Pengujian *stream interface* menunjukkan nilai *end-to-end delay* dari variasi *interval* penarikan data, HTTP memiliki *delay* yang lebih tinggi yakni sebesar 1,19 detik dan MQTT sebesar 0,55 detik. Hal ini disebabkan mekanisme *3-way-handshake* di tiap retrieve pada HTTP. Pengujian *concurrent* user menunjukkan perbedaan *throughput* yang cukup besar pada variasi user 2000 yakni 110,5 request/s pada *batch* dan 172,4 request/s pada *stream*.

Kata kunci: *Internet of Things*, *Web of Things*, *Middleware*, *Cloud*, BLE, *Stream*, *Batch*

ABSTRACT

Satria Adhi Kharisma, Pengembangan *Middleware Web Of Things* Sebagai Antarmuka *Akses Batch* Dan *Stream* Perangkat Sensor Berbasis Protokol *Bluetooth Low Energy*

Supervisors: Adhitya Bhawiyuga dan Eko Sakti Pramukantoro

The health sector is one of the interesting fields in the application of IoT. However, the devices used in the implementation of IoT-based healthcare are very heterogeneous, this is a challenge for application developers to overcome the different communication mechanisms in sensing devices. In addition, the IoT application requires batch and stream data processing. Based on these challenges and requirements, the authors develop middleware to overcome these challenges. Middleware consists of three components it is sensor-to-cloud gateway components, messaging services, and data access interfaces. At the end of development, performance testing is performed on batch interface, stream interface and concurrent user. In the batch interface test, the highest throughput value is obtained at 0.5 MB of data retrieval, 2.7 requests/s. and the highest latency in 10MB of data is 86.58 seconds. Stream interface testing shows the value of end-to-end delay from the variation of data retrieval intervals, HTTP has a higher delay of 1.19 seconds and MQTT of 0.55 seconds. Due to the 3-way-handshake mechanism for each retrieve on HTTP. Concurrent user testing shows a significant difference in throughput at 2000 user, that is 110.5 requests/s in batch and 172.4 requests/s in stream.

Keywords: *Internet of Things*, *Web of Things*, *Middleware*, *Cloud*, *BLE*, *Stream*, *Batch*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Internet of Things (IoT) merupakan paradigma komunikasi masa depan yang memungkinkan objek kehidupan sehari-hari untuk dapat berkomunikasi satu sama lain melalui internet (Atzori, Iera, & Morabito., 2010). Saat ini IoT telah diaplikasikan di berbagai bidang seperti *smart city*, *smart healthcare*, *smart farming*, dan *smart things* lainnya. Berbagai aplikasi IoT seperti pemantau polusi udara sebuah kota, pendeteksi dini bencana alam sebuah negara, hingga pemantau kondisi pertumbuhan padi dapat dibangun melalui konsep IoT. Bidang kesehatan merupakan salah satu bidang yang paling menarik dalam penerapan IoT (Pang. Z., 2013). IoT berpotensi melahirkan banyak aplikasi medis seperti pendeteksi dini penyakit kronis, pemantau kebugaran, hingga pemantauan kesehatan pasien secara *remote*. Potensi tersebut didukung dengan adanya beragam jenis perangkat, sensor, dan alat diagnosa pada bidang kesehatan yang dapat digunakan sebagai *smart device* pada *smart healthcare* berbasis IoT. Karakteristik umum *smart healthcare* berbasis IoT yakni terdiri dari sistem, perangkat dan aplikasi yang digunakan dokter untuk memantau dan merekam kondisi medis pasien (Gupta et al., 2016). Perangkat pada *IoT-based healthcare* umumnya memiliki sensor yang dipasang di tubuh pasien untuk mengambil data vital seperti detak jantung, suhu tubuh, dan sebagainya lalu mengirimkan data tersebut melalui antarmuka jaringan. Lalu dokter menggunakan aplikasi pada *platform* tertentu yang terhubung dengan sistem untuk memantau data vital yang dihasilkan oleh perangkat tersebut secara remote.

Pada penerapan *smart healthcare* berbasis IoT jenis perangkat sensor yang digunakan sangat heterogen karena setiap perangkat sensor memiliki mekanisme kerja yang beragam (Islam et al., 2015). Sebagai contoh sensor ECG, PPG, temperatur tubuh dan lainnya memiliki mekanisme kerja yang berbeda. Selain memperluas potensi, hal ini menjadi tantangan bagi pengembang aplikasi dalam mengatasi perbedaan mekanisme komunikasi pada perangkat sensor. Selain itu, aplikasi IoT memiliki beragam jenis pengolahan pada data yang dihasilkan oleh perangkat sensor. Pengolahan data tersebut dibagi menjadi pengolahan data secara *batch* dan pengolahan data secara *stream*. Pengolahan data secara *batch* membutuhkan data yang bersifat historis dimana data lebih dahulu disimpan dan dikumpulkan sebelum diolah. Sedangkan pada pengolahan data secara *stream* dibutuhkan data terbaru untuk diolah tanpa menyimpan data tersebut terlebih dahulu (Shahrivari, 2014). Pada penerapan *smart healthcare* berbasis IoT pengolahan data secara *batch* dilakukan untuk menganalisis *Electronic Health Record* (EHR) yang berisi tentang riwayat medis seorang pasien. Sebaliknya, pengolahan data secara *stream* dilakukan untuk menganalisis data terbaru pasien secara *real-time* untuk memantau pasien yang memiliki kondisi medis tertentu dalam konteks intensif (Reddy dan Aggarwal, ed., 2015). Adanya kebutuhan pengolahan data yang berbeda menghadirkan tantangan bagi aplikasi untuk dapat mengakses data secara *batch* dan *stream*.

Berdasarkan tantangan dan kebutuhan yang telah dijelaskan *platform web* dapat menjadi antarmuka bagi perangkat dan aplikasi untuk saling berinteraksi berkat penggunaan teknologi *web* pada internet yang tersebar secara masal (Heuer et al., 2015). Integrasi antara teknologi *web* dan IoT akan melahirkan konsep lain yang disebut *Web of Things* (WoT). Implementasi dari konsep ini ialah mengubah seluruh perangkat IoT tanpa memandang bentuk data maupun protokol komunikasinya menjadi sumber daya *web* (Ragget. D., 2015). Abstraksi WoT melalui *middleware* akan memudahkan pengembang untuk mengakses data pada perangkat IoT dan memanfaatkannya untuk mengembangkan aplikasi.

Pengembangan *middleware* berbasis WoT sebelumnya telah dilakukan melalui perancangan *middleware* berbasis RESTful *web service*. *Middleware* tersebut menerapkan konsep *Web of Things* (WoT) sebagai abstraksi perangkat sensor menjadi *web resource*. Melalui abstraksi tersebut *middleware* mampu menyediakan layanan bagi perangkat mobile untuk berinteraksi dan mengakses data dari perangkat sensor menggunakan *Uniform Resource Identifier* (URI) yang difasilitasi oleh *middleware* melalui RESTful *webservice* (N. Philip et al., 2014). Namun desain pada pengembangan tersebut tidak memfasilitasi pengaksesan data secara *stream*. Selain itu penggunaan HTTP pada sinkronisasi antara *device manager* dan *cloud* menyebabkan terjadinya *overhead*, hal ini muncul dikarenakan mekanisme *three-way-handshake* yang terjadi pada tiap siklus sinkronisasi.

Berdasarkan tantangan yang telah dijelaskan, penulis mengembangkan *middleware* untuk menjembatani komunikasi antara aplikasi dan perangkat sensor kesehatan serta memfasilitasi pengaksesan data secara *batch* dan *stream*. *Middleware* tersebut berbasis *cloud* yang terdiri dari tiga komponen yaitu *sensor-to-cloud gateway*, *messaging service*, dan *data access interface*. *Sensor-to-cloud gateway* berperan untuk menangani perangkat sensor dengan menggunakan protokol *Bluetooth Low Energy* (BLE) untuk saling berkomunikasi. Untuk mengelola perangkat, *gateway* mula-mula membangun koneksi dengan perangkat sensor. Setelah terhubung *gateway* akan menerima data yang dihasilkan oleh perangkat dan mengirimnya ke *cloud*. *Messaging service* berperan untuk menerima data yang dikirim oleh *sensor-to-cloud gateway*, memfasilitasi akses data secara *batch* dengan menyimpan data kedalam *database*, dan memfasilitasi akses secara *stream* menggunakan *MQTT Over Websocket*. *Data Access Interface* berperan untuk menyediakan akses data dengan membagi *interface* menjadi *batch access interface* dan *stream access interface*. *Batch access interface* memfasilitasi akses *batch* untuk aplikasi melalui abstraksi menggunakan URI pada RESTful HTTP. *Stream access interface* memfasilitasi akses stream untuk aplikasi melalui abstraksi menggunakan topik pada *MQTT*.

1.2 Rumusan Masalah

Rumusan permasalahan yang dibahas pada penelitian ini yaitu:

1. Bagaimana *middleware* mengatasi heterogenitas jenis perangkat *wearable sensor* pada *healthcare* berbasis IoT?

2. Bagaimana *middleware* memfasilitasi akses perangkat sensor secara *batch* dan *stream*?
3. Bagaimana pengujian fungsionalitas dan kinerja dari *middleware?*

1.3 Tujuan

Tujuan yang ingin dicapai pada penelitian ini adalah sebagai berikut:

1. *Middleware* mampu mengatasi heterogenitas jenis perangkat *wearable sensor* pada *healthcare* berbasis IoT.
2. *Middleware* mampu memfasilitasi akses perangkat secara *batch* dan *stream*.
3. Mengetahui kemampuan fungsional dan kinerja *middleware* yang diimplementasikan.

1.4 Manfaat

Hasil dari pengembangan *middleware* ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Memudahkan pengembangan aplikasi kesehatan di berbagai *platform* baik *web* maupun *native* tanpa harus mengkhawatirkan format data dan jenis perangkat sensor yang heterogen.
2. Menjadi referensi pengembangan *web of things middleware* pada penelitian-penelitian selanjutnya.

1.5 Batasan Masalah

Batasan mengenai masalah yang diangkat pada penelitian ini dibuat untuk memberikan rincian dan fokus pelaksanaan penelitian ini. Berikut batasan masalah pada penelitian ini:

1. Pengembangan ini hanya menggunakan perangkat sensor dengan protokol komunikasi berbasis BLE.
2. Pengembangan tidak berfokus pada pengembangan *sensor device* ataupun aplikasi user.
3. Pengembangan berfokus pada penyediaan antarmuka akses batch dan stream pada perangkat sensor berbasis BLE.
4. Lingkungan pengujian menggunakan *cloud* dan jaringan publik.
5. Pengujian hanya berfokus pada fungsionalitas dan kinerja dari *middleware*.

1.6 Sistematika Pembahasan

Pada penelitian ini disusun suatu sistematika penulisan yang bertujuan untuk memberikan gambaran umum proses penelitian. Berikut sistematika penulisan yang akan diuraikan:

BAB I PENDAHULUAN

Bab ini menjelaskan latar belakang penelitian, rumusan masalah, tujuan penelitian, manfaat dan batasan masalah yang diangkat serta uraian sistematika penulisan penelitian.

BAB II KAJIAN PUSTAKA

Pembahasan dasar-dasar teori dan objek penelitian dibahas pada bab ini. Uraian penjelasan pada bab ini kemudian dijadikan dasar dalam merancang, mengimplementasikan dan menganalisis sistem yang dibangun.

BAB III METODOLOGI PENELITIAN

Bab ini menggambarkan tahapan yang dilakukan penulis dalam menyelesaikan penelitian. Tahapan yang disusun meliputi pendahuluan, kajian pustaka, metodologi, perancangan, implementasi, pengujian dan penutup.

BAB IV ANALISIS KEBUTUHAN DAN PERANCANGAN

Bab ini berisi gambaran umum sistem yang menjelaskan tahap elisitasi kebutuhan fungsional dan non-fungsional yang harus dipenuhi oleh *middleware*. Pada bab ini juga digambarkan proses abstraksi data dari perangkat sensor berbasis BLE menuju aplikasi melalui *middleware*. Secara spesifik proses abstraksi yang menggunakan konsep WoT dijelaskan dalam perancangan alur sistem dan arsitektur *middleware* yang akan di implementasikan.

BAB V IMPLEMENTASI

Bab ini menjelaskan proses implementasi *middleware* dengan membagi implementasi menjadi implementasi BLE pada perangkat sensor dan gateway, implementasi MQTT sebagai media pengiriman data antara *gateway* dan *cloud*, dan implementasi RESTful HTTP pada *batch access* dan *websocket* pada *stream access*. Proses implementasi pada bab ini akan menyertakan pseudocode yang menjelaskan teknis alur kerja dari *middleware*.

BAB VI PENGUJIAN

Bab ini menjelaskan pengujian pada *middleware* yang dikembangkan. Pada penelitian ini pengujian dibagi menjadi dua. Pertama adalah pengujian fungsionalitas pada *middleware* yang menguji kesesuaian middleware terhadap kebutuhan fungsionalitas yang telah dibuat. Kedua adalah kinerja yang terbagi menjadi tiga pengujian kinerja. Pengujian *batch interface*, pengujian *stream interface*, dan pengujian *concurrent user.*

PENUTUP

Penarikan kesimpulan dari perancangan dan pembahasan dibahas pada bab ini. Selain itu juga terdapat saran yang ditulis oleh penulis sebagai catatan yang dapat digunakan peneliti selanjutnya dalam pengembangan lebih lanjut mengenai topik terkait.

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan pada penelitian ini digunakan sebagai referensi pengerjaan dalam proses penelitian. Sumber referensi pada penelitian ini diambil pada jurnal penelitian dan literatur lainnya yang berkaitan dengan konsep WoT, *middleware*, dan pengujian. Jurnal dan literatur yang diambil meliputi konsep, motivasi, dan tantangan pada pengembangan *middleware* dan penggunaan konsep WoT. Landasan kepustakaan tersebut menjadi acuan pada perancangan, implementasi, dan pengujian pada *middleware*

2.1 Kajian Pustaka

Kajian pustaka pada penelitian ini membahas penelitian sebelumnya yang berkaitan dengan *WoT middleware* dan permasalahan yang ada pada penelitian tersebut. Penelitian tentang pengembangan *WoT middleware* telah dilakukan pada penelitian berjudul Design of a *RESTful Middlewar*e to Enable a *Web of Medical things*. Penelitian tersebut mendesain sebuah arsitektur *middleware* menggunakan pendekatan WoT menggunakan REST sebagai media komunikasi akses data oleh aplikasi. Penelitian IoT yang bertema medis ini juga menyebutkan tantangan pada pengembangan *middleware* adalah heterogenitas, interoperabilitas, mobilitas, adaptabilitas, *user experience*, dan *security*. Pada aspek heterogenitas penelitian ini mengembangkan desain middleware yang dapat mengoperasikan lebih dari satu jenis perangkat kesehatan menggunakan protokol komunikasi *Bluetooth Classic*, BLE, dan 802.11. Namun middleware ini memiliki kekurangan pada penyediaan data *real time*.

Penelitian selanjutnya adalah pengembangan *middleware* berbasis publish *subscribe* (pub-sub) yang memungkinkan akses *real time* pada *constrained device* yang berjudul *A publish subscribe based middleware for enabling real time web access on constrained device*. *Middleware* pada penelitian ini menerapakan arsitektur pub-sub antara *client* dan *sensor node* menggunakan *websocket through* MQTT. *Client* maupun *sensor node* dapat bertindak sebagai *publisher* atau *subscriber*. *Middleware* ini berfokus pada bagaimana untuk menyediakan sebuah akses data secara *real time*.

Dari penelitian tersebut penulis membuat sebuah pengembangan *middleware*. Pengembangan berupa perancangan dan implementasi *middleware* yang dapat melakukan abstraksi perangkat sensor berbasis BLE yang memungkinkan data dari perangkat sensor untuk diakses pada sebuah API. Secara khusus implementasi juga bertujuan untuk mengembangkan rancangan sebelumnya dengan menyediakan *stream access interface* pada *middleware* untuk akses *real-time*.

Berikut adalah kajian pustaka yang digunakan sebagai referensi oleh penulis dalam penelitian ini:

No	Nama Penulis, Judul, Tahun	Persamaan	Perbedaan	
			Penelitian terdahulu	**Rencana Penelitian**
1	Nada Philip, Talal Butt, Drishty Sobnath, Reem Kayali, Shereen Nabhani, Barbara Pierscionek, Ioanna Chouvarda, Vassilis Kilintis, Pantelis Natsiavas, N. Maglaveras, Andreas Raptopoulos, "*Design of a RESTful Middleware to Enable Web of Medical Things*", 2015.	*Middleware* Menggunakan pendekatan *web of things* dan menerapkan *webservice* API berbasis RESTful	*Middleware* menyediakan interface RESTful pada aplikasi dan HTTP pada *cloud synchronizatio n*.	Middleware menyediakan RESTful API dan menambahkan MQTT over *websocket* interface serta menerapkan *lightweight* MQTT *messaging* pada *cloud synchronizatio n* untuk menggantikan sinkronisasi HTTP
2	Adhitya Bhawiyuga, Dany Primanita Kartikasari, Eko Sakti Pramukantoro, "*A publish subscribe based middleware for enabling real time web access on constrained device*", 2017	*Middleware* yang menerapkan *websocket* untuk akses data secara r*eal-time*	*Middleware* menyediakan akses data secara *real-time* menggunakan MQTT T*hrough websock*et	*Middleware* menerapkan *gateway* untuk mengelola perangkat sensor berbasis BLE dengan abstraksi *web of things* untuk menghubungk an perangkat dengan antarmuka.
3	Smita Kumari, Santanu Kumar Rath, "*Performance comparison of SOAP dan REST based*	Melakukan perbandingan *webservice access*	Melakukan pengujian kinerja pada REST	Melakukan pengujian kinerja pada REST

	Web Services for Enterprise Application Integration"	protokol SOAP dan REST	*webservice* dengan 28 *request* secara asinkron pada file berukuran 1000kb hingga 7000kb	*webservice* dan MQTT dengan 100, 500, 1000, dan 1500 koneksi secara asinkron sebanyak 5 percobaan.

2.2 Dasar Teori

Dalam menunjang kebenaran penelitian, penulis mengambil dasar teori dari jurnal dan penelitian sebelumnya sebagai landasan. Berikut beberapa teori dasar yang penulis gunakan dalam penelitian ini.

2.2.1 Internet of Things

Tumbuhnya jumlah objek fisik yang terhubung ke internet menjadi awal dasar terealisasinya konsep IoT. Peningkatan jumlah objek fisik yang terhubung dengan internet ini terjadi ketika konsep IoT merambat pada bidang lain seperti transportasi, kesehatan, otomasi industri dan sebagainya. Konsep IoT memungkinkan objek fisik untuk berkomunikasi dan bekerja dengan objek fisik lainnya dengan membuat objek tersebut dapat "saling berbicara" untuk berbagi informasi dari kondisi yang diterima oleh objek lainnya (Al-Fuqaha, 2015).

Survey penelitian menjelaskan terdapat empat arsitektur IoT yang ada yaitu *Three-laye*r, *Middle-ware based*, *SOA based* dan *Five-layer* (Al-Fuqaha, 2015). *Three-layer* merupakan model arsitektur dasar yang terdiri dari lapisan *application*, *network*, *dan perception*, pada literatur terbaru sebuah model baru dibuat untuk menambahkan abstraksi pada arsitektur IoT yaitu *Five-Laye*r. Arsitektur *Five-La*yer memiliki lima lapisan yaitu *perception*, *network*, *Middleware*, *application* dan *business layer* (Khan, 2012).

Penulis akan mengkaji tiga lapisan utama yaitu *perception*, *network* dan *middleware layer*. *Perception layer* merupakan lapisan yang terdiri dari perangkat sensor, lapisan ini menangani metode identifikasi dan pengumpulan informasi perangkat berupa data yang tergantung dari sensor yang digunakan. *Network layer* merupakan lapisan yang mengatur media pengiriman data sensor, teknologi dari media pengiriman dapat berupa Wi-Fi, Zigbee, Bluetooth dsb. *Middleware* layer adalah lapisan yang menangani manajemen layanan, lapisan ini akan menerima data dari lapisan *network* dan menyimpannya ke *database* (Khan R, 2012).

2.2.2 IoT Middleware

Dapat dipahami bahwa IoT dapat memberikan sebuah solusi dan potensi yang baik di berbagai bidang, namun heterogenitas teknologi, perangkat, dan aplikasi pada IoT membawa tantangan pada pengembangan aplikasi. Dalam hal ini *middleware* dapat menjadi solusi tepat untuk mempermudah pengembangan aplikasi dari tantangan heterogenitas yang terjadi.

