Perancangan dan Simulasi Proses Antrean Data Multisensor Untuk Sistem Telemonitoring Multikontrol Berbasis Internet of Things

Abdul Hanif Al Baaits[1*], Wahyu Kusuma Raharja[2]

[1,2]Program Studi Magister Teknik Elektro, Universitas Gunadarma, Depok, Jawa Barat
Email: [1*]hanifalbaits@gmail.com, [2]wahyukr@staff.gunadarma.ac.id

(Naskah masuk: 17 Okt 2021, direvisi: 14 Feb 2022, 4 Mar 2022, diterima: 08 Mar 2022)

Abstrak

Internet of things (IoT) tidak terlepas dari perangkat yang memiliki fungsi dan tujuan yang saling bekerja sama dan berintegrasi secara tepat dan cepat melalui internet. Sistem *monitoring* yang selalu menampilkan data secara *realtime* memerlukan data yang banyak setiap detiknya. Pemrosesan ini membutuhkan suatu sistem yang dapat menangani permintaan data yang banyak dan juga penyimpanan ke dalam *database*. Penelitian ini bertujuan untuk menghasilkan rancangan dan mengimplementasikan *ActiveMQ* untuk antrean data serta membandingkan dua buah metode pengambilan data *Realtime Polling* dan *Realtime Handshake* pada sistem *telemonitoring multikontrol* berbasis *internet of things*. Hasil penelitian ini diharapkan dapat membantu menyelesaikan permasalahan permintaan data sensor yang banyak agar dapat menyelesaikan secara tepat dan cepat. Sistem *monitoring* secara *realtime* ditampilkan dalam bentuk *website*. Hasil Penelitian ini berhasil mengimplementasikan *ActiveMQ* untuk antrean data dengan tingkat keberhasilan data sukses diterima sebesar 96%, dibandingkan dengan tidak menggunakan antrean dengan tingkat keberhasilan data sukses diterima sebesar 83%. Metode *Realtime Handshake* jauh lebih tepat dan cepat dengan waktu tunda antara permintaan data masuk hingga di tampilkan sebesar 63,6 ms, dibandingkan dengan metode *Realtime Polling* dengan waktu tunda antara permintaan data masuk hingga ditampilkan sebesar 669,95 ms.

Kata Kunci: Implementasi *ActiveMQ*, Antrean Data, *Realtime Polling*, *Realtime Handshake*, *Internet Of Things*, *API*.

The Design and Simulation of Multisensor Data Queue Processes for Multicontrol Telemonitoring Systems Based on The Internet Of Things

Abstract

Internet of things (IoT) cannot be separated from devices that have functions and purposes that work together and integrate appropriately and quickly through the internet. A monitoring system that always displays data in real time requires a lot of data every second. This processing requires a system that can handle a lot of data requests and also store it in a database. This study aims to design and implement ActiveMQ for data queuing and to compare two methods of data retrieval for Realtime Polling and Realtime Handshake in a multi-control telemonitoring system based on the internet of things. The results of this study are expected to help solving the problem of requesting a lot of sensor data in order to be able to solve it accurately and quickly. The monitoring system in real time is displayed in the form of a website. The results of this study succeeded in implementing ActiveMQ for data queues with a success rate of 96% of successful data being received, compared to not using a queue with a success rate of 83 % of successful data being received. The Realtime Handshake method is much more precise and faster with a delay time between incoming data requests and being displayed of 63.6 ms, compared to the Realtime Polling method with a delay time between incoming data requests and being displayed at 669.95 ms.

Keywords: *ActiveMQ Implementation, Data Queue, Realtime Polling, Realtime Handshake, Internet Of Things, API.*

I. PENDAHULUAN

Di era globalisasi saat ini, perkembangan ilmu pengetahuan dan teknologi sangat pesat, khususnya perkembangan internet. Oleh karena itu dunia perekonomian pun tidak lepas dari perkembangan internet. Dengan berkembangnya *Internet of Things* (*IoT*), maka internet pun bisa dimanfaatkan untuk keperluan lain yang mendukung perekonomian, diantaranya yaitu dengan memanfaatkannya sebagai aplikasi *real time*, seperti *website*. Hal tersebut sejalan dengan peta jalan dan strategis Indonesia untuk menerapkan revolusi industri 4.0. Peta jalan yang diberi nama peta jalan *Making* Indonesia 4.0 itu memberikan arah bagi pergerakan industri nasional di masa depan [1], dimana salah satu aspek dalam era ini adalah *internet of things*.

Internet of things tidak terlepas dari perangkat yang memiliki fungsi dan tujuan yang saling bekerja sama dan berintegrasi secara tepat dan cepat melalui internet. Semakin banyak sensor yang bekerja dalam suatu sistem, maka semakin banyak juga data yang dikirimkan setiap detiknya. Sistem *monitoring* yang selalu menampilkan data secara *realtime* memerlukan data yang banyak setiap detik nya. Pemrosesan ini membutuhkan suatu sistem yang dapat menangani permintaan data yang banyak dan juga penyimpanan ke dalam *database*.