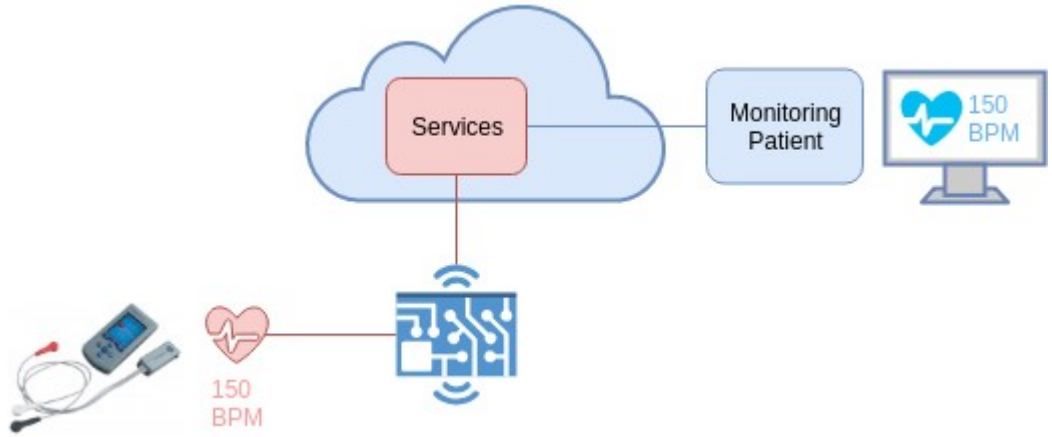
Middleware sendiri merupakan sebuah *layer* yang menghubungkan bagian berbeda antara perangkat dan aplikasi, sebagaimana keduanya memiliki teknologi dan cara interaksi yang berbeda. *Middleware* secara umum bertujuan untuk mengabstraksikan kompleksitas sistem atau perangkat keras yang memudahkan pengembangan aplikasi tanpa harus terganggu oleh sistem dan perangkat keras yang digunakan (Neely S, 2006). Pada sudut pandang komputasi, *middleware* ialah yang menyediakan lapisan bagi perangkat lunak aplikasi dan sistem.

Middleware dalam sudut pandang IoT bertumpu pada kemampuan *middleware* untuk menyediakan layanan dan dukungan pada arsitektur. Kedua hal ini merupakan bagian penting yang harus dipenuhi oleh sebuah *middleware*. Penelitian M. A. Razzaque (2016) menjelaskan kebutuhan layanan fungsional yang harus dipenuhi oleh *middleware* adalah *resource discovery*, *resource*, *data*, *event*, dan *code management*. Secara arsitektur middleware harus mendukung *programming abstraction* dengan menyediakan API yang memudahkan pengembangan aplikasi. Dukungan interoperabilitas yakni kemampuan *middleware* untuk dapat berfungsi pada perangkat, teknologi, dan aplikasi yang berbeda tanpa menambahkan aplikasi dalam mencapai hal tersebut.

2.2.3 Web of Things

Konsep WoT hadir pada penelitian Kindberg (2002) yang dahulu hendak menghubungkan objek fisik dengan halaman *web*. Menggunakan *interface* inframerah atau *barcode* pada objek, *user* dapat mengambil URI (Uniform Resource Identifier) halaman *web* dengan melakukan interaksi pada objek. Konsep ini dikembangkan pada penelitian Guinard dengan menggabungkan *smart things* menjadi arsitektur *heavyweight web* yang tersandarisasi seperti SOAP, WSDL (Guinard, 2010). Arsitektur ini dinilai terlalu berat untuk objek sederhana, sehingga pada pengembangan berikutnya digunakan HTTP dengan teknologi web 2.0. Dalam Konsep WoT *smart things* dan layanan pada *web server* dihubungkan dalam arsitektur REST untuk melakukan abstraksi pada *physical world*.

Pengaplikasian WoT tidak memiliki batasan untuk digunakan pada bidang atau domain apapun, namun saat ini terdapat beberapa bidang yang dapat menggunakan WoT. Jika di spesifikan maka bidang seperti perkebunan, konsumen *smart home*, konstruksi, industri dan kesehatan dapat menjadi beberapa bidang yang dapat menerapkan WoT. Ilustrasi WoT bidang kesehatan ditunjukkan pada gambar 2.1.



Gambar 2.1 Ilustrasi WoT di bidang kesehatan

World Wide Web Consortium (W3C) menjelaskan arsitektur WoT sebagai berikut. Secara konseptual sistem WoT adalah sebuah objek yang berkomunikasi melalui jaringan dan masing-masing objek memiliki peran yang berbeda, berikut adalah peran tersebut:

- WoT *Client*: Objek yang bertingkah sebagai klien, komponen aktif yang melakukan permintaan pada fungsionalitas yang terdapat pada objek lain.
- WoT *Server*: Objek yang bertingkah sebagai *server*, komponen yang bereaksi terhadap suatu permintaan dari objek lain. Permintaan tersebut akan mengubah *state* dari objek *server*, interaksi ini dilakukan untuk mengekspos objek tersebut.
- WoT Servient: Objek yang dapat bertingkah menjadi keduanya, bisa bereaksi pada permintaan dan melakukan permintaan pada fungsionalitas pada objek lain.

Terdapat dua metode komunikasi dalam arsitektur WoT. Komunikasi secara langsung dimana *client* dan *server* berinteraksi dengan protokol jaringan yang sama dan dapat saling mengakses satu sama lain. Metode terakhir adalah komunikasi secara tidak langsung dimana *client* dan *server* berinteraksi dengan protokol jaringan yang berbeda melalui sebuah *proxy/gateway*.

2.2.4 Smart Healthcare

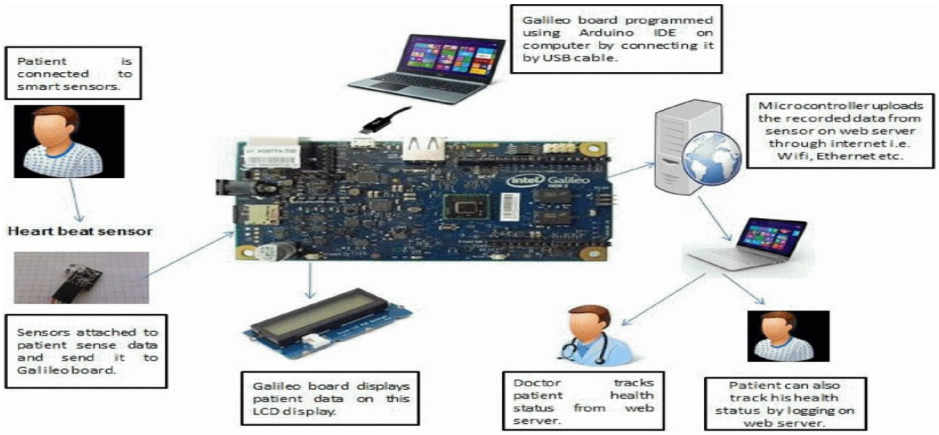
IoT berperan penting pada bidang kesehatan dengan menyediakan kemudahan bagi dokter untuk memonitor dan memantau keadaan pasien. Perpaduan konsep IoT di bidang kesehatan menghasilkan sebuah konsep sistem yang disebut *Smart Healthcare*.

Penelitian oleh G Punit et al (2016) mereferensi sebuah penelitian dan pekerjaan IoT di bidang *healthcare* sebelumnya, terdapat dua produk yang saat itu menjadi pekerjaan yang terkait pada s*mart healthcare* yang ia teliti yaitu:

1. Cooey *Smart Health*, produk ini merekam data medis melalui perangkat *bluetooth*. Produk ini dapat memberi tips kesehatan dan peringatan

kepada pasien terhadap resiko kesehatan yang mungkin dialami. Informasi yang diberi oleh produk ini didapat dari data medis yang telah direkam, dianalisis dan di bagiakn oleh perangkat kepada pasien.

2. Health Vault by Microsoft, produk ini memiliki mekanisme kerja yang sama dengan s*mart healthcare* pada umumnya, namun ia dilengkapi dengan fitur tambahan. Beberapa fitur tersebut adalah autentikasi dengan layanan akun, otorisasi pengguna yang digunakan sebelum berbagi data medis, *user control* dengan mengotorisasi pembagian data, dan data *provenance* dengan memperlakukan data berdasarkan kecerdasan.



Gambar 2.2 Model Arsitektur penelitian *Smart Healthcare* (G Punit, 2016)

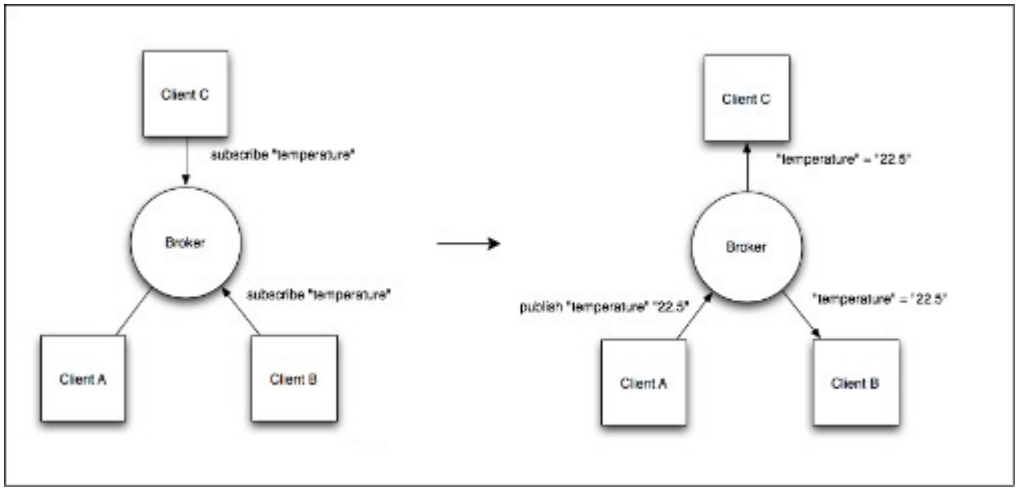
Penelitian G Punit sendiri bertujuan membuat sebuah model smart healthcare pada gambar diatas. Model tersebut terdiri dari banyak sensor yang menghasilkan data mentah. Data mentah tersebut selanjutnya disimpan ke *database* melalui Intel Galileo *Board*, dimana data tersebut selanjutnya di analisis oleh dokter dan pasien melalui sebuah akses pada *webserver*.

2.2.5 Message Query Telemetry Transport (MQTT)

MQTT adalah sebuah protokol pesan berbasis *publish-subscribe* yang di desain untuk komunikasi *machine-to-machine* (M2M). MQTT pertama dikenalkan pada tahun 1999 oleh Andy Stanford-Clark dan Arlen Nipper, kemudian dikembangkan oleh IBM dan menjadi protokol bersifat *open source*.

2.2.5.1 Arsitektur

MQTT memiliki model *client-server*, dimana *client* dapat melakukan *subscribe* atau *publish* pada sebuah topik. Keduanya terhubung dengan *server* yang dikenal sebagai *broker*. Seperti yang dijelaskan bahwa MQTT berbasis pesan dimana tiap pesan di *publish* ke sebuah alamat yang disebut *topic*. *Client* kemudian melakukan *subscribe* pada *topic* yang sudah ditentukan dan menerima setiap pesan yang di *publish* pada *topic* tersebut, seperti pada gambar dibawah ini:



Gambar 2.3 Model Arsitektur MQTT (Eclipse Foundation, 2014)

Gambar diatas menunjukkan terdapat sebuah 3 *device client* A sebagai *publisher*, B dan C sebagai *subscriber*. *Device* B dan C *subscribe* pada *topik* temperature kemudian menerima data ketika device A *publish* sebuah pesan pada topik tersebut. *Broker* bertindak sebagai *server* yang meneruskan pesan tersebut kepada c*lient* yang melakukan *subscribe*.

Pada penggunaan MQTT terdapat sebuah istilah *topic* seperti yang dijelaskan diatas, *topic* sendiri digunakan untuk memfilter pesan kepada *broker* baik dari *publisher* maupun *subscriber* sehingga kedua mendapat topik yang diinginkan. *Topic* sendiri memiliki bentuk hirarki sehingga terdapat *sub-topic* seperti contoh *topic* `sensor/temperature/1` dimana sensor adalah *topic*, temperature dan 1 merupakan *sub-topic*. MQTT juga mengenal istilah *wildcard* dengan menggunakan tanda + dan #, dimana + berarti mengganti satu level topik contoh *client subscribe* pada topic `sensor/+/1` maka *topic* tersebut sama dengan `sensor/temperature/1` dan `sensor/humidity/1`, tapi tidak sama dengan `sensor/temperature/room/1`. *Wildcard* # berarti mengganti *multilevel topic* contoh *client* subscribe pada *topic* `sensor/#` hal tersebut sama dengan *subscriber* pada `sensor/temperature/1` dan `sensor/temperature/room/1`.

2.2.5.2 Quality of Service (QoS)

Terdapat sebuah QoS pada MQTT dalam menjamin pengiriman pesan, tiga macam QoS yang ada pada MQTT adalah QoS level 0,1 dan 2. Tiga level QoS itu disebut:

1. **QoS 0 (*at most once*)**

   Tingkat minimal pada QoS MQTT tidak ada jaminan pesan akan terkirim, sehingga *subscriber* yang menerima pesan dengan QoS 0 tidak akan mengirim ACK dan *publisher* tidak akan mengirim ulang pesan jika pesan tersebut tidak diterima oleh *subscriber*. QoS 0 digunakan ketika koneksi antara pengirim dan penerima sangat stabil, dan ketika tidak mempermasalahkan beberapa pesan yang hilang.

2. **QoS 1 (*at least once*)**

   QOS ditingkat ini menjaminan bahwa pesan akan terkirim setidaknya satu kali ke penerima. Pengirim akan menyimpan pesan yang dikirim hingga menerima sebuah PUBACK dari penerima yang mengkonfirmasi bahwa pesan diterima dan ada kemungkinan duplikasi pesan. QoS 1 digunakan dalam kasus pesan yang harus diterima dan duplikasi dapat diatasi, juga ketika tidak bisa menangani overhead yang dihasilkan oleh QoS 2 .

3. **QoS 2 (*exactly one*)**

   QoS 2 adalah tingkat maksimal QoS pada MQTT. Pada tingkat ini QoS menjamin bahwa setiap pesan pasti diterima sekali tanpa terjadi duplikasi pesan, hal ini menjadikan QoS 2 adalah QoS paling aman dan lambat. QoS 2 digunakan ketika duplikasi dan kehilangan pesan merupakan hal yang sangat kritis untuk dihindari. QoS 2 dapat mengakibatkan *overhead* dikarenakan interaksi yang sangat lama untuk diselesaikan.

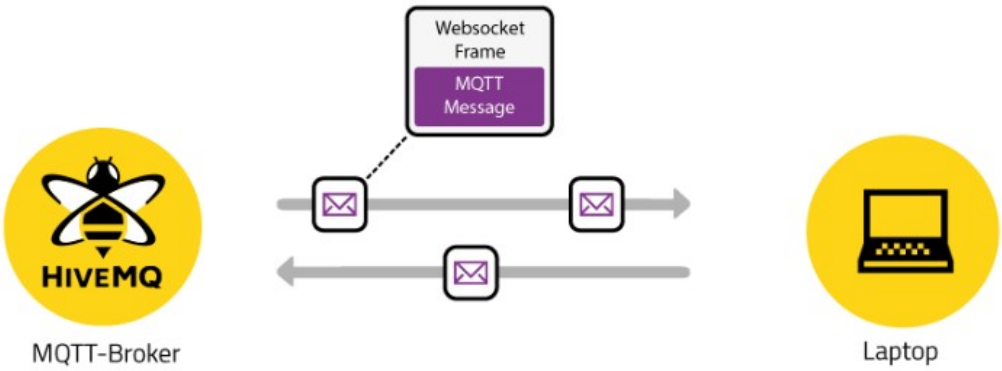
2.2.5.3 MQTT Over Websocket

Mengacu pada salah satu bagian dari penelitian yang akan dibuat penulis yaitu *real-time* interface sebagai akses bagi aplikasi yang membutuhkan *stream data processing*. Implementasi akan menggunakan MQTT *Over Websocket*. Tujuan dari MQTT *over websocket* adalah untuk mengirim dan menerima data MQTT dari *browser* secara stream, MQTT *over* w*ebsocket* di implementasikan pada stream interface penelitian penulis. MQTT Over Websocket akan dikaji di bagian ini.

MQTT *over websocket* bertujuan agar *browser* dapat memanfaatkan fitur yang ada pada MQTT, memberi kapabilitas kepada *browser* dalam beberapa kasus untuk:

- Menampilkan *live information* dari sebuah perangkat atau sensor.
- Menerima *push notification* melalui MQTT.
- Melihat kondisi terkini dari perangkat.
- Berkomunikasi secara efisien dengan aplikasi berbasis *mobile web*.

MQTT *Over Websocket* dapat digunakan oleh setiap *browser* yang mendukung *websocket*, yang dibutuhkan untuk berinteraksi dengan MQTT *over websocket* adalah sebuah javascript *library*, atau *client* yang menggunakan javascript.



Gambar 2.4 Model MQTT *Over Websocket* (HiveMQ, 2015)

Websocket adalah protokol jaringan yang menyediakan komunikasi dua arah antara *browser* dan *webserver*. Mirip seperti MQTT, protokol *websocket* berbasis TCP. MQTT *over websocket* bekerja dengan cara mengirim pesan MQTT ke jaringan dengan mengenkapsulasi paket tersebut dengan *websocket frame*. Pesan MQTT tersebut kemudian diterima oleh *broker* yang telah di konfigurasi secara native pada aplikasi MQTT *broker*. Alasan menggunakan MQTT *over websocket* adalah browser tidak mampu untuk berinteraksi secara langsung dengan MQTT yang merupakan koneksi TCP yang mentah.

2.2.6 *Representational State Transfer* (REST)

REST adalah sebuah architectural style pada sebuah perangkat lunak dalam mendesain sebuah *webservices*. Tujuan penggunaan REST pada w*ebservice* adalah untuk mendefinisikan aturan dalam mengakses *web resource* yang di alamatkan dan di kirimkan dengan HTTP kepada *webservice*. REST pertama dikenalkan oleh Roy Fielding di tahun 2000 dalam disertasinya yang berjudul "*Architectural Style and the Design of Network-based Software Architectures*".

2.2.6.1 HTTP Method dan URI

Karakteristik dari *webservice* yang menggunakan gaya arsitektur REST adalah *webservice* tersebut menggunakan HTTP *method* dalam mengakses *resource* pada *web* tersebut. Contohnya HTTP GET, merupakan method HTTP yang didefinisikan untuk mengambil sebuah *resource* dari *webservice*, mengambil data atau mengeksekusi sebuah kueri pada *webserver* oleh aplikasi *client*.

REST memiliki empat operasi dasar yaitu *Create*, *Read*, *Update*, dan *Delete* (CRUD) juga HTTP Method. Berdasarkan ke empat operasi tersebut, berikut adalah relasinya dengan HTTP Method:

1. POST digunakan untuk membuat sebuah *resource* pada *server*
2. GET digunakan untuk mengambil sebuah *resource* atau data
3. PUT digunakan untuk mengubah *state* atau kondisi dari *resource*
4. DELETE digunakan untuk menghapus *resource*

URI adalah sebuah *string* yang digunakan untuk mengidentifikasi lokasi sebuah resource, pada arsitektur REST URI berhubungan langsung dengan HTTP *method*. HTTP *method* berperan untuk menentukan operasi yang akan dilakukan terhadap *web resource* yang ditunjukkan oleh URI, berikut adalah contoh hubungan keduanya. URI akan digunakan untuk menunjuk resource sebuah *web*, contoh `http://www.layanan.org/nama/pelajar`. Operasi GET dengan URI tersebut akan mengambil data dari *root* nama dan *node* pelajar. Operasi POST dengan URI tersebut digunakan untuk membuat data baru pada *node* pelajar, dan selanjutnya dengan HTTP *method* yang lainnya.

2.2.7 Bluetooth Low Energy

Pada tahun 2010 dikembangkan *bluetooth* versi 4.0 sebagai versi yang memiliki dua jenis *Bluetooth* yakni *low energy* dan *classic*.

Device	BR/EDR (classic Bluetooth) support	BLE (Bluetooth Low Energy) support
Pre-4.0 Bluetooth	Yes	No
4.x Single-Mode (Bluetooth Smart)	No	Yes
4.x Dual-Mode (Bluetooth Smart Ready)	Yes	Yes

Gambar 2.5 Spesifikasi Bluetooth 4.0 (Townsend et al, 2016)

Gambar 2.5 menjelaskan teknologi yang terdapat pada *Bluetooth* 4.0. Terdapat BR/EDR yang merupakan *classic bluetooth* yang telah ada sejak versi 1.0 dan BLE (*Bluetooth Low Energy*) yang merupakan *Bluetooth* dengan standar *low energy*. *Classic bluetooth* didukung pada *dual-mode* dan pre-4.0 *bluetooth*, sedangkan BLE didukung pada *single* dan *dual-mode* yang biasa disebut *bluetooth smart*.

Modulasi pada radio BLE pada spesifikasi secara teori adalah 1 Mbps, pada penggunaan biasanya sebesar 5-10KB per detik tergantung dari batasan perangkat yang digunakan. BLE diperuntukkan pada komunikasi jarak pendek, meskipun dapat di konfigurasi untuk pengiriman data sejauh 30M tapi umumnya hanya digunakan dengan jarak 2 sampai 5 meter.

2.2.7.1 Mekanisme Komunikasi BLE

Komunikasi pada BLE ditangani oleh *Generic Access Profile* (GAP) mekanisme tersebut dibagi dua yaitu *broadcasting atau connections* masing-masing mekanisme memiliki penggunaan, kelebihan dan kekurangan dengan sebagai berikut:

- *Broadcasting*: terdapat dua peran pada mekanisme ini yaitu *broadcaster* dan *obeserver*. *Broadcaster* akan mengirimkan data ke perangkat *observer* dalam jangkauan yang melakukan *listening*. *Broadcasting* adalah cara mengirimkan data dari satu sumber ke banyak penerima tanpa membuat sebuah koneksi antara sumber dan penerima. Mekanisme ini diperuntukkan pada pengiriman data ke lebih dari satu penerima.

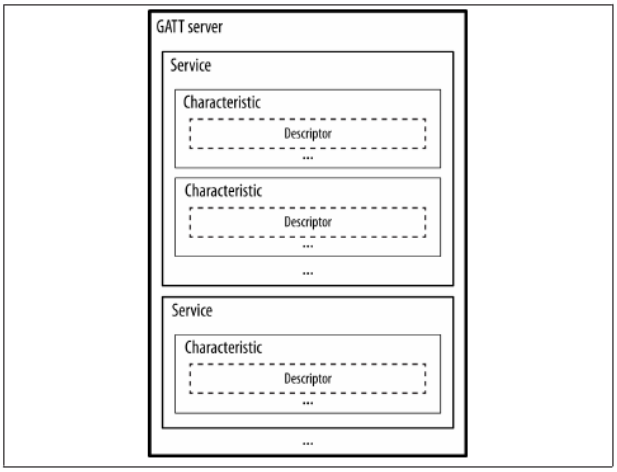
- *Connections*: dua peran pada *connection* adalah *central* (*master*) dan *peripheral* (*slave*). Central berperan untuk mencari *advertising packet* dari *peripheral* dan menginisiasi sebuah koneksi, sedangkan *peripheral* berperan untuk mengirim *advertise packet* dan menerima koneksi dari *central* lalu saling bertukar data.

Perbedaan terbesar antara *connections* dan *broadcasting* adalah jumlah aktor dalam sebuah pertukaran data.

2.2.7.2 Mekanisme Pertukaran Data BLE

Generic Attribute Profile (GATT) merupakan sebuah *framework* yang membuat *bluetooth* pada perangkat yang berbeda dapat saling terhubung. GATT menangani pertukaran datanya dengan melakukan pertukaran *profile* dan *user* data diatas koneksi BLE menggunakan *Attribute Protocol*. Terdapat beberapa hal harus ada pada GATT untuk melakukan pertukaran data antara lain:

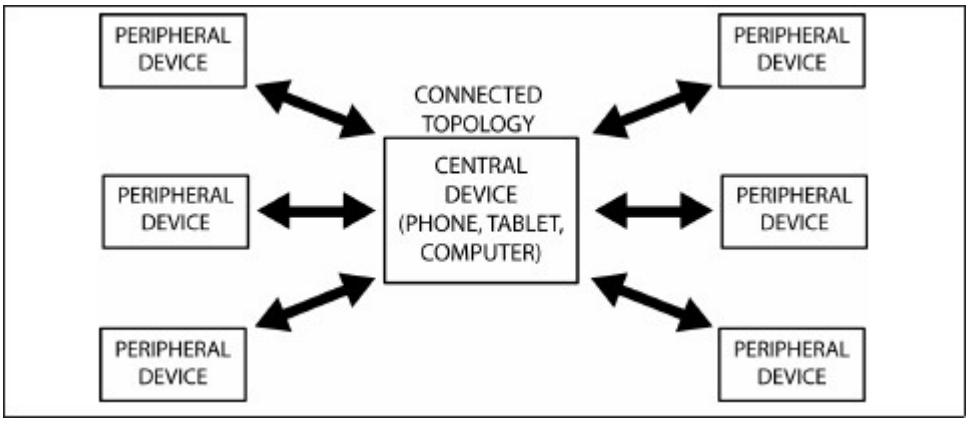
1. *Role* antara *device*, terdapat dua *role* pada GATT yaitu *client* dan *server* dimana *server* berperan untuk menerima *request* dari *client* dan memproduksi data untuk *client*. *Client* berperan untuk mengirim *request* dan menerima respon dari *server*
2. UUID, *Universally unique identifier* merupakan sebuah atribut unik untuk membedakan identitas sebuah *service*, *characteristic* dan *descriptor*
3. *Attribute*, bentuk entitas data yang terdapat pada role *server* dan diakses oleh *client* dengan ketentuan hirarki yang sudah berlaku. Terdapat beberapa *attribute* yang didefinisikan pada GATT. *Handle* yang merupakan *identifier server* untuk dapat diakses, type merupakan *identifier service* dan *characteristic server*, *permission* yang merupakan batasan akses client pada atribut dan *value* dari atribut tersebut.



Gambar 2.6 Hirarki GATT Server (Townsend et al, 2016)

Setiap *server* memiliki *service* yang bisa terdiri dari nol atau lebih *characteristic*, begitu juga dengan *characteristic* yang bisa terdiri dari nol atau lebih *descriptor*. Ketiga atribut tersebut memiliki peran yang berbeda yaitu:

1. *Service*, mendefinisikan layanan yang ada pada *server*.
2. *Characteristic*, menampung data user yang terbagi menjadi dua atribut yaitu *characteristic declaration* yang berisi informasi data user dan *characteristic value* yang merupakan data *user* dalam *kolomvalue*.
3. *Descriptor*, menampung informasi dan *value* tambahan dari *characteristic*.



Gambar 2.7 Contoh Hirarki GATT Heart Rate Service (Townsend et al, 2016)

Gambar 2.7 menunjukkan topologi mekanisme pertukaran data, seperti yang dijelaskan pada mekanisme komunikasi jenis *connections* dimana terdapat dua perangkat yang disebut *peripheral* dan *central* atau *client* dan *server*. Pada mekanisme ini *peripheral device* bekerja dengan *mengadvertise* data dan menerima koneksi. Ketika *central device* hendak mengambil data maka akan diinisiasi koneksi antara *device* tersebut. Setelah keduanya terhubung maka *central device* akan mendapatkan data yang di *advertise* oleh *peripheral device*. Pada kasus ini *central device* dapat terhubung dengan banyak *peripheral device*, sehingga memungkinkan untuk menjadikan *central device* sebagai *gateway* perangkat.

2.2.8 Pengujian Fungsional

Pengujian fungsional dilakukan untuk menguji kesesuaian fungsionalitas antara *middleware* dan kebutuhan fungsionalitas yang dibuat pada analisis kebutuhan. Hasil pengujian ini akan menjadi pertimbangan dalam mengidentifikasi kesesuaian dan adanya kesalahan pada perancangan dan implementasi pada *middleware*. Pengujian fungsional dilakukan dengan menguji mekanisme kerja komponen yang terdapat pada *middleware*, baik dari komunikasi antar komponen ataupun hasil yang diberikan oleh *middleware*.

2.2.9 Pengujian Kinerja

Pengujian kinerja didefinisikan sebagai pengujian untuk memastikan sebuah perangkat lunak dapat bekerja dengan baik pada beban kerja yang telah ditentukan. *Middleware* merupakan perangkat lunak yang harus dipastikan dapat berjalan dengan baik sesuai perannya dan kebutuhannya. *Middleware* berfungsi untuk memungkinkan perangkat sensor dan aplikasi *client* untuk saling

terhubung dan melakukan pertukaran data. Pertukaran data memiliki sebuah ukuran yang dapat diperhitungkan seperti, berapa lama data tersebut dapat diterima, sebanyak apa *middleware* dapat menangani permintaan client untuk bertukar data. Pengujian kinerja akan mengukur kinerja yang dimiliki oleh middleware dalam melakukan fungsinya pada satuan metrik *throughput*, *latency*, *concurrent connection*, dan *end-to-end delay*.

a. *Throughput*

   Throughput adalah jumlah request atau besaran data per detik. Throughput dihitung berdasarkan jumlah request dibagi waktu keseluruhan yang digunakan untuk menyelesaikan request (Kumari & Rath, 2015). Pada Penelitian ini dilakukan pengujian throughput untuk mengukur jumlah request yang dapat ditangani oleh middleware.

b. *Concurrent* connection

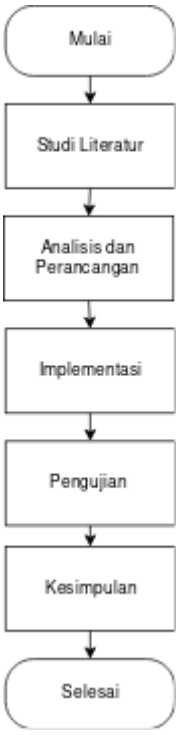
   Concurrent connection adalah jumlah koneksi yang dapat ditangani dalam satu waktu pada protokol TCP. Pengujian concurrent connection dilakukan untuk mengukur kemampuan middleware dalam menangani banyak koneksi dalam waktu bersamaan. Pada penelitian ini pengujian dilakukan pada request publish dan subscribe oleh client dan aplikasi.

c. *End-to-end delay*

   Delay adalah parameter yang terkandung pada komunikasi, dimana pada setiap komunikasi selalu ada delay. Nilai waktu dipengaruhi oleh kualitas jaringan. Delay dapat dihitung secara *one-way* yakni jumlah waktu yang berlalu dari paket dikirim hingga sampai pada tujuan, atau *round-trip* dari sumber ke tujuan hingga kembali ke sumber lagi (Carvalho & Magedanz, 2009). Pada penelitian ini pengujian delay dilakukan untuk mengukur waktu yang dibutuhkan pada pengiriman data dari *sensor device* hingga *client app*.

BAB 3 METODOLOGI PENELITIAN

Pembahasan tahapan-tahapan yang dilakukan dalam penelitian digambarkan melalui metodologi penelitian ini. Alur tahapan penelitian mulai dari studi literatur, perancangan, implementasi, pengujian, dan kesimpulan digambarkan pada gambar 3.1.



Gambar 3.1 Alur Metodologi Penelitian

3.1 Studi Literatur

Studi literatur pada penelitian ini berisi penjelasan mengenai teori dasar dan penelitian terdahulu terkait dengan objek penelitian. Sumber studi literatur berasal dari jurnal, situs resmi, dan berbagai penelitian sebelumnya. Tahap ini dilakukan sebagai bahan bagi penulis dalam mendukung dan menyelesaikan masalah agar tujuan penelitian tercapai. Studi literatur juga dibuat untuk menggambarkan objek penelitian dan menguraikan penelitian-penelitian terdahulu sesuai dengan objekterkait. Studi literatur pada penelitian ini membahas teori seputar *Web of Things*, *Middleware*, MQTT, *Websocket*, RESTful HTTP, *Bluetooth Low Energy*, dan pengujian kinerja.

3.2 Analisis Kebutuhan dan Perancangan

Terdapat perencanaan dalam implementasi *middleware* berupa analisis yang meliputi kebutuhan fungsional, kebutuhan data dan kebutuhan non-fungsional. Kebutuhan yang dihasilkan dari analisis kemudian dijadikan sebagai acuan dalam

perancangan *middleware* yang meliputi perancangan arsitektur, aliran data, struktur data, perangkat keras dan perangkat lunak.

3.2.1 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional dilakukan untuk menggali fungsionalitas yang harus terpenuhi dan proses-proses yang dapat dilakukan oleh *middleware* yang diimplementasikan. Hasil dari analisis kebutuhan fungsional akan ditunjukkan pada sebuah tabel yang berisi kode, nama dan deskripsi dari fungsi yang sesuai dengan kebutuhan *middleware*. Keluaran dari analisis kebutuhan fungsionalitas akan dijadikan sebagai pertimbangan dalam perancangan arsitektur, perangkat lunak pada bab 4 dan implementasi dari *middleware* pada bab 5.

3.2.2 Analisis Kebutuhan Data

Analisis kebutuhan data dilakukan untuk menggali informasi terkait data yang dibutuhkan untuk diterima oleh aplikasi melalui *middleware*, hasil analisis akan ditunjukkan pada sebuah tabel yang berisi kode, nama dan deskripsi data. Keluaran dari analisis kebutuhan data akan dijadikan sebagai acuan dalam perancangan struktur data dan perancangan aliran data pada Bab 4.

3.2.3 Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional berisi pengalian kebutuhan non fungsional yang meliputi kebutuhan perangkat keras dan perangkat lunak dari yang dibutuhkan untuk menunjang operasional dari *middleware*. Hasil analisis akan ditunjukan dalam sebuah tabel yang berisi kode, nama dan deskripsi dari perangkat keras dan perangkat lunak. Keluaran dari analisis kebutuhan non fungsional akan dijadikan sebagai acuan dalam perancangan perangkat keras, perangkat lunak pada bab 4 dan implementasi dari *middleware* pada bab 5.

3.2.4 Perancangan Arsitektur

Sub bab Perancangan arsitektur merupakan gambaran umum dari arsitektur *middleware*. Arsitektur *middleware* sendiri terdiri dari tiga komponen yaitu *sensor-to-cloud gateway*, *messaging service* dan *data access interface*. Untuk menjelaskan proses dari cara kerja *middleware*, maka arsitektur dilengkapi dengan komponen eksternal yakni perangkat sensor sebagai penghasil data dan *client* sebagai pengakses *data access interface*.

3.2.5 Perancangan Aliran Data

Setelah menjelaskan arsitektur selanjutnya sub-bab perancangan aliran data akan menjelaskan proses abstraksi, dari data dihasilkan oleh perangkat sensor dan diterima oleh *gateway* hingga diakses oleh *client-app* melalui *interface*. Proses abstraksi merupakan proses perubahan data pada *middleware* dari bentuk

yang belum dapat diakses hingga dapat diakses oleh aplikasi, proses perubahan inilah yang kemudian disebut sebagai proses abstraksi. Perancangan aliran data dibuat dalam bentuk sequence diagram untuk menjelaskan tahapan yang dilalui data dan fungsi yang dijalankan oleh *middleware* selama proses abstraksi berlangsung.

3.2.6 Perancangan Struktur Data

Perancangan struktur data dibuat untuk merepresentasikan identitas sensor, perangkat dan *gateway*. Identitas dari perangkat, sensor dan *gateway* akan direpresentasikan menggunakan struktur yang terdapat didalam data yaitu *topic*, URI, dan *payload*. Tujuan dari representasi identitas perangkat pada perancangan struktur data adalah untuk mempermudah proses identifikasi sumber dari data yang diterima oleh *client app*.

3.2.7 Perancangan Perangkat Keras

Perancangan perangkat keras menjelaskan perangkat keras dan protokol komunikasi yang digunakan beserta layanan-layanan yang berjalan pada *middleware* dalam melakukan abstraksi. Berbeda dengan arsitektur dan alur data, dalam sisi perangkat keras *middleware* dibagi menjadi dua komponen yaitu *sensor-to-cloud gateway* dan *cloud*. Selain menjelaskan perangkat keras dari *middleware*, dijelaskan juga perangkat keras, protokol komunikasi yang digunakan oleh perangkat sensor beserta layanan yang digunakan untuk menghasilkan dan mengirimkan data.

3.2.8 Perancangan Perangkat Lunak

Perancangan perangkat lunak menjelaskan secara spesifik perangkat lunak dan protokol komunikasi yang ada pada perangkat sensor, *middleware* dan *client app* dalam proses abstraksi. Perancangan perangkat lunak akan menjelaskan proses abstraksi secara spesifik melalui hubungan perangkat lunak, protokol dan layanan. Perancangan perangkat lunak akan dibuat menggunakan diagram alir yang menjelaskan proses dan tahapan yang dilalui dari masing-masing komponen.

3.3 Implementasi

Implementasi berisi proses pengembangan *middleware* dari perancangan menjadi sebuah sistem dengan implementasi yang dilakukan. Dalam penelitian ini langkah awal yang dilakukan adalah membangun sebuah abstraksi dengan membuat alur komunikasi antara perangkat sensor yang menggunakan BLE pada ESP32 dengan gateway yang di buat pada Raspberry Pi. Langkah selanjutnya yaitu membangun komunikasi antara *gateway* dengan *cloud* menggunakan protokol MQTT dengan *library* di python. Langkah berikutnya adalah dengan membangun

sebuah *data access interface* dengan dua jenis akses yaitu stream menggunakan websocket dan batch menggunakan *webservice* RESTful HTTP dengan Flask.

3.4 Pengujian

Tahap ini adalah pengujian untuk mengukur kesesuaian *Middleware* baik pada kebutuhan fungsional, kinerja dalam mengirimkan data pada ukuran dan delay yang ditentukan. Pengujian fungsional dilakukan untuk memastikan kesesuaian *middleware* dengan kebutuhan fungsional yang telah dirancang. Fungsional yang diuji yakni kesesuaian pengiriman, penyimpanan dan akses data. Ketika proses pekerjaan pada *middleware* tidak berjalan sesuai kebutuhan yang dibuat maka selanjutnya akan ditinjau kembali, diperbaiki dan diujikan kembali.

Untuk menguji kinerja *middleware* dibuat pengujian kinerja melalui pengumpulan dan penghitungan data yang menghasilkan kesimpulan terhadap kinerja *middleware*. Terdapat tiga jenis pengujian kinerja yaitu pengujian *batch interface*, pengujian *stream interface*, dan *concurrent user*. Hasil dari pengukuran ini nantinya akan ditampilkan sebagai bahan pertimbangan dalam mengambil kesimpulan pada implementasi *middleware* untuk menjawab rumusan masalah yang telah ditetapkan. Pengujian diawali dengan perancangan lingkungan uji. Kemudian diikuti oleh perancangan skenario, pengujian, dan pemaparan hasil pengujian.

3.5 Penutup

Tahap kesimpulan dilakukan setelah semua tahapan telah berjalan sesuai dengan susunan metodologi. Tahap ini bertujuan untuk menjawab rumusan masalah pada penelitian, memastikan kesesuaian implementasi *middleware* dengan fungsionalitas yang ditetapkan, dan pengambilan kesimpulan pada hasil dari pengujian yang telah dihasilkan. Bagian akhir dari penutup adalah saran dan masukkan terhadap implementasi *middleware* baik dari sisi penulisan dan dokumentasi ataupun sisi penerapan dan implementasi yang dilakukan sebagai acuan penelitian selanjutnya.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

Analisis kebutuhan dan perancangan menjelaskan deskripsi umum, proses analisis kebutuhan serta perancangan dari *middleware* yang hendak diimplementasikan. Bab ini diawali dengan menjelaskan deskripsi umum *middleware* dan kebutuhan yang harus dipenuhi oleh *middleware*. Kemudian merancang arsitektur, alur data, struktur data, komponen perangkat keras, dan komponen perangkat lunak.

4.1 Deskripsi Umum Sistem

Terdapat dua tantangan utama yang memotivasi pengembangan *middleware* yakni heterogenitas pada perangkat sensor *healthcare* berbasis IoT dan adanya kebutuhan pengolahan data secara *batch* dan *stream*.

Berdasarkan tantangan tersebut penulis mengembangkan *middleware* untuk mengatasi heterogenitas perangkat sensor melalui abstraksi *web of things* yang memungkinkan aplikasi untuk mengakses data pada perangkat sensor. Selain memungkinkan akses data perangkat sensor, middleware juga dapat memfasilitasi akses data tersebut secara *batch* dan *stream*.

Middleware dikembangkan untuk memungkinkan akses data antara *client application* dan perangkat sensor, sehingga secara umum sistem terbagi menjadi tiga bagian yaitu perangkat sensor, *middleware*, dan *client application*. *Middleware* sendiri dibagi menjadi dua lingkungan yakni *gateway* yang diimplementasikan pada *mini-pc* dan *cloud* yang diimplementasikan pada *virtual server*. Dua lingkungan tersebut terbagi menjadi tiga komponen yaitu *sensor-to-cloud gateway*, *messaging service*, dan *data access interface*. Untuk memungkinkan akses data, pertama-tama perangkat sensor dan *middleware* diberi protokol komunikasi BLE untuk memungkinkan pengiriman data sensor ke *sensor-to-cloud gateway*. Setelah *middleware* dan perangkat sensor memiliki protokol komunikasi yang sama selanjutnya *sensor-to-gateway middleware* dapat mengakses data dari perangkat sensor. Data yang berhasil diakses selanjutnya dikirim oleh *sensor-to-cloud gateway* ke *messaging service*. *Messaging service* akan mengelola data tersebut dengan memasukan kedalam *database* dan menjadi perantara *stream* access melalui *broker*. *Client application* yang membutuhkan akses data secara *batch* akan mengakses data tersebut pada protokol HTTP. Sedangkan *client application* yang membutuhkan data sensor secara stream akan mengakses data tersebut melalui protokol MQTT. Deskripsi umum tersebut dijelaskan detail dari analisis kebutuhan hingga perancangan dalam bab ini.