Proses pengambilan data sensor sangat berpengaruh dalam mengambil tindakan dan keputusan. Data sensor yang banyak disimpan dan dijadikan pusat data atau *big data* dalam *database*. Permintaan data yang banyak oleh sensor melalui mikrokontroler akan mengakibatkan sistem kebanjiran permintaan dan kegagalan fatal dalam penerimaan data, atau disebut serangan *DDoS (Distributed Denial of Service).* Sehingga sistem akan menolak permintaan data dari mikrokontroler dan banyak data yang terbuang. Pada umumnya setiap aplikasi menggunakan *API (Application Programming Interface)* untuk terhubung dengan sumber data, layanan pihak ketiga atau aplikasi lainnya [2]. Sehingga, penelitian ini menggunakan *REST API* untuk berkomunikasi dan berintegrasi dengan perangkat lainnya. *REST API* adalah gaya arsitektur untuk merancang layanan *web*, yang dapat digunakan dari klien yang berbeda menggunakan panggilan *HTTP* sederhana [3]. Penelitian ini menggunakan *Spring Boot* untuk membuat layanan *API*. *Spring Boot* adalah kerangka kerja dalam membuat layanan *API web* yang menggunakan bahasa pemrograman *Java*.

Sistem antrean data dengan *ActiveMQ* akan digunakan untuk menangani permasalahan banyak permintaan data oleh sensor-sensor perangkat *IoT* melalui mikrokontroler. Setiap data yang diterima oleh sistem dikirim ke dalam antrean. Setelah itu terdapat sistem *consumer listener* yang mengambil antrean data tersebut. *ActiveMQ* adalah produk implementasi *JMS* dengan kinerja tinggi dan berdasarkan sumber terbuka dari protokol *Apache*, dan mesin *JMS open source* terbaik dengan kematangan yang terbukti dan fitur yang kaya [4].

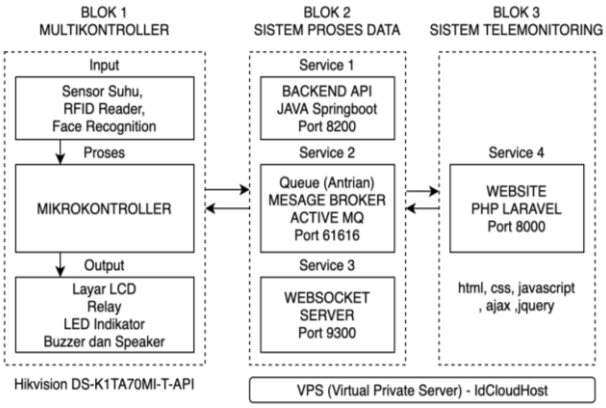
Penelitian ini akan membandingkan dua buah metode untuk menampilkan data secara *realtime*. Pertama, metode *polling* yaitu *web* melakukan permintaan data secara terus menerus secara selang waktu satu detik, sehingga data *web* akan berubah secara *realtime* setiap satu detik. Kedua, metode *handshake* yaitu *web* dan *service backend* akan melakukan koneksi secara terus menerus seperti berjabat tangan sehingga proses perubahan data akan dipicu oleh setiap data yang diterima di *service backend*.

Hasil penelitian ini diharapkan dapat membantu menyelesaikan permasalahan permintaan data sensor yang banyak agar dapat menyelesaikan secara tepat dan cepat dalam memproses tampilan *realtime*. Sistem *monitoring* secara *realtime* ditampilkan dalam bentuk *website* dan terintegrasi dengan sistem *API* ke perangkat *IoT* (*Internet of Things*) lainnya.

II. METODOLOGI PENELITIAN

A. Rancangan Arsitektur

Arsitektur yang digunakan dalam perancangan sistem ini terbagi menjadi 3 bagian besar. Blok pertama adalah sistem multikontroler yaitu inputan yang diterima oleh sistem ini. Blok kedua adalah sistem yang menangani data masuk dari perangkat alat dan antrean data serta pengaturan data yang masuk ke dalam *database*. Pada blok kedua memiliki 3 buah sistem. Sistem pertama yaitu *backend API Springboot Java* yang menerima data dari perangkat alat, tempat integrasi *API* dari perangkat *IoT* ataupun *platform* lain nya dan penyimpanan data ke dalam *database*. Sistem kedua adalah *activeMQ* yaitu *message broker* sistem antrean yang menampung data sensor dari *backend API* untuk dikembalikan ke dalam konter yang menangani pemrosesan data selanjutnya. Sistem ketiga adalah *websocket* yang membuat layanan komunikasi dua arah layaknya seperti ruang obrolan (*chat room)* atau panggilan telepon dimana antara *client* terkoneksi secara langsung ke dalam *web socket*. *Websocket* adalah standar baru untuk komunikasi *full-duplex* (dua arah secara bersamaan) sehingga komunikasi yang terjadi antara *client* dan *server* lebih *real-time* [5]. Gambar 1 merupakan diagram dari rancangan arsitektur penelitian ini.



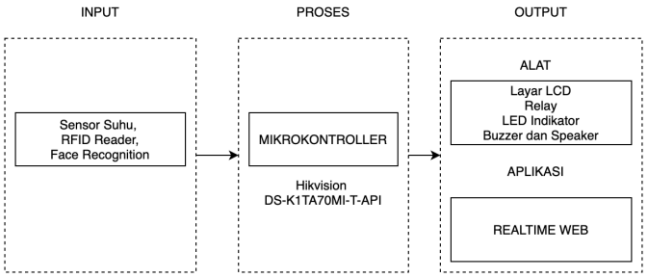
Gambar 1. Rancangan Arsitektur

Blok ketiga adalah sistem *telemonitoring*, digunakan untuk menampilkan data secara *realtime*. Pada blok ketiga terdapat *website* yang menangani tampilan *realtime.* Pada tampilan *realtime* terdapat dua tampilan beranda. Beranda

DOI: 10.34148/teknika.v11i1.418

pertama menampilkan data *realtime* dengan metode *polling* dan beranda kedua menampilkan data *realtime* dengan metode *handshake*.

1. Blok Multikontroler

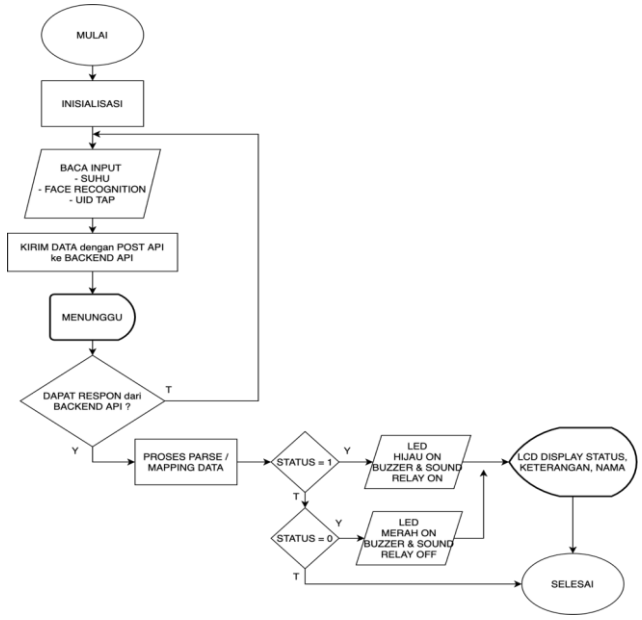


Gambar 2. Blok Diagram Alat

Pada Gambar 2 terdapat blok diagram penelitian ini dimana menggunakan alat *Hikvision DS-K1TA70MI-T*. Alat ini suatu *device* akses pintu yang dapat membaca inputan berupa *face recognition*, nomor identifikasi kartu *RFID*, dan derajat suhu. *RFID* adalah sebuah metode atau teknologi identifikasi berbasis gelombang radio (*radio frequency*). Teknologi ini mampu mengidentifikasi berbagai obyek secara simultan tanpa diperlukan kontak langsung. Simultan mempunyai pengertian bahwa bermacam obyek tersebut diidentifikasi tidak satu persatu sebagaimana dilakukan pada identifikasi terhadap sistem *barcode* [6]. Hasil keluaran dari *Hikvision* adalah layar *LCD* tampilan informasi keterangan tentang balasan status dari *Backend API*. Alat ini memerlukan *service* atau layanan tambahan untuk mengirim data *API* dengan ke *backend API*. Protokol *API* yang digunakan harus sesuai dengan yang diterima oleh *backend API*.

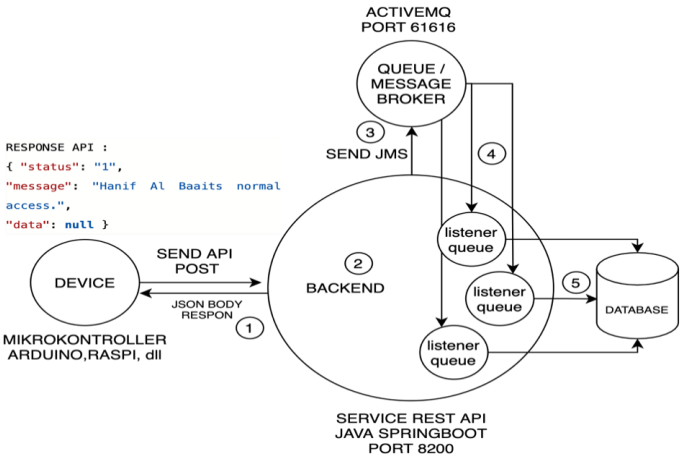
Protokol yang digunakan untuk mengirim data dari alat adalah *POST* dengan *header authorization encrypt SHA 256*. Alamat dari *API* sesuai dengan yang diterima pada sisi *Backend API*. Autorisasi alat ini sesuai dengan data pintu yang terdapat pada *database Backend API*. Bentuk data yang dikirim berupa *form-data* dengan variabel seperti, jenis pintu, kode pintu, kode kartu dan derajat dalam bentuk *celcius*. Variabel jenis pintu digunakan untuk menandakan data masuk dari pintu masuk atau keluar. Kode kartu untuk identifikasi *user* yang *tap*. Derajat celcius akan terbaca ketika *user* melakukan *tap* kartu ataupun melalui *face recognition*.