4.2 Analisis Kebutuhan

Analisis kebutuhan adalah kegiatan yang dilakukan peneliti dalam menggali kebutuhan yang menjadi dasar acuan sebelum dilakukannya perancangan dan

implementasi. Analisis kebutuhan dibagi menjadi tiga macam yaitu kebutuhan fungsional, kebutuhan data dan kebutuhan non-fungsional. Analisis akan disajikan di sebuah tabel beserta penjelasan dan keterangan dari kebutuhan tersebut.

4.2.1 Analisis Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan atau fungsi utama yang harus dipenuhi oleh *middleware*. Kebutuhan fungsional dijelaskan pada tabel 4.1.

Tabel 4.1 Kebutuhan Fungsional

No	Kode	Kebutuhan Fungsional
1	A-MDL-GW-01	*Middleware gateway* dapat terhubung dan menerima data dari satu atau lebih perangkat sensor menggunakan protokol *bluetooth low energy*.
2	A-MDL-GW-02	*Middleware gateway* dapat meneruskan data yang diterima dari perangkat sensor ke *cloud middleware*.
3	A-MDL-MS-01	*Storage subscriber* pada *cloud messaging service* dapat menerima dan menyimpan data yang diteruskan oleh gateway pada database.
4	A-MDL-AI-01	Stream access interface dapat menerima koneksi dan menyediakan data bagi client app menggunakan protokol MQTT Over Websocket.
5	A-MDL-AI-02	*Batch Access Interface* dapat menerima request, mengakses database dan menyediakan data yang di request client .

4.2.2 Analisis Kebutuhan Data

Analisis kebutuhan data dilakukan untuk memudahkan proses representasi *payload* pada perancangan struktur data dengan mempertimbangkan data-data yang nantinya perlu disajikan untuk di proses dalam sebuah aplikasi analisis disajikan. Penjelasan lebih rinci mengenai analisis kebutuhan data dapat dilihat pada tabel 4.2.

Tabel 4.2 Kebutuhan Data

No	Kode	Nama Data	Deskripsi
1	B-MDL-DR-01	*Key*	*Key* pada *payload* dibutuhkan sebagai representasi untuk identifikasi sensor yang menghasilkan *value*.
2	B-MDL-DR-02	*Value*	Value dibutuhkan sebagai nilai asli yang dihasilkan oleh sensor dan dikirim melalui sensor node, value sendiri adalah nilai yang di dapatkan dari kondisi tubuh pasien.
3	B-MDL-DR-03	*Timestamp*	*Timestamp* dibutuhkan untuk menandai waktu ketika data baik *key* dan *value* dikirim.

4.2.3 Analisis Kebutuhan Non-Fungsional

Kebutuhan non-fungsional pada penelitian adalah kebutuhan penunjang sistem, dalam hal ini dibagi menjadi kebutuhan perangkat keras dan perangkat lunak yang digunakan oleh sistem.

4.2.3.1 Kebutuhan Perangkat Keras

Perangkat keras yang dibutuhkan untuk mengimplementasikan *middleware* dijelaskan pada tabel 4.3.

Tabel 4.3 Kebutuhan perangkat keras

No	Kode	Perangkat	Deskripsi
1	C-MDL-HW-01	Mikrokontroler	Mikrokontroler yang digunakan sebagai sensor *node* untuk mengirim data dari perangkat sensor ke *gateway*. Mikrokontroler yang digunakan adalah ESP32
2	C-MDL-HW-02	*Mini computer*	*Mini computer* yang digunakan sebagai *gateway* untuk menerima data dari sensor node dan meneruskan data tersebut ke *messaging service* di *cloud*. *Mini computer* yang digunakan adalah Raspberry Pi Zero W
3	C-MDL-HW-03	*Virtual server*	Sebuah *server virtual* untuk menjalankan *cloud* yang menyediakan *messaging service*, *storage* dan *access interface*. *Virtual server* yang digunakan adalah *virtual instance* pada google cloud platform.

4.2.3.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan untuk mengimplementasikan *middleware* dijelaskan pada tabel 4.4.

Tabel 4.4 Kebutuhan Perangkat Lunak

No	Kode	Perangkat	Deskripsi
1	C-MDL-SW-01	Arduino IDE	IDE untuk memprogram mikro kontroler menggunakan bahasa C.
2	C-MDL-SW-02	Raspbian Stretch	Sistem Operasi yang digunakan untuk mengoperasikan *mini pc* sebagai *gateway*.
3	C-MDL-SW-03	Mosquitto	*Message broker* berbasis protokol MQTT. Yang digunakan sebagai *broker* dan penyedia *stream data access interface*.
4	C-MDL-SW-04	Python	Bahasa Pemrograman yang dilengkapi dengan library untuk mengimplementasikan Bluetooth Low Energy, MQTT *publisher* dan *subscriber*, dan RESTful API.
5	C-MDL-SW-05	Mongodb	*Database* non-sql yang digunakan untuk menyimpan data sensor.

Tabel 4.5 Kebutuhan Perangkat Lunak (lanjutan)

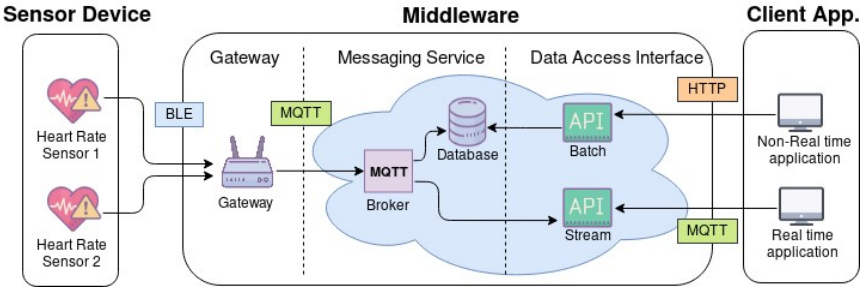
No	Kode	Perangkat	Deskripsi
6	C-MDL-SW-06	pymongo	*Library* python yang digunakan untuk menghubungkan *storage subscriber* dengan *database* mongodb.
7	C-MDL-SW-07	Flask	*Library* dari Python yang digunakan untuk mengimplementasikan RESTful API *webservice*
8	C-MDL-SW-08	bluepy	*Library* dari python yang digunakan untuk mengoperasikan BLE pada komponen *gateway*
9	C-MDL-SW-09	Paho.mqtt.client	*Library* dari python yang digunakan untuk menginisiasi *subscriber* dan *publisher client* pada *indirect communication* protokol MQTT.
10	C-MDL-SW-10	*Ubuntu Server*	Sistem operasi yang digunakan *virtual server*.

4.3 Perancangan

Perancangan *middleware* dilakukan berdasarkan hasil analisis kebutuhan yang sebelumnya telah dilakukan, perancangan bertujuan untuk mendesain *middleware* yang sesuai dengan kebutuhan. Perancangan *middleware* terdiri dari perancangan arsitektur, perancangan aliran data, perancangan struktur data, perangkat keras, dan perangkat lunak. Perancangan tersebut akan dijelaskan secara detail pada sub bab perancangan.

4.4 Perancangan Arsitektur

Sebelum menjelaskan arsitektur dari *middleware*, akan dijelaskan terlebih dahulu dua komponen eksternal diluar middleware yakni perangkat sensor dan *client application*. Kedua komponen tersebut digunakan untuk menguji fungsionalitas *middleware*. Komponen perangkat sensor berfungsi untuk menghasilkan data sensor dan mengirimkannya ke *middleware* melalui *sensor-to-cloud gateway*. Komponen *client application* berfungsi untuk mengakses data yang dihasilkan dari perangkat sensor melalui *data access interface* yang merupakan komponen dari *middleware*. *Middleware* sendiri memiliki tiga komponen yakni *sensor-to-cloud gateway*, *messaging service* dan *data access interface* seperti yang terdapat pada gambar 4.1.

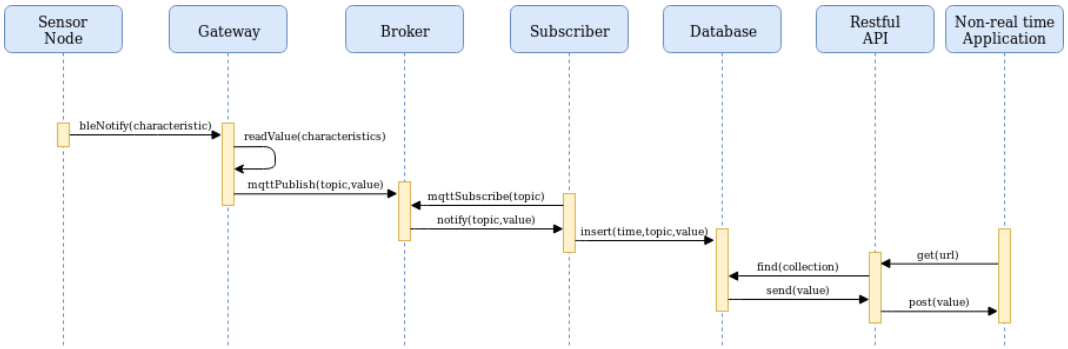


Gambar 4.1 Arsitektur *Middleware*

Komponen pertama adalah *gateway* yang berperan untuk menerima data dari perangkat sensor dan mengirimkan data tersebut ke *cloud*. Komponen kedua adalah *messaging service* yang merupakan bagian dari *cloud*, bertugas untuk meneruskan data dari *gateway* ke dalam *database* dan *stream access interface*. Komponen ketiga adalah *data access interface* yang juga bagian dari cloud yang berfungsi sebagai antarmuka akses bagi client application. Komponen ini memiliki dua jenis akses yakni *batch* dan *stream*. *Stream access interface* diperuntukkan bagi *client application* yang membutuhkan data untuk diolah secara *real time* tanpa menyimpannya secara akumulatif. Sedangkan *batch access interface* diperuntukkan bagi *client application* yang membutuhkan data bersifat historis dengan disimpan sebelum diakses.

4.5 Perancangan Aliran Data

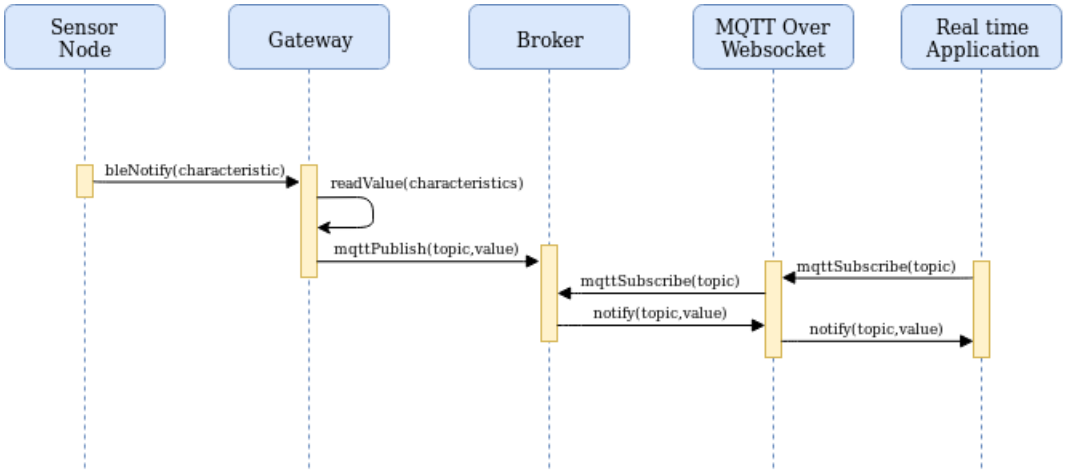
Perancangan aliran data berisi proses abstraksi perangkat yang terjadi dari *gateway* hingga data diakses oleh client app pada data access interface. Terdapat dua aliran data berdasarkan jenis akses yang terdapt pada midddleware, yakni *batch* dan *stream*.



Gambar 4.2 Aliran Data pada *Batch Access*

Gambar 4.2 menjelaskan mekanisme aliran data dari *batch access interface*. Aliran data dimulai saat *sensor node* menotfikasi *value* melalui *characterstics* protokol BLE, selanjutnya *gateway* melakukan *readvalue* yang memungkinkan *gateway* untuk membaca *value* dari BLE *characteristics sensor node*. Setelah *value* didapatkan selanjutnya *gateway* mengirim *value* tersebut ke *cloud* dengan fungsi mqttpublish pada protokol MQTT. Selanjutnya data diteruskan oleh broker

ke *storage subscriber* yang berlangganan pada topik yang di *publish*, ditahap ini data berada pada komponen *cloud*. Data yang diteruskan kemudian disimpan oleh *storage subscriber* ke dalam *database*, ketika API menerima *request* dari aplikasi yang meminta sebuah data maka API akan mengambil data tersebut dari database dan mengirimnya ke aplikasi.

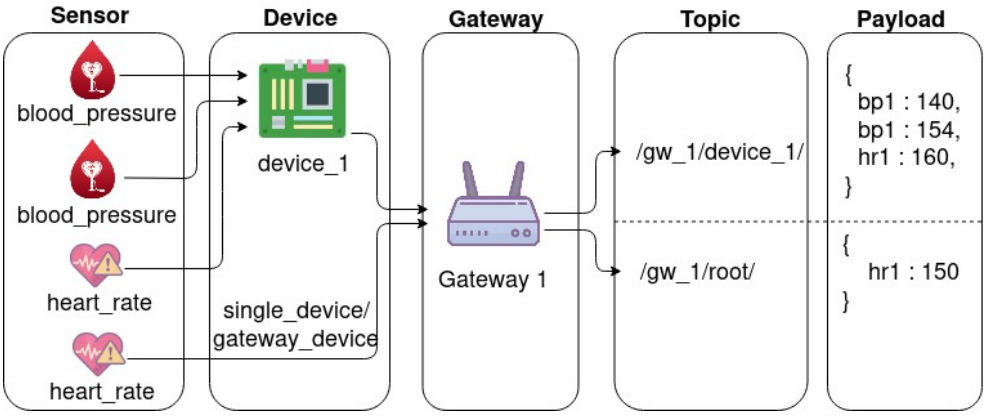


Gambar 4.3 Aliran Data Pada *Stream Access*

Gambar 4.3 menjelaskan mekanisme aliran data dari *stream access interface*. Aliran data dimulai saat *sensor node* menotfikasi *value* melalui *characterstics* protokol BLE, selanjutnya *gateway* melakukan *readvalue* yang memungkinkan *gateway* untuk membaca *value* dari BLE *characteristics sensor node*. *Stream access interface* memiliki urgensi untuk menyediakan data secara langsung setelah data di hasilkan oleh *sensor node*. Hal ini menjadi pertimbangan mengapa *stream access interface* tidak mengakses *database*, melainkan langsung meneruskan data kepada aplikasi. Seperti yang ditunjukkan gambar 4.3 ketika data diteruskan oleh *broker* di *cloud*, selanjutnya data tersebut langsung diteruskan ke aplikasi melalui *stream access interface* menggunakan MQTT *Over Websocket*.

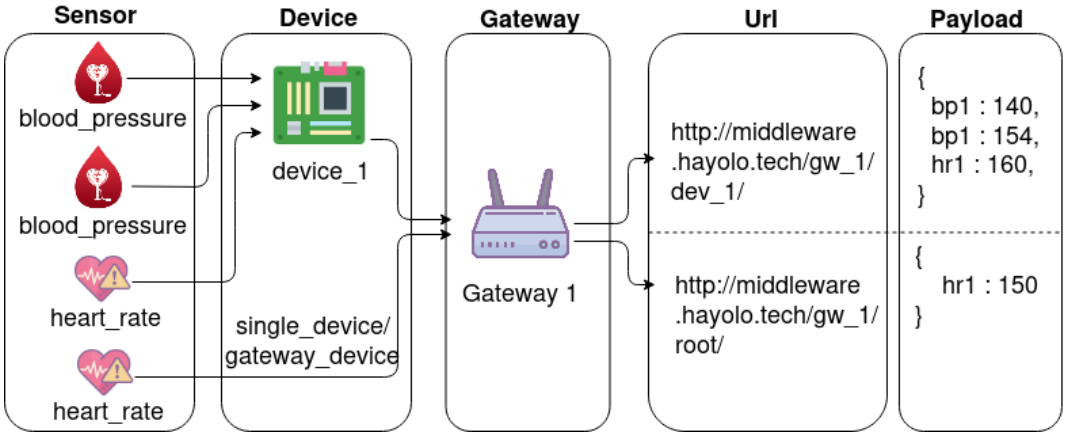
4.6 Perancangan Struktur Data

Terdapat hal penting bagi aplikasi ketika mengakses *data access interface* pada *middleware*, yakni identitas dari perangkat sensor yang akan diakses oleh aplikasi. Untuk mempermudah pengembangan aplikasi dalam mengenali perangkat yang akan diakses oleh aplikasinya perancangan struktur data bertujuan untuk merepresentasikan perangkat sensor dan gateway yang terhubung dengan *data access interface*. Mekanisme pengiriman data yang melibatkan *sensor*, *mikrokontroler* dan *gateway* menjadi acuan dasar dari perancangan struktur data seperti yang ditunjukkan pada gambar 4.4.



Gambar 4.4 Representasi Perangkat *Stream Interface*

Gambar 4.4 menjelaskan acuan dasar dari dua macam representasi perangkat pada *stream interface*. Pertama adalah *stand-alone device* dengan sensor yang terhubung melalui perangkat dan kedua adalah *gateway device* dengan sensor yang menjadi bagian dari *gateway*. *Stand-alone device* dengan topic "/gw_1/device_1/" dan *payload* "hr1" dan "bp1" merepresentasikan struktur data "bp1" dan "hr1" yang berasal dari sensor blood_pressure1 dan heart-rate1 yang dikirim oleh device_1 melalui gateway_1. *Gateway device* dengan topic "/gw_1/root/ dan payload dengan key "hr1" merepresentasikan struktur data dari sensor yang merupakan bagian *gateway*. Perbedaan antara *stand-alone device* dan *gateway device* adalah pada *topic*, gateway device direpresentasikan sebagai root atau bagian dari gateway_1.



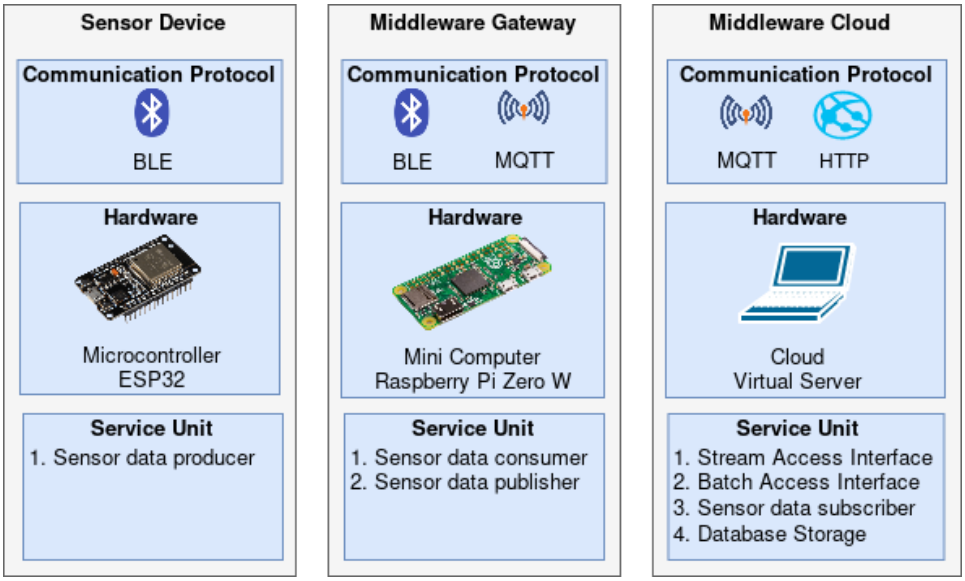
Gambar 4.5 Representasi Perangkat Batch Interface

Gambar 4.5 menjelaskan akses *batch access* yang menggunakan URI untuk mengidentifikasi perangkat yang hendak diakses oleh aplikasi. Representasi perangkat antara stream dan batch interface secara garis besar sama dan perbedaan antar keduanya adalah pada penggunaan *topic* dan URI.

4.7 Perancangan Perangkat Keras

Terdapat beberapa perangkat keras yang digunakan oleh *middleware* untuk beroperasi yakni virtual *server* dan *mini computer*, selain *middleware* terdapat

komponen penghasil data yang beroperasi pada perangkat keras berupa mikro kontroler. Perancangan perangkat keras dari penelitian ini ditunjukkan pada gambar 4.6.