Pada Gambar 3 adalah diagram alur dari cara kerja sistem alat. Pertama alat akan meninisialisasi keseluruhan variabel-variabel dan *library* (kumpulan fungsi-fungsi) yang digunakan. Setelah itu alat membaca masukan suhu dan *face recognition* ataupun kode kartu. Selanjutnya data akan dikirim ke *backend* melalui *API* dengan *method post*. Sistem akan menunggu hingga permintaan *API* mendapat balasan dari *backend*. Jika alat mendapat balasan dari *backend*, selanjutnya akan mengubah bentuk data agar dapat dibaca. Jika status balasan 1 maka *LED indikator* hijau akan menyala dan relay akan menyala sehingga pintu akses *door* dapat terbuka. Jika status balasan 0 maka *LED indikator* merah akan menyala dan *relay* akan mati sehingga pintu akses *door* tetap tertutup. Selain itu alat juga akan mengeluarkan keterangan pada layar *display* dan bunyi suara.



Gambar 3. Diagram Alur Cara Kerja Alat

2. Blok Sistem Proses Data



Gambar 4. Sistem Kerja Proses Data

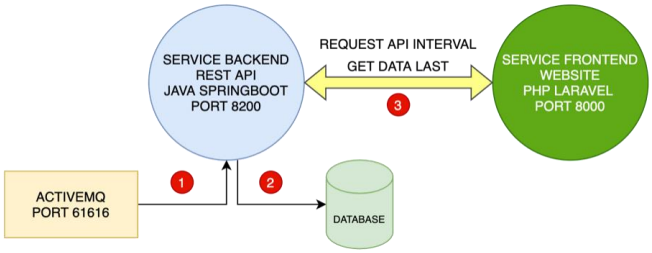
Pada Gambar 4 sistem kerja dimulai dari nomor 1 dimana data akan diterima oleh *backend* pada *port 8200* melalui *API* dengan *method post*. Setelah itu data diproses pada nomor 2 di dalam *backend* untuk menentukan dan validasi apakah pintu tersedia, apakah *autorisasi* pintu benar, apakah *user* terdaftar dan terakhir apakah *user* diberikan akses. Jika keseluruhan data telah diproses maka *backend* akan memberikan respon balasan dengan bentuk *JSON* dengan variabel *status* dan *message*. Setelah itu data dikirimkan ke dalam antrean melalui *JMS (Java Message Service)* pada nomor 3. Data yang diterima akan ditampung di dalam antrean atau *message broker activeMQ* pada *port 61616*. Metode yang digunakan pada *message broker activeMQ* adalah *round robin*, yang artinya konsumen antrean akan

bergantian untuk mengambil dan mengelola data yang datang dari setiap data yang masuk.

3. Blok Sistem *Telemonitoring*

Pada sistem *telemonitoring* penelitian ini menggunakan kerangka kerja atau *framework Laravel 8*. Aplikasi *web* berbasis arsitektur *Laravel* didasarkan pada *multi-layer*. Dalam teknologi *web* struktur *layer* ketiga, *database* bukan layanan langsung ke setiap klien, tetapi menghubungkan ke *web server*, sehingga mencapai layanan informasi pelanggan yang dinamis, *realtime* dan interaktif [7]. Pada *website* ini seluruh data yang diterima melalui API dari *backend API*. Di dalam *website* terdapat beberapa tampilan halaman yang memiliki banyak fungsi dalam proses penelitian ini.

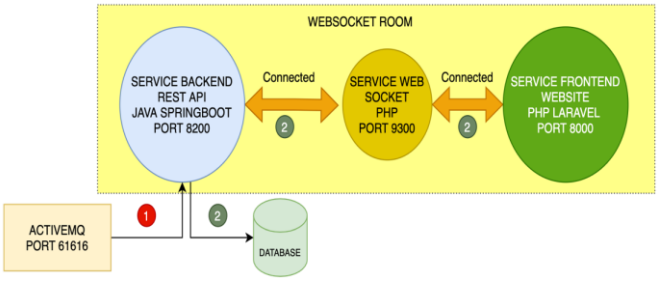
B. Arsitektur Proses *Realtime Polling*



Gambar 5. Arsitektur Model *Realtime Polling*

Pada Gambar 5, nomor 1 adalah data alat dari antrean *activeMQ* diterima dan diambil oleh konsumen antrean yang berada di *backend API*. Setelah itu data akan di masukan ke dalam *database* seperti nomor 2. *Website* akan mengambil secara terus menerus untuk mendapatkan data setiap detik nya pada nomor 3.

C. Arsitektur Proses *Realtime Handshake*

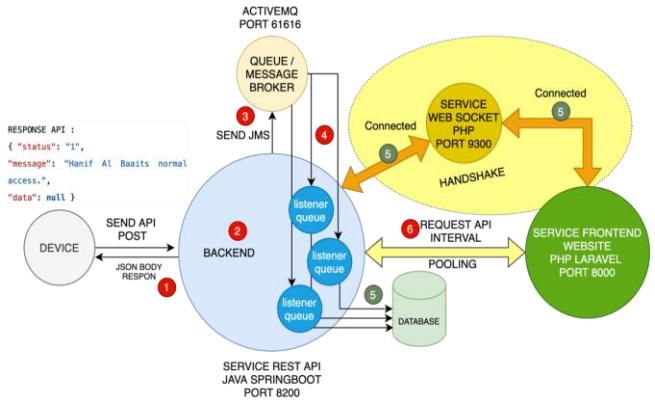


Gambar 6. Arsitektur Model *Realtime Handshake*

Pada Gambar 6, nomor 1 adalah data alat dari antrean *activeMQ* diterima dan diambil oleh konsumen, antrean yang berada di *backend API*. Setelah itu data akan di masukan ke dalam *database* seperti nomor 2. Pada nomor 2 data juga dikirimkan kepada *websocket* dan otomatis sisi *web* juga menerima data yang masuk ke dalam *websocket*. Pada gambar tersebut terdapat kotak kuning yang bernama *websocket room*, yang artinya dua buah aplikasi atau *service* saling mendengarkan dan berkomunikasi layaknya di dalam suatu ruangan. Ruang tersebut dapat dianalogikan seperti *chat room* atau menelepon antara dua klien, sehingga koneksi akan terus berjalan atau disebut *handshake* (jabat tangan).

D. Analisis Secara Detail

Pada bagian ini menjelaskan secara detail cara kerja proses penelitian. Pada Gambar 7 banyak aplikasi yang saling bekerja sama dalam membangun sistem antrean untuk data sensor *IoT*. Perbedaan warna yang menjadikan berbeda fungsinya dan nomor yang tersedia juga termasuk langkah-langkah alur yang di dahulukan dari data datang hingga selesai diproses. Pada nomor 1, data dikirimkan oleh perangkat *IoT* menggunakan *API*.



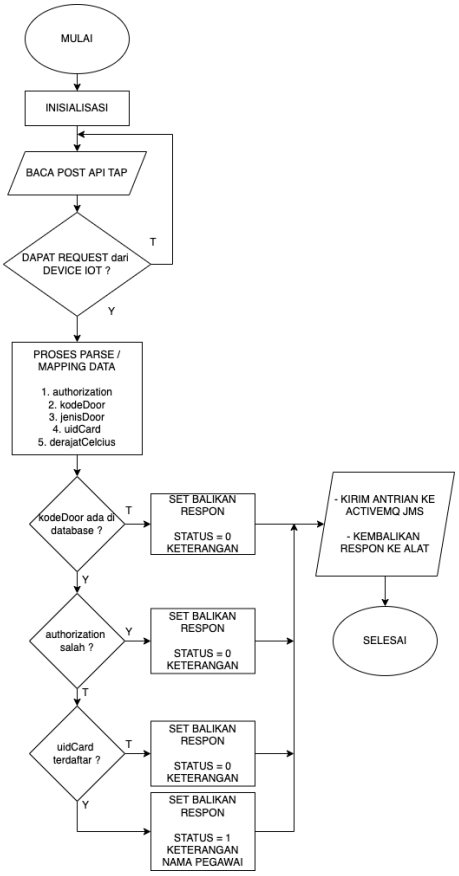
Gambar 7. Arsitektur Secara Detail

Setelah itu data diterima di *backend API* pada nomor 2, di dalam *backend* data akan diproses untuk melakukan validasi yang ada, selanjutnya respon akan dikembalikan kepada perangkat *IoT*. Data yang diterima pada *backend API* langsung diteruskan ke dalam antrean data melalui *JMS (Java Message Service)* , terlihat pada nomor 3.

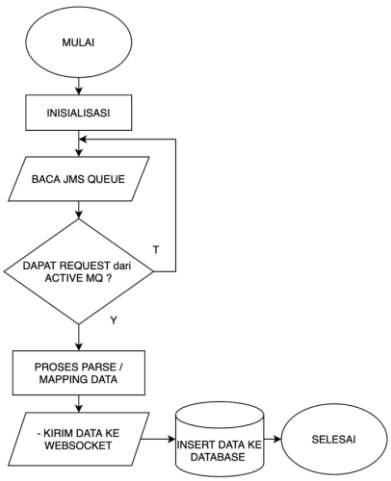
Pada nomor 4, layanan konsumen antrean menerima dan mengambil data yang tersedia pada antrean *activeMQ* dengan metode *round robin*. Setelah itu data dimasukan ke dalam *database* pada nomor 5. Jika diperhatikan pada langkah nomor 5 data juga langsung diteruskan ke *websocket server* untuk berkomunikasi dengan *client side* pada *website Laravel*. Sehingga data pada *website* berubah secara *realtime*. Pada nomor 6, ada permintaan data dari *website laravel* ke *backend API* secara berkala, sehingga *backend API* akan memberikan data terbaru sensor yang ada di dalam *database*.

Pada Gambar 8 *backend* akan menunggu hingga ada data yang dikirimkan dari alat mikrokontroler. Jika terdapat permintaan selanjutnya akan memproses data agar dapat dibaca oleh sistem *backend API*. Setelah itu data tersebut akan melakukan proses validasi seperti pembacaan setiap pintu, *autorisasi*, *user* dan akses. Validasi benar ataupun salah akan di *set* status nya diberikan kembali ke mikrokontroler sebagai respon balikan data. Setelah itu data dilanjutkan ke dalam antrean *activeMQ*.