Gambar 4.6 Perangkat Keras Pada *Middleware*

Untuk mengetahui kemampuan *middleware* untuk menerima data sensor maka di diimplementasikan lah sebuah perangkat sensor untuk menghasilkan data. Dalam penelitian ini perangkat sensor terpasang pada mikro kontroler ESP32 dan mengirimkan datanya ke *gateway* menggunakan protokol komunikasi Bluetooth Low Energy.

Gateway merupakan komponen pada *middleware* yang terpasang dan berjalan pada sebuah *mini computer* Raspberry Pi zero W. Terdapat dua layanan pada komponen ini *gateway* pertama adalah sensor data *consumer* untuk menerima data dari perangkat sensor menggunakan BLE. Layanan yang kedua adalah sensor *data publisher* yang berperan untuk mengirim data yang diterima dari perangkat sensor ke komponen *cloud middleware* menggunakan protokol komunikasi MQTT. *Cloud* merupakan komponen *middleware* yang terdapat pada *virtual server* dan memiliki 4 layanan. Layanan pertama adalah sensor data *subscriber* yang bertugas untuk berlangganan dan menerima data dari *gateway* menggunakan MQTT. Layanan yang kedua adalah *database storage* yang di implementasikan menggunakan mongodb, layanan ketiga adalah *batch access interface* dan layanan keempat adalah *stream access interface* yang merupakan sebuah antarmuka bagi *client application*.

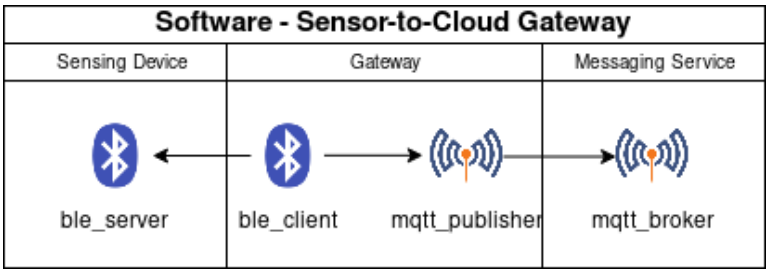
Alur kerja dari *service unit* atau layanan tiap komponen akan dijelaskan pada sub bab 4.8 perancangan perangkat lunak dalam sebuah hubungan antar perangkat lunak dan diagram alir.

4.8 Perancangan Perangkat Lunak

Pada Arsitektur *middleware* terdapat tiga komponen yaitu sensor-to-cloud gateway, *messaging service* dan *data access interface*. Masing-masing dari ketiga komponen terdiri dari beberapa perangkat lunak yang menunjang fungsionalitas *middleware* secara keseluruhan. Perancangan perangkat lunak akan menjelaskan perangkat lunak yang terdapat pada *middleware* beserta mekanisme kerjanya. Perancangan dijelaskan pada tiap komponen yang terdapat pada *middleware*.

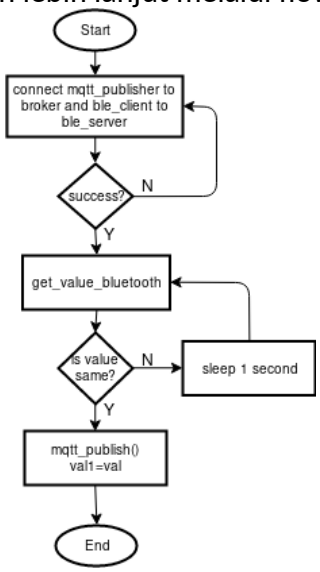
4.8.1 Sensor-to-Cloud Gateway

Sensor-to-Cloud gateway adalah komponen pertama yang melakukan abstraksi dengan terhubung dan mengambil data dari perangkat sensor menggunakan protokol BLE.



Gambar 4.7 Perangkat Lunak pada Sensor-to-Cloud Gateway

Perangkat lunak pada *gateway* terdiri dari *library* bluepy dan paho.mqtt.client, bluepy bertindak sebagai ble_client yang mengambil data sensor dari mikrokontroler dan mqtt.client sebagai *publisher* yang akan meneruskan data ke *broker*. Mekanisme abstraksi yang dilakukan oleh perangkat lunak yang terdapat pada gateway akan dijelaskan lebih lanjut melalui flowchart pada gambar 4.8.



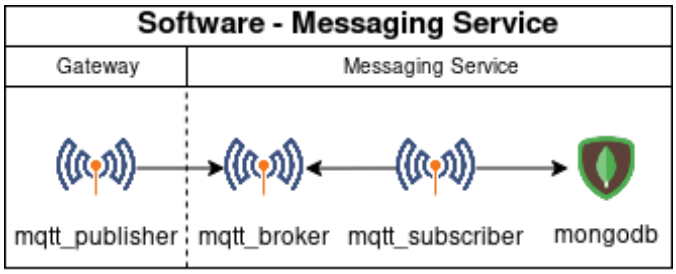
Gambar 4.8 Flowchart Komponen *Gateway*

Hal pertama yang dilakukan oleh *gateway* adalah menghubungkan *publisher* pada *gateway* kepada *broker* di *cloud*, jika gagal terhubung maka program harus dijalankan ulang hingga terhubung. Setelah berhasil terhubung dengan *broker*

selanjutnya menghubungkan ble_client pada *gateway* ke ble_server pada sensor, jika berhasil terhubung selanjutnya *gateway* akan mengambil *service* dan *characteristics*. *Value* dari perangkat sensor diterima oleh *gateway* dengan membaca *value* dari *characteristic* milik ble_server, setelah membaca *value* selanjutnya *gateway* melakukan set *variable* val dan val1 dengan *value* 0 dan *value* dari *characteristic* untuk seleksi. Seleksi ini dilakukan untuk menghindari pengambilan data yang sama dalam waktu tertentu, ketika *value* diupdate maka selanjutnya *value* dikirim ke *cloud* menggunakan protokol MQTT.

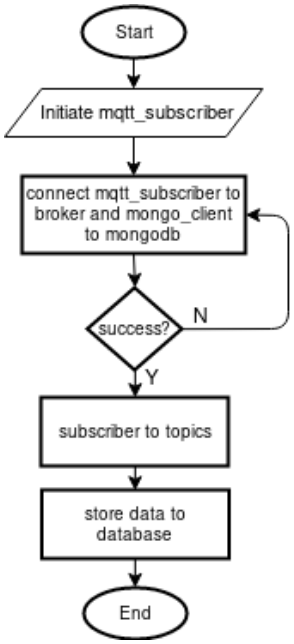
4.8.2 *Messaging Service*

Perangkat lunak pada komponen *messaging service* terdiri dari MQTT *broker*, MQTT *subscriber* dan mongodb. Komponen *messaging service* bekerja dalam beberapa tahapan yang ditunjukkan pada gambar 4.9.



Gambar 4.9 Perangkat Lunak Pada *Messaging Service*

Storage subscriber menyimpan data pada mongodb dengan berlangganan data pada mosquitto *broker* menggunakan paho.mqtt client subscriber.



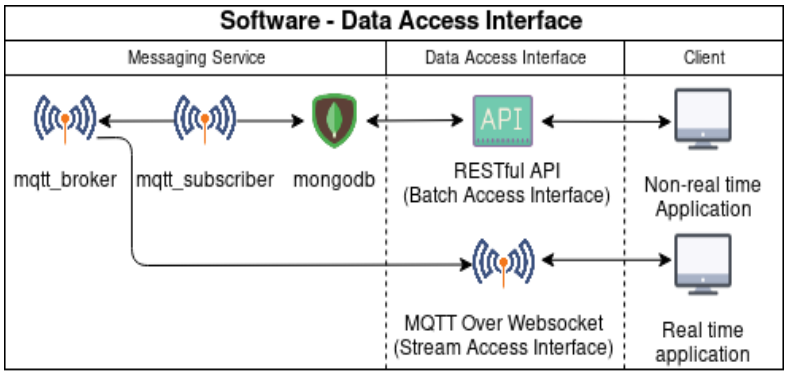
Gambar 4.10 Flowchart *Messaging Service*

Proses pada komponen *messaging service* dimulai dari deklarasi sebuah mqtt subscriber, lalu menghubungkan *subscriber* ke *broker* dan sebuah mongo *client*

pada database mongodb. Baik koneksi *subscriber* pada *broker* maupun koneksi mongo_client pada mongodb jika salah satu dari keduanya gagal terhubung maka proses *connect* akan diulang. Jika subscriber berhasil terhubung selanjutnya akan berlangganan sebuah topic pada *broker*, setelah mendapatkan pesan yang diteruskan oleh *broker* kemudian pesan tersebut disimpan ke dalam *database*.

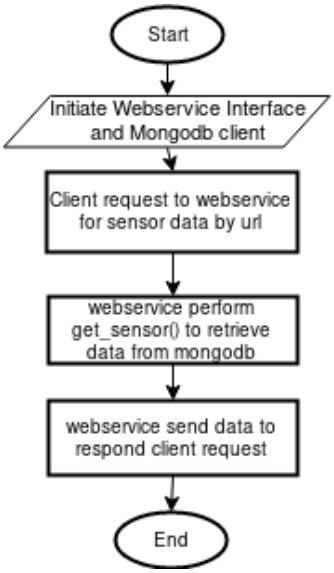
4.8.3 Data Access Interface

Perangkat lunak pada *data access interface* terdiri dari dua jenis *interface* yakni *batch access interface* yang menggunakan RESTful API *webservice* dan *stream access interface* yang menggunakan MQTT *over websocket*. Proses kerja dari komponen *data access interface* dijelaskan pada gambar 4.11.



Gambar 4.11 Perangkat Lunak Pada *Data Access Interface*

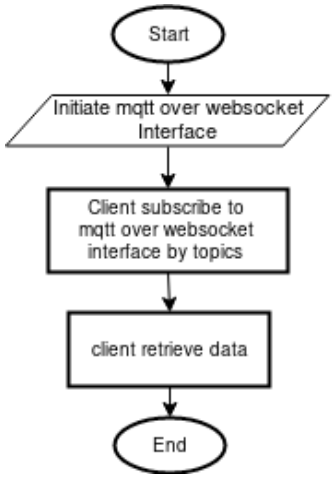
Mekansime pengiriman data pada *batch interface* dijelaskan pada gambar 4.12 dan pada *stream interface* dijelaskan pada gambar 4.13



Gambar 4.12 Flowchart *Batch Data Access Interface*

Proses dari kerja *batch access interface* dimulai dengan mendeklarasikan sebuah *webservice* dan mongodb *client*. *Webservice* merupakan layanan untuk menerima request data dari aplikasi client dan mongodb client bertugas untuk

menghubungkan *webservice* ke *database*. Ketika *webservice* menerima *request* dari *client*, selanjutnya *webservice* akan menjalankan fungsi untuk mengambil data sensor pada *database* yang sesuai dengan url yang di akses oleh *client*. Fungsi get_sensor akan me return data yang didapat dari *database* hingga diterima oleh aplikasi *client*.



Gambar 4.13 Flowchart *Stream Data Access Interface*

Proses dari kerja *stream* access interface dimulai dengan deklarasi *websocket interface* oleh mqtt *broker*, selanjutnya aplikasi *client* akan berlangganan topik sebagai *subscriber* dan menerima data yang dilewatkan melalui *broker* dengan topik yang sesuai secara *stream*.

BAB 5 IMPLEMENTASI

Dalam bab implementasi dijelaskan proses implementasi *middleware* berdasarkan perancangan dan analisis kebutuhan yang sebelumnya dibuat. Implementasi terbagi menjadi implementasi *middleware* dan implementasi komponen eksternal. Implementasi *middleware* meliputi instalasi perangkat lunak dan implementasi layanan pada *gateway*, *messaging service* dan *data access interface*. Implementasi komponen eksternal meliputi implementasi perangkat sensor dan *client application*, komponen eksternal diimplementasikan untuk membantu pengujian *middleware*.

5.1 Implementasi Middleware

Bagian ini menjelaskan proses pengembangan *middleware* yang terbagi menjadi tiga komponen yakni *gateway*, *messaging service* dan *data access interface*. Implementasi diawali dengan instalasi python sebagai lingkungan pengembangan pada tiga komponen *middleware*, lalu instalasi *library* yang dibutuhkan seperti yang dijelaskan pada analisis kebutuhan perangkat lunak. Selanjutnya akan dijelaskan proses implementasi *middleware* dimulai dari komponen *gateway* sebagai pengelola perangkat, *messaging service* sebagai pengelola pesan dan *data access interface* sebagai media akses bagi *client application*. Implementasi akan dijelaskan dalam bentuk *source code*, *screenshoot*, dan deskripsi pada tiap fungsi yang di implementasikan.

5.1.1 Instalasi Middleware

Instalasi *middleware* dilakukan pada dua perangkat keras yakni *mini pc* dan *virtual server*, instalasi keduanya akan dijelaskan pada sub bab Instalasi *middleware*.

Gambar 5.1 Raspberry pi sebagai gateway device

Komponen *gateway* berjalan pada perangkat keras mini pc Raspberry Pi dengan sistem operasi raspbian stretch. Gambar 5.1 menunjukkan raspberry pi zero w

yang digunakan sebagai *gateway*. Mini pc ini dilengkapi dengan Bluetooth 4.0 dan wireless adapter.

Komponen *messaging service* dan *data access interface* berjalan pada sebuah cloud. Pada penelitian ini *cloud* diimplementasikan pada *virtual server* menggunakan google cloud platform dengan detail spesifikasi yang ditunjukkan pada tabel 5.1

Tabel 5.1 Spesifikasi *Virtual Server*

Hardware Name	g1-small
Processor	2.30GHz, Single-core CPU
Memory	1.7 GB
Connectivity	VPC, External IP
IP Address	10.128.0.2, 35.188.201.80
Domain	middleware.hayolo.tech
Region	us-central1-a

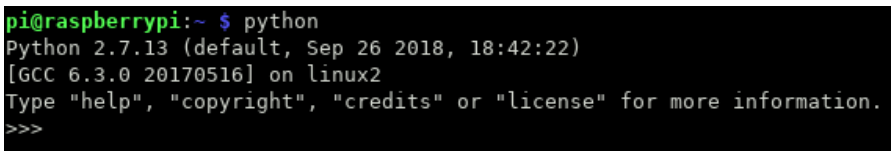
5.1.2 Instalasi Python

Middleware dikembangkan menggunakan bahasa dan kumpulan library python. Instalasi python pada *gateway* dan *cloud* sebagai interpreter kode yang digunakan sebagai langkah awal pengembangan. Tabel 5.2 menunjukkan perintah instalasi python.

Tabel 5.2 Perintah Instalasi Python

```
$ sudo apt-get install python
```

Untuk melihat apakah python telah terpasang pada *gateway* dapat dilihat dengan menjalankan perintah yang terdapat pada gambar 5.2



Gambar 5.2 Cek Instalasi Python

Setelah terpasang selanjutnya adalah instalasi *library* yang dibutuhkan *middleware* untuk beroperasi. *Library* python dapat di install menggunakan *package manager* python yang disebut pip berikut adalah perintah instalasi pip yang ditunjukkan pada tabel 5.3.

Tabel 5.3 Perintah Instalasi Python *Package Manager* Pip

```
$ sudo apt-get install python-pip
```

Untuk melihat python yang telah terpasang dapat dilakukan pengecekan versi seperti gambar 5.3

Gambar 5.3 *Version Check* pada pip

Selanjutnya adalah instalasi kumpulan *library* yang dibutuhkan *middleware*. Berikut adalah perintah instalasi *library* yang ditunjukkan pada tabel 5.4

Tabel 5.4 Perintah Instalasi *Library Middleware*

```
$ sudo pip install paho.mqtt
$ sudo pip install bluepy
$ sudo pip install pymongo
$ sudo pip install flask
```

Kumpulan *library* python tersebut adalah *library* yang dibutuhkan untuk menjalankan *service* di masing-masing komponen *middleware*. *library* tersebut akan dijelaskan fungsinya berdasarkan layanan dari komponennya. *Library* pertama adalah paho.mqtt yang digunakan oleh komponen *gateway* dan *messaging service* untuk melakukan *publish* dan *subscribe* pada pesan yang dikirim menggunakan protokol MQTT. *Library* selanjutnya adalah Bluepy yang diperlukan untuk membangung koneksi dan melakukan pertukaran data antara *gateway* dengan perangkat sensor pada protokol ble. Pada komponen *messaging service* terdapat sebuah *database storage* mongodb, *library* pymongo dibutuhkan oleh komponen *messaging service* dan *data access interface* untuk mengakses *database* tersebut. *Library* terakhir yakni flask dibutuhkan komponen *data access interface* untuk membuat sebuah RESTful *webservice* yang menyediakan API.

5.1.3 Instalasi Mosquitto Broker

MQTT Broker yang digunakan pada penelitian ini adalah mosquitto, broker tersebut diinstall pada komponen *messaging service*. Tabel 5.5 menunjukkan perintah instalasi mosquitto pada penelitian ini

Tabel 5.5 Perintah Instalasi Library Middleware

```
$ sudo apt-get install mosquitto
```

5.1.4 Instalasi MongoDB

Instalasi mongodb dilakukan untuk menyediakan *storage* bagi middleware di penelitian ini. Perintah instalasi mongodb ditunjukkan pada tabel 5.6

Tabel 5.6 Perintah Instalasi *Library Middleware*

```
$ sudo apt-get install mongodb
```

5.1.5 Konfigurasi Jaringan

Untuk mempermudah penelitian dilakukan konfigurasi jaringan pada *gateway* dengan detail seperti yang ditunjukkan pada tabel 5.7

Tabel 5.7 File Konfigurasi Jaringan pada G*ateway*

```
interface eth0
static ip_address=192.168.43.13/24
static routers=192.168.43.16
static domain_name_servers=192.168.43.1 8.8.8.8
```

Sementara konfigurasi jaringan pada cloud menggunakan default setting dengan detail seperti yang ditunjukkan pada tabel 5.8

Tabel 5.8 File Konfigurasi Jaringan pada *Gateway*

Instance Domain	Zone	Internal IP	External IP
middleware.hayolo.tech	us-central1-a	10.128.0.2	35.188.201.80

Tabel 5.8 menunjukkan domain, zona, alamat ip publik, dan *private* yang digunakan komponen *messaging service* dan *data access interface* yang berjalan pada *virtual server*.

5.1.6 Implementasi *Sensor-to-Cloud Gateway*

Komponen *sensor-to-cloud gateway* berperan sebagai *device manager* yang bertugas untuk menangani perangkat sensor yang terhubung ke *middleware*, mengambil data dari perangkat sensor dan mengirimkannya ke *cloud*. Tugas tersebut dilakukan oleh *service unit device management*. Penjelasan *service unit device management* pada tabel 5.9

Tabel 5.9 Pseudocode Service Unit Device Management

```
SET pub <- mqttClient(transport='websocket')
DO pub.connect(brokerAddress, brokerPort)
DO print "mqtt client connected"
Function deviceManagement(device_address, service_uuid, topic)
  SET dev <- connect(device_address)
  SET uuid <- getService(service_uuid)
  SET sensorservice <- dev.getService(uuid)
  LOOP ch in sensorservice.getCharacteristic()
    SET val,val1 = 0
    LOOP if true
      val <- ch.read()
      IF val != val1
        SET value <- decode(val)
        DO print value
        DO pub.publish(topic, value)
      ELSE IF val == val1
        DO sleep 1 second
```

Tabel 5.10 Pseudocode Service Unit Device Management (lanjutan)

```
SET thread <- createThread(bluetooth(device_address, service_uuid,
topic))
```

Untuk menjalankan *device management*, *gateway* terlebih dahulu menginisialisasi mqtt *client* sebagai *publisher* dan menghubungkannya dengan *broker*. Selanjutnya menjalankan *device management* dengan parameter MAC *address*, uuid *service*, dan *topic* mqtt. Hal pertama yang dilakukan *device management* adalah membangun koneksi dengan perangkat sensor menggunakan MAC address dan mengambil datanya menggunakan UUID *service* pada protokol BLE. Setelah berhasil selanjutnya fungsi melakukan seleksi untuk hanya mengirimkan data yang terbaru saja pada *broker*. Fungsi tersebut dijalankan menggunakan *thread* yang di inisialisasikan di akhir *service unit*, tujuan penggunaan *thread* adalah untuk mengelola dua atau lebih perangkat.

5.1.7 Implementasi *Messaging Service*

Messaging service memiliki dua *service unit* yakni MQTT *broker* dan *storage subscriber*, MQTT *broker* diimplementasikan dengan mosquitto *broker* yang berjalan pada port 9001. *Service unit* yang kedua adalah *storage subscriber* yang berfungsi untuk berlangganan pesan pada *broker* dan menyimpan pesan tersebut kedalam *database storage* mongodb, cara kerja *service unit storage subscriber* dijelaskan pada tabel 5.11.