DOI: 10.34148/teknika.v11i1.418



Gambar 8. Diagram Alur *Backend Service*



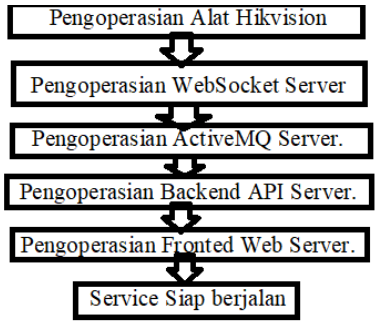
Gambar 9. Diagram Alur *Consumer Queue*

Pada Gambar 9 pemrosesan konsumer antrean data, sistem akan menunggu dan membaca apabila terdapat *JMS* (*java message service*) yang dikirimkan, setelah itu data tersebut di *parse* (ubah bentuk datanya agar dapat dibaca). Selanjutnya data dikirimkan ke *websocket server* untuk ditampilkan ke dalam *website*. Setelah itu data di masukan ke dalam *database*.

III. HASIL DAN PEMBAHASAN

A. Persiapan Pengoperasian Sistem

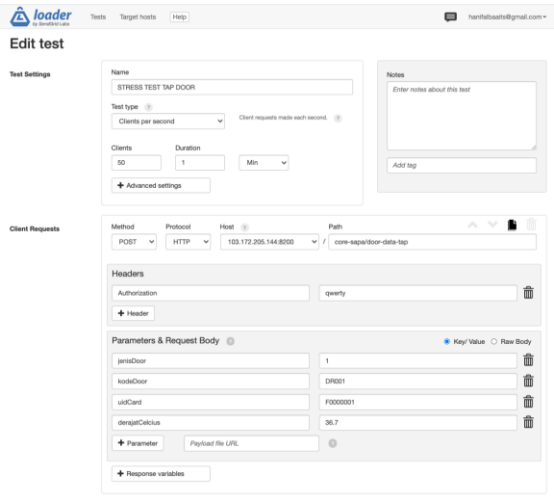
Tahapan ini menjelaskan cara pengoperasian perangkat *hikvision* dan pengoperasian pada halaman *website*. Tahapan pengoperasian ini digunakan untuk mengetahui alur dari pengoperasian sistem agar dapat berjalan dengan baik, terlihat pada blok diagram Gambar 10.



Gambar 10. Blok Diagram Langkah Persiapan Pengoperasian Sistem

B. Pengujian Antrean dan *Stress Test API*

Pada pengujian antrean ini dilakukan pengujian untuk melakukan permintaan data ke *server* sebanyak mungkin/membanjiri *service* dengan permintaan data dengan jumlah banyak dalam durasi periode tertentu dan bertahap. Pengujian ini untuk membandingkan permintaan data yang banyak ketika melakukan dengan atrian data dan tidak melakukan antrean data. *Stress Test* disini memakan kerja *processor* yang maksimal. Salah satu faktor yang dapat mempengaruhi yaitu spesifikasi *server* yang digunakan. Pengujian disini menggunakan *server VPS* (*Virtual Private Server*) dengan spesifikasi, *CPU* 2 *Core*, *RAM* 4 Gb, *Storage* 30 Gb *SSD*, *Bandwith Unlimited*. Cara kerja *stress* disini yaitu melakukan permintaan data setiap detik selama periode tertentu. Jika ada 2 mikrokontroler yang mengirimkan data selama 30 detik. Berarti ada 2 *request API* setiap detik nya sehingga total *request API* yang terjadi adalah 60 hit *API*.



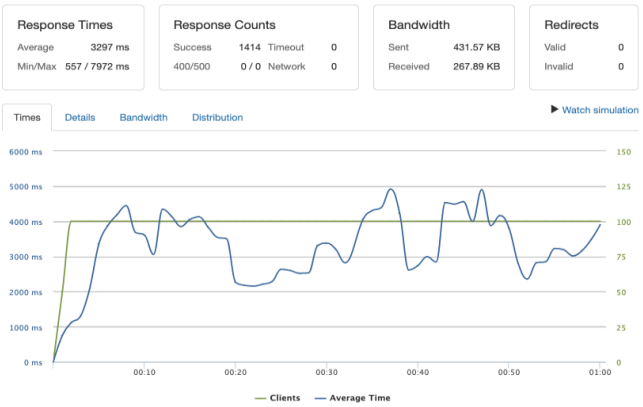
Gambar 11. *Stress Test* Menggunakan *Loader.io*

Pada Gambar 11 adalah tampilan untuk pengujian *stress test* menggunakan *Loader.io*. Pada pengujian *stress test* ini melakukan pengujian sebanyak 18 percobaan menggunakan dengan antrean dan tidak menggunakan antrean. Susunan pengujian nya terlihat pada tabel 1.

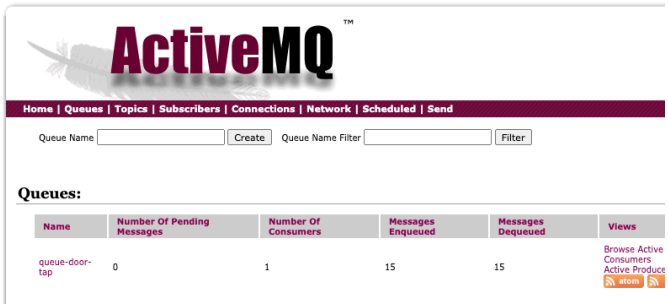
Tabel 1. Tabel Sususan Pengujian

Pengujian Ke-	Jumlah *Mikrokontroler*	Durasi Selama 15 Detik		Durasi Selama 30 Detik		Durasi Selama 1 Menit	
		Jumlah *Request* / Detik	Total *Request*	Jumlah *Request* / Detik	Total *Request*	Jumlah *Request* / Detik	Total *Request*
1	1	1	15	1	30	1	60
2	5	5	75	5	150	5	300
3	10	10	150	10	300	10	600
4	20	20	300	20	600	20	1200
5	30	30	450	30	900	30	1800
6	50	50	750	50	1500	50	3000

Gambar 12. Grafik 1 Mikrokontroler Periode 15 Detik.

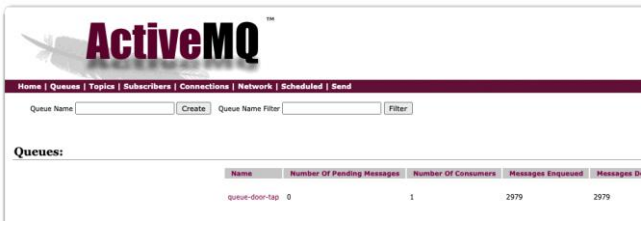


Gambar 13. Grafik 50 Mikrokontroler Periode 1 Menit



Gambar 13. Antrean *ActiveMQ 15 Request*

Pada Gambar 12 dan 13 percobaan pertama untuk melakukan *stress test* 1 mikrokontroler selama 15 detik dengan total permintaan data 15 *request*. *ActiveMQ* telah berhasil melakukan antrean untuk data yang masuk dan memproses kembali (*message enqueue* dan *message dequeue*) sebanyak 15 antrean.



Gambar 15. Antrean *ActiveMQ 3000 Request*

Pada Gambar 14 dan 15 percobaan terakhir antrean data menggunakan 50 mikrokontroler selama 1 menit dengan total permintaan data 3000 *request*. *ActiveMQ* telah berhasil melakukan antrean untuk data yang masuk dan memproses kembali (*message enqueue* dan *message dequeue*) sebanyak 2979 antrean.

DOI: 10.34148/teknika.v11i1.418

Tabel 2. Data Pengamatan *Stress Test* Dengan Antrean

No	*Request API*			*Time Respon (ms)*			*Status*		*Queue*		*Percent Success*
	Total *Client*	*Duration (seconds)*	Total *Request*	*Average*	*Min*	*Max*	*Success Send*	*Failed Send*	*Messagge enqueue*	*Mesasage dequeue*	
1	1	15	15	356	287	575	15	0	15	15	100%
2	1	30	30	454	327	1063	30	0	30	30	100%
3	1	60	60	358	287	1344	60	0	60	60	100%
4	5	15	75	420	286	972	75	0	75	75	100%
5	5	30	150	643	296	3139	150	0	150	150	100%
6	5	60	300	486	280	1790	300	0	300	300	100%
7	10	15	150	575	296	1720	150	0	150	150	100%
8	10	30	300	502	278	1410	300	0	300	300	100%
9	10	60	600	522	283	2575	600	0	600	600	100%
10	20	15	300	746	284	2352	300	0	300	300	100%
11	20	30	600	692	276	2546	600	0	600	600	100%
12	20	60	1200	861	275	2780	1200	0	1200	1200	100%
13	30	15	450	876	287	2152	443	7	443	443	98,40%
14	30	30	900	1392	511	3812	813	87	813	813	90,33%
15	30	60	1800	1382	481	3888	1728	72	1728	1728	96%
16	50	15	750	2093	549	4664	528	222	528	528	70,40%
17	50	30	1500	2310	518	4704	988	512	988	988	65,86%
18	50	60	3000	3297	557	7927	2979	1518	2979	2979	99,30%
											96%

Tabel 3. Data Pengamatan *Stress Test* Tanpa Antrean

No	*Request API*			*Time Respon (ms)*			*Status*		*Percent Success*
	Total Client	*Duration (seconds)*	Total *Request*	*Average*	*Min*	*Max*	*Success Send*	*Failed Send*	
1	1	15	15	442	377	575	15	0	100%
2	1	30	30	507	454	1063	30	0	100%
3	1	60	60	738	647	1344	60	0	100%
4	5	15	75	718	425	986	72	3	96%
5	5	30	150	1178	649	3151	147	3	98%
6	5	60	300	652	495	1816	295	5	98%
7	10	15	150	1172	582	1741	146	4	97%
8	10	30	300	738	517	1452	292	8	97%
9	10	60	600	1756	535	2617	592	8	99%
10	20	15	300	1445	764	2410	289	11	96%
11	20	30	600	1877	717	2625	585	15	98%
12	20	60	1200	2877	932	3002	1158	42	97%
13	30	15	450	2123	1120	2915	306	144	68%
14	30	30	900	3949	1947	5545	573	327	64%
15	30	60	1800	3671	2041	5944	1412	388	78%
16	50	15	750	4669	2674	6471	409	341	55%
17	50	30	1500	7251	4461	11419	512	1267	34%
18	50	60	3000	11238	7280	21508	675	2325	23%
									83%