Tabel 5.11 Pseudocode *Service Unit Storage Subscriber*

```
SET sub <- mqttClient(transport='websockets')
SET client <- MongoClient(databaseIp, databasePort)
SET db <- client.databaseName
DO sub.connect(brokerAddress, brokerPort)
DO sub.subscribe(messageTopic)
Function handle_message(mqttc, obj, msg)
  SET topic <- msg.topic
  SET split1 <- splitTopicBy("/")
  SET payload <- decodeMessage()
  DO print(topic, payload, split[1], split[2])
  DO insertDatabase(timestamp, gateway, device, payload)
DO handle_message
LOOP for until interrupt
```

Tabel 5.11 menunjukkan proses kerja *storage subscriber* yang diawali dengan inisialisasi *client* untuk koneksi pada broker mosquitto dan database mongodb, selanjutnya *service unit subscribe* pada sebuah *topic*. Terdapat fungsi handle_message yang melakukan penanganan *payload*, identifikasi *gateway* dan *device* berdasarkan hasil split topic, kemudian fungsi simpan pesan pada *database*. Terakhir adalah pemanggilan fungsi *handle message* yang diikuti sebuah proses *loop*.

5.1.8 Implementasi *Data Access Interface*

Data access interface memiliki dua jenis *interface* yang dibagi menjadi dua *service unit*. *Service unit* yang pertama adalah *stream access interface* yang berjalan menggunakan protokol *websocket* pada MQTT *over websocket*. Implementasi dari MQTT *over websocket* dapat dilakukan dengan mengkonfigurasi MQTT broker dengan baris perintah yang didefinisikan pada file konfigurasi mosquitto broker seperti yang ditampilkan pada tabel 5.12.

Tabel 5.12 Konfigurasi *Broker* pada file /etc/mosquitto/mosquitto.conf

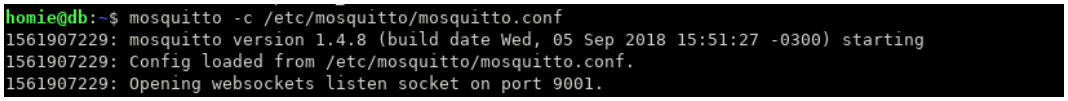
```
listener 9001
protocol websockets
```

Konfigurasi pada tabel 5.12 memungkinkan *broker* menyediakan akses *websockets* bagi paket MQTT yang melewati *port* 9001. Semua paket MQTT yang melewati *port* tersebut akan dienkapsulasi kedalam paket *websocket* lalu dikirimkan. Untuk menjalankan mosquitto broker dengan konfigurasi tersebut dapat dilakukan dengan perintah yang ditunjukkan pada tabel 5.13.

Tabel 5.13 Perintah Menjalankan Mosquitto Dengan File Konfigurasi

```
mosquitto -c /etc/mosquitto/mosquitto.conf
```

Perintah tersebut akan menjalankan *broker* dengan *file* konfigurasi yang sudah dibuat sebelumnya, gambar 5.4 menunjukkan *output* pada saat menjalankan mosquitto pada protokol *websocket*.



Gambar 5.4 Menjalankan mosquitto dengan websocket

Gambar 5.4 menunjukkan bahwa broker melakukan listen untuk koneksi dengan protokol *websocket* pada *port* 9001. Untuk bekerja dengan protokol *websocket*, baik *client publisher* maupun *subscriber* harus menambahkan metode pengiriman *websocket* pada kode program seperti yang ditunjukkan pada tabel 5.14.

Tabel 5.14 Implementasi client stream interface

```
client = mqtt_client.Client(transport='websockets')
client.connect("middleware.hayolo.tech", port=9001)
```

Dua baris kode tersebut adalah inisiasi *client* menggunakan protokol *websocket* yang digunakan pada *service unit* dengan menyesuaikan *port* maupun alamat dari *broker* pada *messaging service*.

Selanjutnya adalah *service unit batch access interface* yang diimplementasikan menggunakan library flask untuk membuat *webservice* API dengan arsitektur RESTful. Pseudocode dari *service unit batch access interface* ditunjukkan pada tabel 5.15.

Tabel 5.15 Pseudocode *webservice batch interface*

```
SET app <- Flask()
DO app.configDatabase(“mongodb://address:port/database”)
SET mongo <- PyMongo(app)
DO app.route(/gw_1/, method=get)
Function get_sensor1():
  SET sensor <- mongo.db.sensor
  SET output
  Loop for sensor data
    output <- sensordata[time][gateway][device][value]
  Return output
DO app.route(/gw_1/dev_1/, method=get)
Function get_sensor2():
  SET sensor <- mongo.db.sensor
  SET output
  Loop for sensor data which(device == dev_1)
    output <- sensordata[time][gateway][device][value]
  Return output
DO app.run(host=’10.128.0.2’)
```

Dimulai dengan menginialisasi *webservice* menggunakan library flask beserta Pymongo untuk menghubungkan flask dengan mongodb. Fungsi app.route dijalankan untuk menyediakan akses pada URI /gw_1/, lalu menginisialisasi *collection database* pada *variable* sensor dan *variable* kosong output. Lakukan *loop* sebanyak isi data pada *database* dan *store value* dari fungsi sensordata pada *variable output* dan beri kembalian *output*. Fungsi kedua merupakan fungsi yang sama dengan fungsi pertama. Fungsi kedua diperuntukkan bagi akses untuk perangkat sensor yang berjalan pada perangkat tertentu. Terakhir, fungsi app.run digunakan untuk menjalankan *webservice*.

5.2 Implementasi Komponen Eksternal

Pada bab perancangan telah dijelaskan komponen eksternal merupakan komponen yang diimplementasikan untuk menguji kinerja dari middleware. Komponen ini adalah perangkat sensor dan client application.

5.2.1 Implementasi Perangkat Sensor

Perangkat sensor diimplementasikan menggunakan mikrokontroler ESP32 sebagai perangkat penghasil data. Pada implementasi penulis menggunakan data dummy. Perangkat sensor menggunakan protokol komunikasi BLE dalam melakukan pengiriman data. Tabel 5.16 menjelaskan pseudocode dari sensor device.

Tabel 5.16 Pseudocode BLE *Server*

```
SET BLEServer* pServer, BLECharacteristic* characteristic,
deviceConnected, oldDeviceConnected, value[]
DEFINE serviceUUID, CharacteristicUUID
Class ServerCallback
  Function onConnect
    SET deviceConnected <- true
  Function onDisconnect
    SET deviceConnected <- false
Function setup
  DO BLEDevice::init(“ESP32Device”)
  SET pServer <- BLEDevice::createServer()
  SET pServer -> setCallbacks(new ServerCallback())
  SET BLEService *pService <- createService(serviceUUID)
  characteristic <- createCharacteristic(CharacteristicUUID, READ,
NOTIFY)
  DO startService
  SET BLEadvertising *pAdvertising <- BLEDevice::getAdvertising()
  DO startAdvertising()
Function loop
  IF deviceConnected
    DO characteristicSetValue(value[])
    DO characteristicNotify()
    DO value[]++
    DO delay(3000)
  IF Not deviceConnected and oldDeviceConnected
    DO delay(500)
    DO reAdvertise()
    SET oldDeviceConnected <- deviceConnected
  IF deviceConnected and Not oldDeviceConnected
    SET oldDeviceConnected <- deviceConnected
```

Perangkat sensor pertama-tama melakukan set beberapa variable yakni server dan characteristic BLE, deviceConnected, oldDeviceConnected. Selanjutnya melakukan set alamat UUID pada serviceUUID dan characteristicUUID yang akan digunakan sebagai identitas dari *service* dan *characteristic* BLE. Setelah set variabel beserta alamatnya, lalu mendefinisikan class berisi callbackserver yang berfungsi untuk mengubah *state* dari BLE *server* saat terkoneksi dan terdiskoneksi. Selanjutnya terdapat fungsi *setup* yang berisi inisiasi *server*, *service characteristic* dari BLEServer, dan fungsi advertise untuk broadcast BLEServer. Fungsi yang terakhir adalah loop yang berisi perintah untuk menyediakan value dan advertise kembali ketika koneksi terputus secara berulang.

5.2.2 Implementasi *Client Application*

Terdapat dua *client application* pada *data access interface*. Namun untuk menguji *batch access interface* dapat dilakukan dengan akses melalui *browser*. Sedangkan *stream client application* diimplementasikan menggunakan javascript dengan dukungan *library* dengan akses *websocket*. Implementasi client application dijelaskan pada tabel 5.17

Tabel 5.17 Pseudocode *Client Application Websocket*

```
SET BrokerAddress, BrokerPort, Topic, mqtt
Function MQTTconnect()
  IF path == undefined THEN path == ‘/mqtt’
  mqtt <- new mqtt.client(host,port,path)
Function onConnect()
  SET #status <- html(host port path)
  DO mqtt.subscribe(topic)
  SET #topic <- html(topic)
Function onConnectionLost(response)
  DO setTimeOut()
Function onMessageArrived(message)
  SET topic <- message.destinationName;
  SET payload <- message.payloadString
  SET #message <- html(topic , payload)
  SET message = topic.split(‘/’)
  SET gateway = message[1]
  SET device = message[2]
  IF device == ‘dev_1’
    SET #value5 <- html(gateway)
    SET #value6 <- html(device)
    SET #label1 <- text(payload)
  IF device == ‘dev_2’
    SET #value3 <- html(gateway)
    SET #value4 <- html(device)
    SET #label2 <- text(payload)
```

Pertama *client application* menginisialisasi alamat *broker*, *port broker*, topik pesan dan variable kosong mqtt. Kemudian mendefinisikan fungsi untuk membangun koneksi dengan broker. Dideklarasikan fungsi onconnect untuk menghandle koneksi yang masuk dan mengisi variable status dan topic. Fungsi onlostconnection akan melakukan set timeout untuk membatasi waktu kosong koneksi hingga mengulang proses penghubungan kembali. Fungsi onMessageArrived merupakan fungsi untuk menghandle pesan dengan mengambil payload dan topic. Fungsi onMessageArrived juga melakukan

pemisahan topic untuk merepresentasikan gateway dan perangkat. Hasil pseudocode terdapat pada gambar 5.5.

Gambar 5.5 *Client Application Websocket*

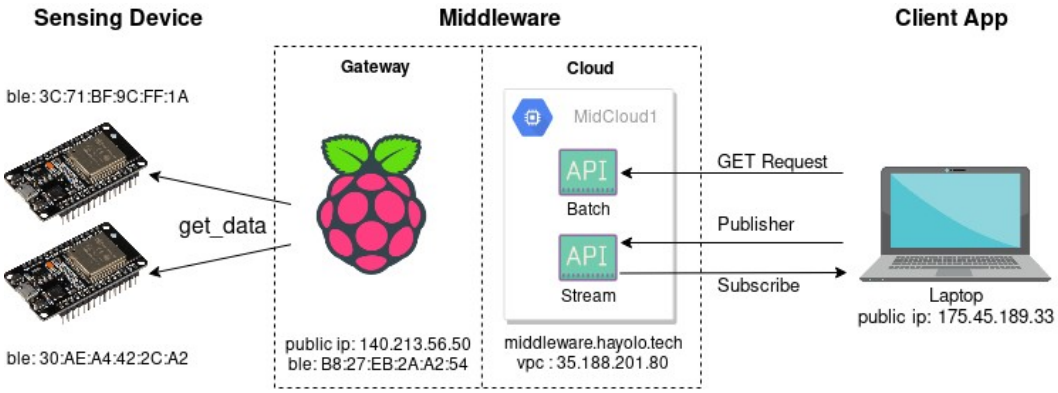
Websocket client application yang ditunjukkan pada gambar 5.5 memiliki beberapa field yang berisi jenis sensor dan data yang dihasilkan secara *stream*. Selain itu terdapat field yang berisi identitas gateway dan device sumber yang digunakan.

BAB 6 PENGUJIAN

Bab pengujian menjelaskan lingkungan uji, skenario, proses, dan hasil pengujian *middleware*. Pengujian yang dilakukan terdiri dari pengujian fungsional dan pengujian kinerja pada batch interface, stream interface, dan concurrent user.

6.1 Perancangan Lingkungan Uji

Perancangan lingkungan uji dibuat untuk menggambarkan lingkungan yang digunakan pada pengujian. Gambar 6.1 menunjukkan lingkungan pengujian pada penelitian ini.



Gambar 6.1 Lingkungan Uji Penelitian

Lingkungan uji terdiri dari perangkat sensor, *middleware*, dan *client application*. Perangkat sensor menggunakan ESP32 untuk menghantarkan data menggunakan perangkat *bluetooth* dengan alamat fisik 3C:71:BF:9C:FF:1A dan 30:AE:A4:42:2C:A2 yang berperan sebagai BLE *server*. *Middleware* terbagi menjadi *gateway* dan *cloud*. *Gateway* diimplementasikan di sebuah Raspberry Pi yang dilengkapi *bluetooth interface* dengan alamat B8:27:EB:2A:A2:54 untuk berkomunikasi dengan perangkat sensor. *Gateway* juga memiliki 802.11 *interface* yang terpasang alamat IP publik untuk berkomunikasi dengan cloud. Saat pengujian berlangsung alamat IP publik yang didapatkan adalah 140.213.56.50. Cloud diimplementasikan pada layanan virtual instance google cloud platform dengan domain middleware.hayolo.tech dan IP publik 35.188.201.80. *Cloud* terletak pada *availability zone* us-central1-a kota *Council Bluffs*, *Amerika Serikat*. *Client application* yang digunakan pada pengujian menggunakan jaringan Universitas Brawijaya dengan IP publik 175.45.189.33. *Client application* merupakan entitas yang menguji dua *interface* pada *middleware yakni batch dan stream*. Pengujian *batch interface* dilakukan dengan mengirim *request dan stream* dengan melakukan *publish-subscribe* yang secara detail dijelaskan pada sub bab skenario.

6.2 Pengujian Fungsional

Pengujian fungsional dilakukan untuk menyesuaikan *middleware* yang diimplementasikan dengan kebutuhan fungsional yang telah dibuat pada analisis kebutuhan fungsional. Terdapat lima kebutuhan fungsional yang diuji dengan proses, hasil serta penjelasan pada kumpulan sub bab berikut.

6.2.1 Skenario Pengujian Fungsional

Pengujian fungsional dilakukan untuk menyesuaikan *middleware* dengan analisis kebutuhan fungsional. Pengujian fungsional dilakukan berdasarkan skenario yang dibuat pada tabel 6.1.

Tabel 6.1 Skenario Pengujian Fungsional

Kode	Fungsi	Skenario
A-MDL-GW-01	*Middleware gateway* dapat terhubung dan menerima data dari satu atau lebih perangkat sensor menggunakan protokol *bluetooth low energy*.	• BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi • Menjalankan *service unit device management* pada *middleware gateway* • Melakukan validasi terhadap kemampuan *middleware* untuk menerima data dari perangkat sensor dengan melakukan *print value* pada *data* yang diterima dari perangkat sensor.
A-MDL-GW-02	*Middleware gateway* dapat meneruskan data yang diterima dari perangkat sensor ke *broker* pada *cloud messaging service* menggunakan protokol MQTT.	• BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi • Menjalankan *broker* pada *cloud messaging service* • Menjalankan *service unit device management* pada *middleware gateway* • Melakukan validasi terhadap fungsi *middleware* dalam meneruskan *data* dengan melihat *output* dari *verbose broker* yang dijalankan.

Tabel 6.2 Skenario Pengujian Fungsional (lanjutan)

ADL-MS-01	*Broker* pada *cloud messaging services* dapat meneruskan pesan dari *gateway* ke *storage subscriber* dan *stream data access interface*.	• BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi • Menjalankan *broker* pada *cloud messaging service* • Menjalankan *subscriber* pada *cloud* • Menjalankan *service unit device management* pada *middleware gateway* • Melakukan validasi terhadap fungsi *middleware broker* dalam meneruskan *data* dengan melihat *output* dari *subscriber* yang berjalan pada *cloud*.
A-MDL-AI-01	*Stream access interface* dapat menangani koneksi dan menyediakan data bagi *client application* yang melakukan *subscribe* pada MQTT *over websocket*	• BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi • Menjalankan *broker* pada *cloud messaging service* dengan konfigurasi mosquitto *over websocket* • Menjalankan *websocket client application* • Menjalankan *service unit device management* pada *middleware gateway* • Melakukan validasi terhadap fungsi *stream interface* pada *middleware* dalam menerima koneksi dan menyediakan data ke *client* dengan mengecek *output* yang dihasilkan *websocket client application*.
A-MDL-AI-02	*Batch Access Interface* dapat menangani request, mengakses mongo *database* dan menyediakan data yang di *request client* .	• Menjalankan *database* mongodb • Menjalankan *webservice* API • Mengakses API melalui *browser* pada URI http://middleware.hayolo.tech/gw_1/, http://middleware.hayolo.tech/gw_1/dev_1/, http://middleware.hayolo.tech/gw_1/dev_2/ • Melakukan validasi terhadap fungsi

		batch interface pada *middleware* dalam menerima koneksi, mengakses *database* dan menyediakan data untuk *request client* dengan mengecek *output* dari akses *browser* ke API.

6.2.2 A-MDL-GW-01

A-MDL-GW-01 bertujuan untuk menguji kemampuan *sensor-to-cloud gateway* untuk terhubung dan menerima data dari ESP32 melalui protokol BLE. Proses pengujian A-MDL-GW-01 dapat dilihat pada tabel 6.3

Tabel 6.3 Proses pengujian A-MDL-GW-01

Kode	A-MDL-GW-01
Nama fungsi	*Middleware gateway* dapat terhubung dan menerima data dari satu atau lebih perangkat sensor menggunakan protokol BLE (*Bluetooth Low Energy)*.
Prosedur pengujian	1. BLE *Server* pada satu atau lebih perangkat sensor berjalan dan melakukan *advertise* 2. Menjalankan *service unit device management* pada *gateway* 3. Melakukan validasi terhadap kemampuan *middleware* untuk menerima data dari perangkat sensor dengan melakukan *print value* pada data yang diterima dari perangkat sensor.
Hasil yang diharapkan	*Gateway* berhasil menerima data dari ESP32 melalui protokol BLE (Bluetooth Low Energy)

Pengujian pada A-MDL-GW-01 dibuktikan dengan keberhasilan *gateway* dalam menerima data dari perangkat sensor. Berikut adalah hasil dan penjelasan dari pengujian A-MDL-GW-01.

```
pi@raspberrypi:~/skripsi/main_code $ ./gateway.py
Bluetooth Connecting...
Bluetooth Connected
value: 117.00
value: 105.00
```

```
value: 139.00
value: 158.00
value: 117.00
```

Gambar 6.2 Output *Gateway* Berisi *Value* Perangkat Sensor

Gambar 6.2 menunjukkan hasil *output* dari *gateway* yang mendapatkan *value* dari perangkat sensor. Gambar tersebut membuktikan bahwa gateway dapat terhubung dan mengambil data dari perangkat sensor.

6.2.3 A-MDL-GW-02

A-MDL-GW-02 bertujuan untuk menguji kemampuan *sensor-to-cloud gateway* untuk meneruskan data yang diterima dari ESP32 melalui protokol BLE (Bluetooth Low Energy) menuju *cloud* pada *middleware*. Proses pengujian A-MDL-GW-02 dapat dilihat pada tabel 6.4

Tabel 6.4 Proses pengujian A-MDL-GW-02

Kode	A-MDL-GW-02
Nama fungsi	*Middleware gateway* dapat meneruskan data yang diterima dari perangkat sensor ke *cloud middleware*.
Prosedur pengujian	1. BLE *Server* pada satu atau lebih perangkat sensor berjalan dan melakukan *advertise* 2. Menjalankan *cloud messaging service* 3. Menjalankan *service unit device management* pada *middleware gateway* 4. Melakukan validasi terhadap fungsi *middleware* dalam meneruskan *data* dengan melihat *output* dari *verbose broker* yang dijalankan.
Hasil yang diharapkan	*Middleware* cloud berhasil menerima data yang dikirim oleh *gateway*.

Pengujian pada A-MDL-GW-02 dibuktikan dengan keberhasilan *gateway* dalam meneruskan data dari perangkat sensor ke *cloud* pada *middleware*. Berikut adalah hasil dan penjelasan dari pengujian A-MDL-GW-02.

```
1566208752: New client connected from 140.213.58.6 as Middleware
Gateway (c1, k60).
```

```
1566208766: Socket error on client Middleware Gateway,
disconnecting.
```

Gambar 6.3 Output log file mosquitto broker

Gambar 6.3 menunjukkan output berupa log dari mosquitto *message broker* pada *cloud*. Pada log tersebut terlihat *cloud* menerima koneksi dari *gateway*, hal ini membuktikan keberhasilan koneksi antara *gateway* dan *cloud*.

```
satria@mid-cloud1:~$ python storage_subscriber.py
Topik : /gw_1/dev_1/ Payload : 95.00 Gateway : gw_1 Device :
dev_1
Topik : /gw_1/dev_1/ Payload : 133.00 Gateway : gw_1 Device :
dev_1
Topik : /gw_1/dev_1/ Payload : 90.00 Gateway : gw_1 Device :
dev_1
Topik : /gw_1/dev_1/ Payload : 128.00 Gateway : gw_1 Device :
dev_1
Topik : /gw_1/dev_1/ Payload : 100.00 Gateway : gw_1 Device :
dev_1
```

Gambar 6.4 Data diterima oleh messaging service pada cloud

Selanjutnya gambar 6.4 menunjukkan *output* dari *storage subscriber* pada *cloud*. Pada *output* tersebut *cloud* telah menerima data yang dikirim dari *gateway*. Dari hasil tersebut dapat disimpulkan bahwa *gateway* mampu meneruskan data yang dihasilkan oleh perangkat sensor hingga diterima oleh cloud.

6.2.4 A-MDL-MS-01

A-MDL-MS-01 bertujuan menguji kemampuan *storage subscriber* untuk menerima dan menyimpan data yang diteruskan oleh *broker* kedalam *storage database*. Proses pengujian A-MDL-MS-01 dapat dilihat pada tabel 6.5.

Tabel 6.5 Proses pengujian A-MDL-MS-02

Kode	A-MDL-MS-01
Nama fungsi	*Storage subscriber* pada *cloud messaging service* dapat menerima dan menyimpan data yang diteruskan oleh *gateway* pada *database*.
Prosedur pengujian	• BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi • Menjalankan *broker* pada *cloud messaging service*

	• Menjalankan *service unit storage subscriber* pada *cloud* • Menjalankan mongodb *database service* • Menjalankan *service unit device management* pada *middleware gateway* • Melakukan validasi terhadap fungsi *middleware storage subscriber* dalam menerima dan menyimpan data dengan mengecek kesesuaian *output* dari *service unit storage subscriber* dengan isi *collection* pada mongodb.
Hasil yang diharapkan	*Storage subscriber* dapat menyimpan data yang didapat dari *messaging service* kedalam *database*.

Pengujian pada A-MDL-MS-01 dibuktikan dengan keberhasilan *storage subscriber* dalam menyimpan data yang diterima dari *gateway* kedalam *database*. Berikut adalah hasil dan penjelasan dari pengujian A-MDL-MS-01.

```
> db.mySensor.find()
{ "_id" : ObjectId("5d5b7665bfd8556a03f60859"), "Device" :
"dev_1", "TIMESTAMP" : ISODate("2019-08-20T04:26:13.368Z"),
"Gateway" : "gw_1", "value" : "95.00" }
{ "_id" : ObjectId("5d5b768cbfd8556a03f6085a"), "Device" :
"dev_1", "TIMESTAMP" : ISODate("2019-08-20T04:26:52.381Z"),
"Gateway" : "gw_1", "value" : "133.00" }
{ "_id" : ObjectId("5d5b768ebfd8556a03f6085b"), "Device" :
"dev_1", "TIMESTAMP" : ISODate("2019-08-20T04:26:54.090Z"),
"Gateway" : "gw_1", "value" : "90.00" }
{ "_id" : ObjectId("5d5b7691bfd8556a03f6085c"), "Device" :
"dev_1", "TIMESTAMP" : ISODate("2019-08-20T04:26:57.898Z"),
"Gateway" : "gw_1", "value" : "128.00" }
{ "_id" : ObjectId("5d5b7694bfd8556a03f6085d"), "Device" :
"dev_1", "TIMESTAMP" : ISODate("2019-08-20T04:27:00.705Z"),
"Gateway" : "gw_1", "value" : "100.00" }
```

Gambar 6.5 *Output* Berupa Data Tersimpan Pada *Storage Subscriber*

Gambar 6.5 menunjukkan *output* dari mongodb pada *cloud*. Pada *output* tersebut terdapat data yang sebelumnya diterima oleh *storage subscriber* dan saat ini berada di dalam *database*. Hal ini menunjukkan bahwa fungsional *storage subscriber* adalah *valid*.

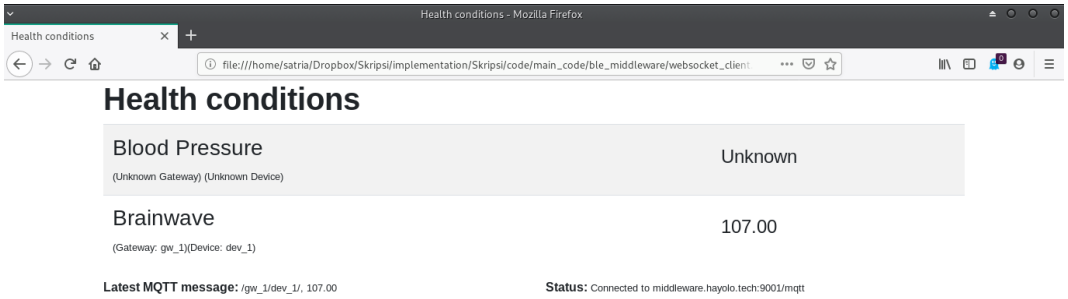
6.2.5 A-MDL-AI-01

A-MDL-MS-01 bertujuan untuk menguji kemampuan *stream access interface* dalam menangani koneksi dan menyediakan data bagi *client application* pada protokol MQTT *over websocket*. Proses pengujian A-MDL-AI-01 dapat dilihat pada tabel 6.6.

Tabel 6.6 Proses Pengujian A-MDL-AI-01

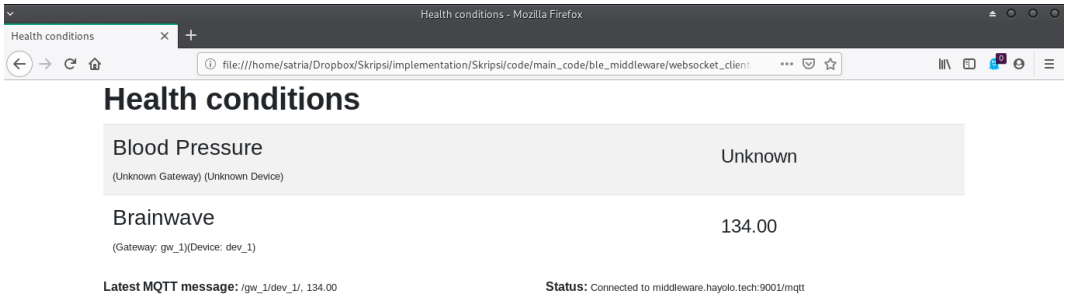
Kode	A-MDL-AI-01
Nama fungsi	*Stream access interface* dapat menerima koneksi dan menyediakan data bagi *client application* yang melakukan *subscribe* pada MQTT *Over Websocket*
Prosedur pengujian	1. BLE *Server* pada satu atau lebih perangkat sensor berjalan dan menerima koneksi 2. Menjalankan *cloud messaging service* dengan konfigurasi mosquitto *over websocket* 3. Menjalankan *websocket client* application 4. Menjalankan *service unit device management* pada *middleware gateway* 5. Melakukan validasi terhadap fungsi *middleware stream interface* dalam menerima koneksi dan menyediakan data ke *client* dengan mengecek *output* dari *websocket client application*.
Hasil yang diharapkan	*Stream access interface* berhasil menerima koneksi dan meneruskan data ke *websocket client application*.

Pengujian pada A-MDL-AI-01 dibuktikan dengan keberhasilan *stream access interface* untuk menerima koneksi dan meneruskan data dari perangkat sensor ke *client application*.



Gambar 6.6 Output Client Application Pada Stream Access Interface

Gambar 6.6 menampilkan *websocket client application*. Pada *client application* terlihat bahwa status koneksi sedang terhubung dengan *stream access interface* pada *port* 9001. Begitu juga dengan pesan yang diterima, terlihat bahwa *client application* menerima data dari perangkat sensor berupa *value* bernilai 107.00



Gambar 6.7 Output Client Application Pada Stream Access Interface (lanjutan)

Stream access interface melayani *client application* untuk menerima data secara stream. Oleh karena itu pada gambar 6.7 terlihat client app menerima value 134.00 yang merupakan value dari perangkat sensor. Pada output terlihat bahwa value telah berubah secara stream begitu data dikirim tanpa melakukan request berulang-ulang. Dapat disimpulkan bahwa fungsional stream access interface adalah valid.

6.2.6 A-MDL-AI-02

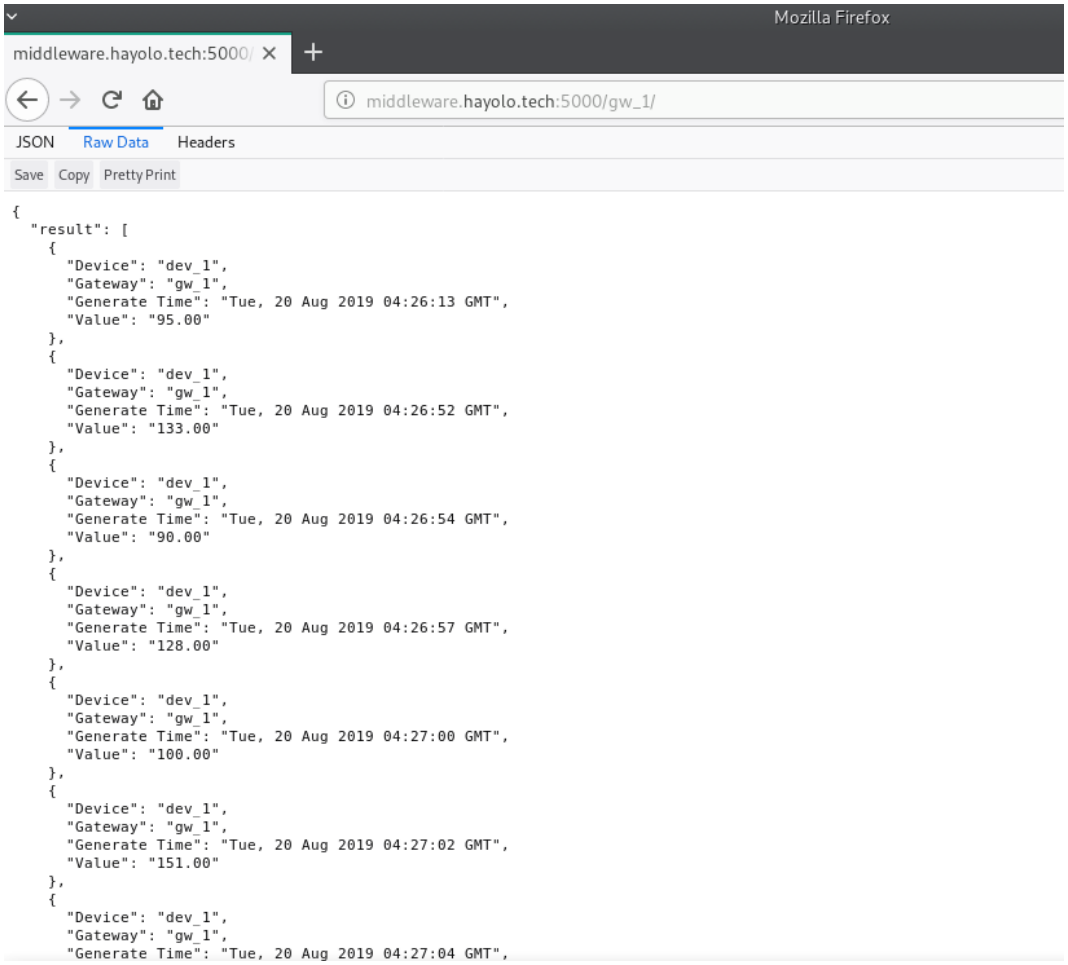
A-MDL-AI-02 bertujuan untuk menguji kemampuan *batch access interface* dalam menangani *request*, mengakses *database* dan menyediakan data yang di *request* oleh *client application*. Proses pengujian A-MDL-AI-02 dapat dilihat pada tabel 6.7.

Tabel 6.7 Proses pengujian A-MDL-AI-02

Kode	A-MDL-AI-02
Nama fungsi	*Batch Access Interface* dapat menerima

	request, mengakses mongo *database* dan menyediakan data yang di *request client* .
Prosedur pengujian	1. Menjalankan *database* mongodb 2. Menjalankan *webservice* API 3. Mengakses API melalui *browser* pada URI http://middleware.hayolo.tech/gw_1/, http://middleware.hayolo.tech/gw_1/dev_1/, http://middleware.hayolo.tech/gw_1/dev_2/ 4. Melakukan *validasi* terhadap fungsi *middleware batch interface* dalam menerima koneksi, mengakses *database* dan menyediakan data untuk *request client* dengan mengecek output dari akses browser ke API.
Hasil yang diharapkan	*Batch Access Interface* berhasil merespon *request* dengan data yang diambil dari *database* sesuai *request client application*.

Pengujian pada A-MDL-AI-01 dibuktikan dengan keberhasilan stream access interface untuk menerima koneksi dan meneruskan data dari sensor device ke client app.



Gambar 6.8 Hasil *request* pada *Batch Access Interface*

Gambar 6.8 menunjukkan hasil *output* yang merupakan respon dari *request* yang dilakukan pada *batch access interface*. Pada *output* tersebut terlihat *payload value*, *waktu* dan identitas dari *gateway* serta perangkat yang menghasilkan data tersebut. Data tersebut memiliki struktur yang sama dengan data yang terdapat pada mongodb. Hal ini menunjukkan bahwa *batch access interface* berhasil memberikan respon kepada *client* dengan data yang terdapat pada mongodb.

6.3 Pengujian Kinerja

Kesesuaian fungsionalitas *middleware* telah divalidasi melalui pengujian fungsional pada sub bab pengujian fungsional. Pengujian selanjutnya adalah kinerja *middleware* yang bertujuan untuk memastikan middleware telah bekerja dengan baik pada beban tertentu. Parameter kinerja yang diuji pada penelitian ini adalah kinerja dari *batch interface*, *stream interface*, dan *concurrent user* pada

keduanya. Metrik pengukuran pada pengujian ini adalah *throughput*, *end-to-end delay*, *latency*, dan *concurrent user*. Proses dan hasil pengujian kinerja akan diuraikan pada sub bab pengujian kinerja.

6.3.1 Skenario Pengujian Kinerja

Merupakan pengujian yang dilakukan untuk mengetahui besaran dari kinerja *middleware* saat beroperasi untuk memastikan bahwa *middleware* telah berjalan dengan baik. Pengujian *kinerja* dilakukan pada *batch access interface* dan *stream access interface*, berikut skenario pengujian yang ditampilkan pada tabel 6.8.

Tabel 6.8 Skenario pengujian kinerja

Kode	Deskripsi	Skenario

PK-1	Pengujian *batch access interface* pada penarikan *data historis* dengan variasi ukuran *data* untuk mengukur nilai *throughput* dan *latency*.	• Mengkonfigurasi *batch interface* untuk menyediakan data dengan variasi ukuran sebesar 0.5, 1, 5, dan 10 MB
		• Membuat konfigurasi pengujian pada JMETER dengan detail sebagai berikut:
		  ○ Terdapat 50 *user thread* yang mengambil data dengan ukuran 0.5, 1, 5, dan 10 MB pada *batch interface*.
		  ○ Menyediakan *listener* untuk merekam nilai *throughput* dan *latency* dari hasil pengujian
		• Menjalankan *batch access interface* dengan konfigurasi yang sesuai.
		• Melakukan pengujian pada JMETER dengan pengujian yang sudah dikonfigurasi.

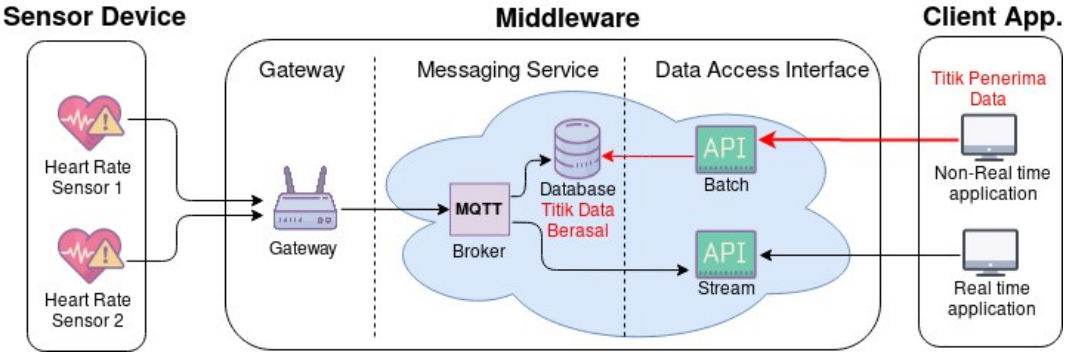
PK-2	Pengujian *stream access interface* pada protokol HTTP dan MQTT untuk mengukur *end-to-end delay* pengiriman data.	• Menghidupkan layanan *stream access interface* pada *middleware cloud*.
		• Membuat konfigurasi pengujian pada JMETER dan script pengujian dengan ketentuan sebagai berikut:
		  ○ Terdapat satu *thread* yang mengambil data dari *stream access interface* dengan *host* middleware.hayolo.tech.
		  ○ Pengambilan data dijalankan secara sinkron dengan variasi *interval* penarikan data 1, 1.5, dan 2 detik.
		  ○ *Looping* penarikan data sebanyak 5x.
		  ○ Menyimpan *timestamp* pesan pada saat di *publish* dan diterima.
		• Menjalankan *publisher* dari gateway yang mempublish data ke *broker* pada middleware.hayolo.tech. Dengan interval 1, 1.5, dan 2 detik.
		• Menjalankan pengambilan data pada *stream interface* menggunakan HTTP dan MQTT.
		• Melakukan analisis terhadap hasil pengujian menggunakan untuk melihat nilai *end-to-end delay* pada masing-masing *interval* dan protokol.

PK-3	Pengujian *concurrent user* untuk mengukur jumlah koneksi yang dapat ditangani oleh *batch* dan *stream access interface* berdasarkan *nilai throughput*.	• Menyalakan layanan *batch* dan *stream access interface* pada *middleware cloud*.
		• Melakukan konfigurasi pengujian pada JMETER dengan ketentuan sebagai berikut:
		  ○ Terdapat user sejumlah 500, 1000, 1500, dan 2000 thread yang mengakses *batch* dan *stream access interface* pada middleware.hayolo.tech
		  ○ Akses dijalankan secara asinkron dengan interval antar *user thread* sebesar 0.1 detik.
		  ○ *Looping* percobaan sebanyak 3x.

		○ Menambahkan *listener* yang merekam nilai *throughput* pada masing-masing variasi *user thread* dan percobaan. • Melakukan pengaksesan *batch* dan *stream access interfaces* menggunakan JMETER dengan konfigurasi yang telah dibuat. • Melakukan analisis terhadap nilai *throughput* yang dihasilkan *listener* JMETER.

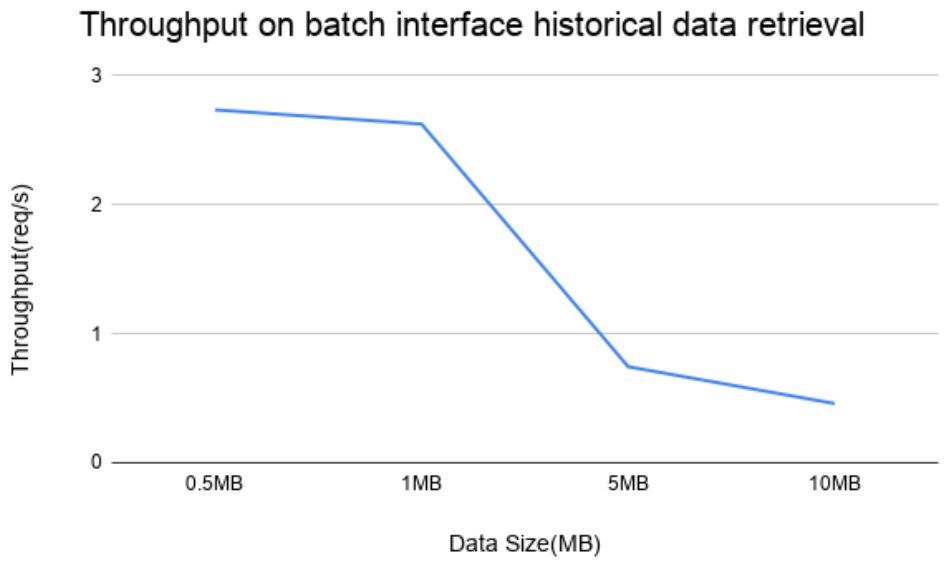
6.3.2 Pengujian *Batch Access Interface*

Batch access interface dikembangkan untuk memfasilitasi penarikan data-data lama yang bersifat historis. Sehingga pengujian *batch access interface* berfokus dalam kinerja pada penarikan data historis dengan ukuran 0,5 MB, 1 MB, 5 MB, dan 10 MB oleh 50 *concurrent user*. Pengujian batch interface terjadi antara client application, batch interface, dan database seperti yang ditunjukkan pada gambar 6.9.



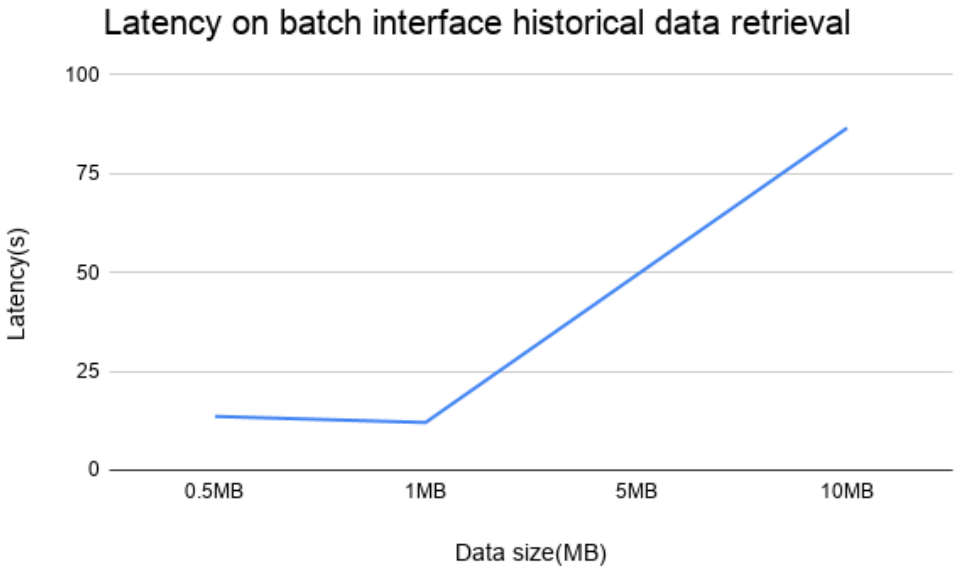
Gambar 6.9 Titik Pengujian Batch Access Interface

Diperoleh hasil *throughput* pada pengujian *batch access interface* seperti yang ditunjukkan pada gambar 6.10.



Gambar 6.10 *Throughput* Rata-rata pada Pengujian Batch Access Interface

Gambar 6.10 menunjukkan nilai *throughput* yang dihasilkan pada penarikan data dengan ukuran 0,5 MB yakni sebesar 2,73 *request* per detik. Pada penarikan data berukuran 1 MB *throughput* yang didapat adalah 2,62 *request* per detik. *Throughput* menurun drastis pada penarikan data dengan ukuran 5 MB menjadi 0,74 *request* per detik dan 10 MB menjadi 0,45 *request* per detik. Besarnya ukuran data yang diambil menyebabkan nilai *throughput* yang dihasilkan menurun.



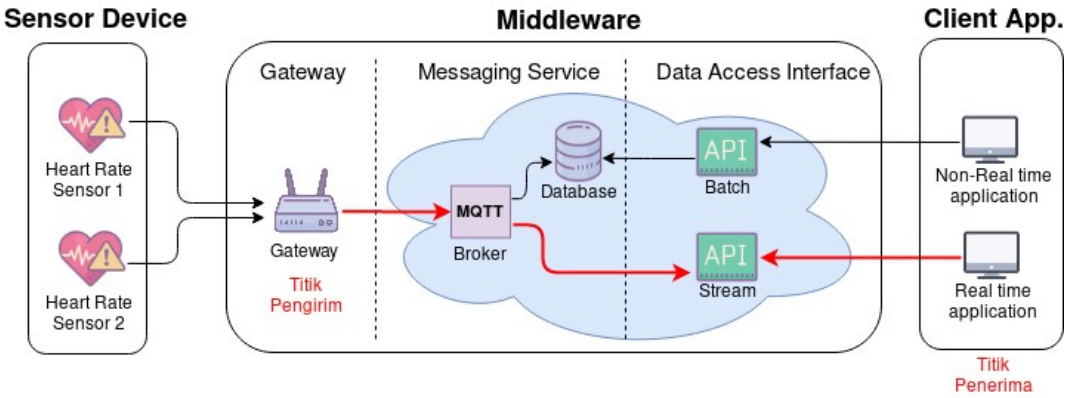
Gambar 6.11 *Latency* Rata-rata pada Pengujian *Batch Access Interface*

Gambar 6.11 menunjukkan *latency* rata-rata yang dibutuhkan pada penarikan data oleh 50 *concurrent user* pada variasi ukuran data. Penarikan data berukuran 0,5 MB oleh 50 *concurrent user* memakan waktu sebanyak 13,61 detik, lalu pada

1 MB memakan waktu sebanyak 12 detik. Selisih waktu penarikan data berukuran 0,5 MB lebih tinggi daripada 1 MB, hal ini disebabkan oleh jaringan yang tidak stabil pada penarikan data 1 MB. Kemudian penarikan data berukuran 5 MB oleh 50 *concurrent user* memakan waktu selama 49,32 detik, sedangkan penarikan data berukuran 10 MB memakan waktu rata-rata yang lebih lama yakni 86,58 detik. Kesimpulan dari pengujian *batch access interface* adalah semakin besar ukuran data maka waktu akses yang dibutuhkan menjadi lebih lama. Berbanding terbalik dengan nilai *throughput* jika semakin besar ukuran data, maka semakin kecil nilai *throughput* yang dihasilkan.

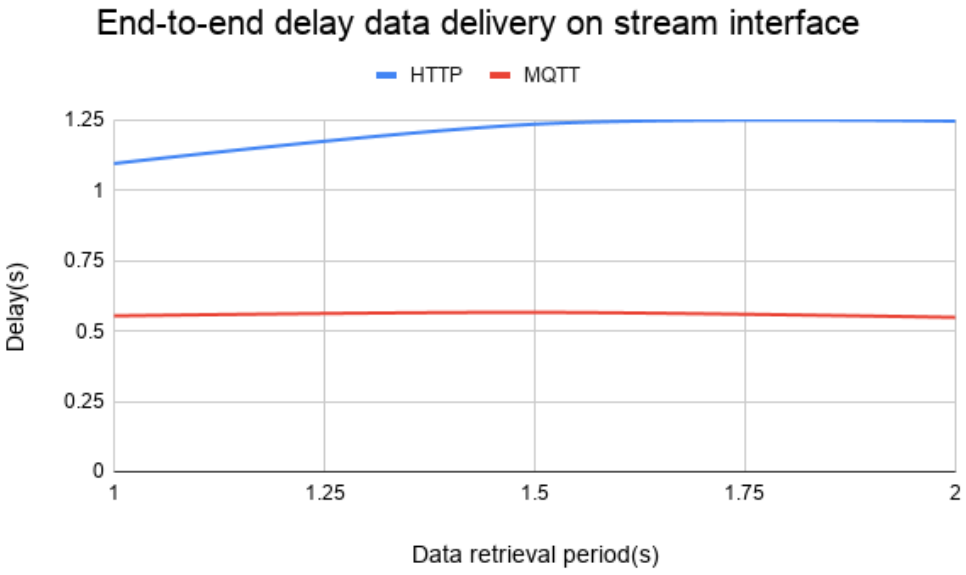
6.3.3 Pengujian *Stream Access Interface*

Stream access interface dikembangkan untuk pengolahan data yang membutuhkan data terbaru secepat mungkin setelah data tersebut dihasilkan. Oleh karena itu pengujian *stream access interface* akan berfokus pada *end-to-end delay* pengiriman yakni waktu yang dibutuhkan sebuah data yang dikirim untuk sampai ke tujuan.



Gambar 6.12 Titik pengujian end-to-end delay

Pengujian *end-to-end delay* dilakukan pada *stream access interface* menggunakan layanan HTTP dan MQTT, terhitung ketika *gateway* mengirimkan data hingga data tersebut diterima oleh *client application*. Pada gambar 6.12 dijelaskan komponen yang merupakan titik terjadinya pengujian. Nilai end-to-end delay didapatkan dari pengurangan waktu ketika pesan di publish oleh titik pengirim dan ketika pesan diterima oleh titik penerima. Hasil pengujian dijelaskan pada gambar 6.13.

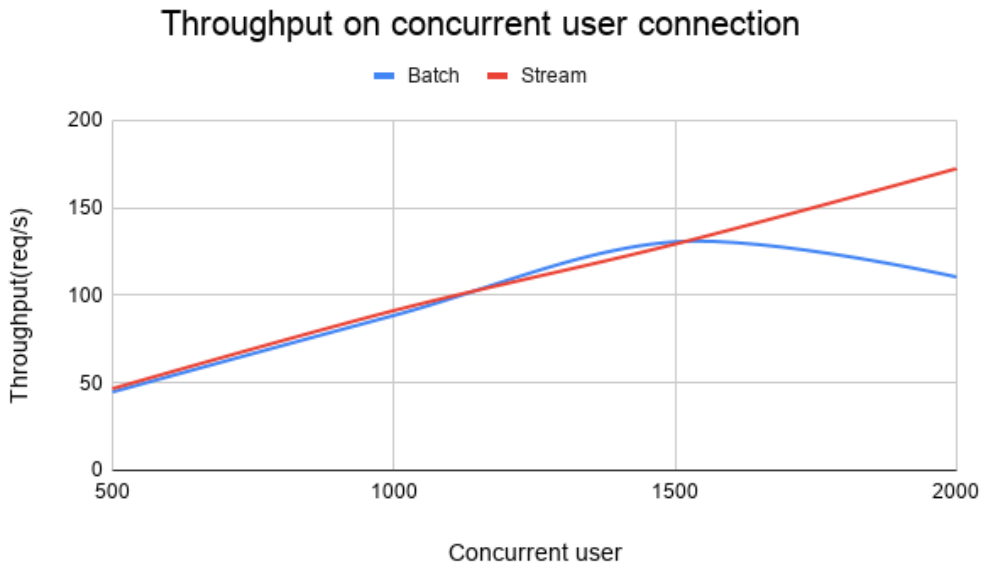


Gambar 6.13 Delay pengiriman data pengujian end-to-end delay

Gambar 6.13 menunjukkan nilai *end-to-end delay* pengiriman dan penarikan pada interval 1, 1.5, dan 2 detik. Grafik tersebut membandingkan nilai *delay* dari protokol HTTP dan MQTT dalam penarikan pada *stream access interface*. Pada *interval* satu detik *delay* protokol HTTP bernilai 1.09 detik dan delay protokol MQTT bernilai 0.55. Pada *interval* 1.5 detik didapat *delay* penarikan sebesar 1.23 detik pada HTTP dan 0.56 pada MQTT. Pada interval dua detik didapatkan *delay* penarikan pada HTTP sebesar 1.24 detik dan pada MQTT sebesar 0.54 detik. Penarikan data menggunakan HTTP memiliki *delay* yang lebih tinggi dikarenakan mekanisme *3-way-handshake* yang dilakukan berulang pada tiap penarikan. Sementara MQTT hanya membutuhkan sekali *3-way-handshake* dalam penarikan data dan penerimaan data secara *real-time*.

6.3.4 Pengujian *Concurrent User*

Pengujian ini dilakukan untuk menguji kinerja dari kedua *data access interface* dalam melayani *user*. *Metric* yang digunakan pada pengujian ini adalah *throughput* dengan satuan request per detik, semakin tinggi throughput yang dihasilkan semakin baik kinerja yang *middleware*. Pengujian *concurrent user* dilakukan pada *batch* dan *stream access interface* dengan variasi jumlah *user* sebanyak 500, 1000, 1500 dan 2000. Hasil pengujian *concurrent user* ditunjukkan pada gambar 6.12



Gambar 6.14 Nilai throughput pada pengujian concurrent user

Gambar 6.14 menunjukkan nilai rata-rata *throughput* dari tiga kali percobaan pengujian *throughput* dengan variasi *user* 500 hingga 2000. Pada variasi *user* 500 stream *interface* memiliki nilai *throughput* yang lebih tinggi dari *batch* dengan nilai 46,7 *request* per detik dan batch dengan nilai 44,8 *request* per detik. Serupa dengan variasi user 1000 stream sedikit unggul dengan nilai 91 *request* per detik dan batch 88 *request* per detik. Pada variasi 1500 *batch* unggul dengan nilai 130 *request* per detik dan *stream* 129 *request* per detik, namun pada variasi user 2000 *stream* jauh lebih unggul dengan nilai 172 *request* per detik dan *batch* 110 *request* per detik. Terdapat error pada *request* percobaan ketiga pengujian throughput dengan variasi user 2000 yang menyebabkan nilai throughput *batch* turun dan jauh dari nilai *stream* yang semakin naik. Dari hasil pengujian *concurrent user* dapat disimpulkan bahwa koneksi dari *stream interface* lebih stabil dibandingkan oleh *batch interface*.

BAB 7 PENUTUP

Bab penutup menjelaskan uraian kesimpulan dan saran berdasarkan analisis kebutuhan, perancangan, implementasi dan pengujian yang telah dilakukan.

7.1 Kesimpulan

Berdasarkan analisis kebutuhan, perancangan, implementasi, dan pengujian yang telah dilakukan ditarik sebuah kesimpulan yang menjawab rumusan masalah sebagai berikut:

1. Penanganan perangkat sensor heterogen dengan melakukan abstraksi WoT menggunakan *middleware* berbasis *cloud* dapat dilakukan. Abstraksi dilakukan oleh *middleware* dengan mengubah data yang dihasilkan perangkat sensor menjadi sebuah *web resource* yang dapat diakses oleh *client application* pada *data access interface* yang difasilitasi *middleware*.

2. Pemfasilitasan akses bagi aplikasi yang membutuhkan pengolahan data secara *batch* dan *stream* dapat dilakukan oleh middleware dengan menyediakan dua antarmuka akses pada satu *middleware*. Penyediaan akses data secara *batch* dapat dilakukan dengan menerapkan abstraksi perangkat sensor kedalam bentuk URL pada penggunaan RESTful *webservice*. Sedangkan penyediaan akses data secara stream dapat dilakukan dengan menerapkan abstraksi perangkat sensor kedalam bentuk *topic* pada penggunaan MQTT *over websocket*.

3. Dari pengujian dapat disimpulkan bahwa *middleware* telah bekerja sesuai dengan fungsional yang diharapkan. Kesimpulan terhadap kinerja pada *middleware* dibagi menjadi beberapa parameter yakni:

    a. Pada pengujian *batch access interface* nilai *throughput* dan *latency* ditentukan oleh besar ukuran data yang diambil. Semakin besar ukuran data yang diambil akan menyebabkan *throughput* menurun dan *latency* meningkat. *Throughput* tertinggi terdapat pada penarikan data berukuran 0,5 MB yaitu 2,7 *request/s*. dan *latency* tertinggi pada data berukuran 10 MB yaitu 86,58 detik.

    b. Pada pengujian *stream access interface* nilai *end*-to-*end delay* pada penarikan data menggunakan HTTP dan MQTT memiliki perbedaan yang signifikan. Perbedaan ini disebabkan oleh penarikan data menggunakan HTTP memiliki *delay* yang lebih tinggi sebesar 1,19 detik dikarenakan mekanisme *3-way-handshake* di setiap penarikan. Sementara MQTT memiliki delay yang lebih rendah sebesar 0,55 detik karena hanya membutuhkan sekali *3-way-handshake* dalam penarikan data.

c. Pada pengujian *concurrent user* disimpulkan bahwa koneksi dari *stream interface* lebih stabil dibandingkan *batch interface*. Hal tersebut dapat terlihat pada perbedaan nilai *throughput* antara *batch* dan *stream* saat *concurrent user* berjumlah 2000. Dimana pada jumlah *user* tersebut *batch interface* memiliki nilai *throughput* sebesar 110 *request* per detik dan *stream interface* sebesar 172 *request* per detik. Perbedaan *throughput* yang signifikan disebabkan oleh *error* pada *request client* ke *batch interface*.

7.2 Saran

Berdasarkan penelitian yang sudah dilakukan, dihasilkan beberapa saran yang dapat dilakukan pada penelitian berikutnya yaitu:

1. Middleware dapat dikembangkan lebih lanjut melalui implementasi mekanisme keamanan pada komponen cloud seperti, batch access interface dan stream access interface.
2. Dapat dilakukan pengembangan fitur manajemen device otomatis pada *sensor-to-cloud gateway* untuk perangkat dengan protokol BLE (Bluetooth Low Energy), dalam hal ini fitur tersebut akan membuat *middleware* lebih mudah dalam pencarian perangkat sensor.
3. Dapat dikembangkan sebuah abstraksi protokol selain BLE pada perangkat sensor seperti MQTT, LoRa, ZigBee, 802.11 dan sebagainya untuk menangani aspek interoperabilitas pada *middleware*.
4. Perlu dilakukan analisis pada middleware secara detail untuk membandingkan middleware ini dengan middleware lainnya baik dari segi keamanan, kinerja, fungsionalitas, interoperabilitas dan lain sebagainya.

DAFTAR REFERENSI

Al-fuqaha, A., Member, S., Guizani, M., Mohammadi, M., & Member, S., 2015. Internet of Things : A Survey on Enabling, 17(4), 2347–2376.

Atzori, L., Iera, A., & Morabito, G., 2010. The Internet of Things: A survey. Computer Networks, 54(15), 2787–2805.

Baker, S. B., Xiang, W., & Atkinson, I., 2017. Internet of Things for Smart Healthcare: Technologies, Challenges, and Opportunities. IEEE Access, 5, 26521–26544.

Bhawiyuga, A., Kartikasari, D. P., & Pramukantoro, E. S., 2017. A publish subscribe based middleware for enabling real time web access on constrained device, *2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Phuket, 2017, pp. 1-5.

Bluetooth., 2003. *Bluetooth technology*. [online] Bluetooth SIG. Tersedia di: <https://www.bluetooth.com/bluetooth-technology/radio-versions> [Diakses 4 Maret 2019]

Eclipse Foundation., 2014. *MQTT and CoAP, IoT Protocols*. Dipetik April 10, 2016, dari Eclipse Foundation: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

Fielding, Roy T., Taylor, Richard N., 2002. "Principled Design of the Modern Web Architecture" ACM Transactions on Internet Technology (TOIT), New York: Association for Computing Machinery. 115–150, doi:10.1145/514183.514185, ISSN1533-5399

Guinard, D., Trifa, V., Mattern, F., & Wilde, E., 2011. From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. Architecting the Internet of Things.

Gupta, P., Agrawal, D., Chhabra, J., & Dhir, P. K., 2016. IoT based smart healthcare kit. In 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT) (pp. 237-242). IEEE.

Hillar, G. C., 2017. *MQTT Essential - A Lightweight IoT Protocol*. Birmingham: Packt Publishing Ltd.

HiveMQ., 2015. *MQTT Essential – Part 6: Quality of Service 0, 1 & 2*. Dipetik April 10 , 2019, dari HiveMQ: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

HiveMQ., 2015. *MQTT Essential Special: MQTT over WebSockets*. Dipetik April 10 , 2019, dari HiveMQ: https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets/

Islam, S. M. R., Kwak, D., Kabir, M. H., Hossain, M., & Kwak, K. S., 2015. The internet of things for health care: A comprehensive survey. IEEE Access, 3, 678–708.

Khan, R., Khan, S. U., Zaheer, R., & Khan, S., 2012. Future internet: the internet of things architecture, possible applications and key challenges. In *2012 10th international conference on frontiers of information technology* (pp. 257-260). IEEE.

Kumari, S. & Rath, S. K., 2015. Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration, *2015 International Conference on Advances in Computing*, *Communications and Informatics (ICACCI)*, Kochi, 2015, pp. 1656-1660.

Pang, Z., 2013. *Technologies and Architectures of the Internet-of-Things (IoT) for Health and Well-being*. Doctoral dissertation. KTH Royal Institute of Technology. Tersedia di <https://pdfs.semanticscholar.org/222d/206e8fc758c19ac06680db61a555fd6b71ed.pdf> [Diakses 2 September 2019]

Philip, N., et al., 2014 ”Design of a RESTful middleware to enable a web of medical things.” Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on. IEEE.

Pramukantoro, E. S., Yahya, W. & Bakhtiar, F. A., 2017. *Performance Evaluation of IoT Middleware for Syntactical Interoperability.* s.l., IEEE.

Raggett, D., 2015. The Web of Things: Challenges and Opportunities. Computer, 48(5), 26–32.

Reddy, C.K. dan Aggarwal, C.C. eds., 2015. Healthcare Data Analytics. BocaRaton, FL:CRC Press

Shahrivari, S., 2014. Beyond Batch Processing: Towards Real-Time and Streaming Big Data. Computers, [e-journal] 3(118). Tersedia melalui: <https://www.mdpi.com/2073-431X/3/4/117/pdf> [Diakses 14 Oktober 2019]

Townsend, K., et al., 2014. Getting Started with Bluetooth Low Energy. Sebastopol,CA: O’Reilly Media, Inc.

Web of Things (WoT) Architecture - W3C on GitHub., 2019. Web of Things(WoT) Architecture. Dipetik Maret 5, 2019, dari W3C on GitHub: https://w3c.github.io/wot-architecture/#sec-wot-architecture