Terlihat pada Tabel 2 dan 3 adalah data pengamatan dari hasil pengujian *stress test* dengan antrean dan tanpa antrean data. Setiap percobaan pengujian dihitung tingkat persentase keberhasilan permintaan yang datang hingga dapat diproses dan dimasukan ke dalam *database*, seperti pada rumus (1).

$$Persentase\ berhasil = \frac{Jumlah\ request\ yang\ berhasil\ di\ proses}{Jumlah\ keseluruhan\ request\ yang\ dikirim}\ x\ 100\% \quad (1)$$

Setelah menghitung total persentase keberhasilan, penelitian ini menghitung rata rata ke seluruhan percobaan tingkat keberhasilan dengan rumus (2).

$$Rata\ rata\ persentase\ sukses = \frac{Total\ Persentase\ Success}{Banyaknya\ Percobaan}\ x\ 100\% \quad (2)$$

- Untuk pengujian *stress test* dengan melakukan antrean data dihitung tingkat keberhasilan sebesar 96 %
- Untuk pengujian *stress test* dengan tidak melakukan antrean data dihitung tingkat keberhasilan sebesar 83 %.

Ada beberapa faktor yang menyebabkan perbedaan tingkat persentase keberhasilan dalam pengujian ini, yaitu.

- Beban Pemrosesan *server* saat melakukan *stress test* dibanjiri oleh banyak pengiriman data dari mikrokontroler. Karena itu *server* melakukan *multitasking* maka beban proses CPU dan RAM meningkat.
- *Bandwith* data yang diberikan dari *server VPS* yaitu *unlimited*. Tetapi juga memperhitungkan kecepatan data mikrokontroler saat terhubung ke internet. Penelitian ini menggunakan *Provider* Indihome Telkom dengan kecepatan sebesar 31,46 mbps untuk *download* dan 10,56 mbps untuk *upload*.
- Implementasi model antrean dari alat ini adalah akses pintu dimana *request* atau permintaan data yang masuk atau datang harus di periksa terlebih dahulu seperti kode pintu, *autorisasi*, kode *uid* dan *user*. Sehingga memerlukan proses yang lebih dibandingkan dengan pengiriman data sensor yang hanya memasukan ke dalam *database* tanpa ada validasi terlebih dahulu.

C. Pengujian *Realtime Polling* dan *Realtime Handshake*

Pada pengujian ini akan melakukan pengujian dan percobaan untuk melakukan perbandingan seberapa cepat data yang ditampilkan setelah alat atau mikrokontroler mengirim data antara metode *Polling* dan metode *Handshake*. Pengujian ini untuk menghitung waktu jeda antara data masuk sampai data ditampilkan.

1. Metode *Realtime Polling*.

Pada Tabel 4 adalah data pengamatan menggunakan metode *Polling*. Dimana penelitian ini melakukan percobaan selama 20 *request API*. Pengunaan penelitian ini menggunakan cetak catatan atau *logger* pada sistem. *Epochtime* adalah waktu hari ini dalam bilangan unik *millisecond*. Sehingga *Epochtime Tap* adalah waktu *milisecond* unik pada saat tap atau pada saat data datang. *Epochtime Show* adalah waktu *milisecond* unik pada saat data ditampilkan ke *website* secara *Realtime*. *Epochtime Diff* adalah waktu selisih antara *Epochtime Tap* dengan *Epochtime Show.* Hasil selisih tersebut dengan satuan *milisecond* (ms).

Tabel 4. Data Pengamatan Metode *Polling*

No	*Epochtime Tap*	*Epochtime Show*	*Epochtime Diff*
1	1634044724913	1634044726354	1441
2	1634044720076	1634044720281	205
3	1634044717055	1634044717276	221
4	1634044713837	1634044714009	172
5	1634044708347	1634044709457	1110
6	1634044704852	1634044706358	1506
7	1634044700384	1634044701502	1118
8	1634044695359	1634044697043	1684
9	1634044692915	1634044693719	804
10	1634044689644	1634044690568	924
11	1634044687445	1634044687634	189
12	1634045261222	1634045261351	129
13	1634045259433	1634045259767	334
14	1634045257954	1634045258277	323
15	1634045256516	1634045256811	295
16	1634045255016	1634045255333	317
17	1634045253517	1634045253765	248
18	1634045251560	1634045252155	595
19	1634045248808	1634045250367	1559
20	1634045246992	1634045247217	225
			669,95 (ms)

2. Metode *Realtime Handshake*

Tabel 5. Data Pengamatan Metode *Handshake*

No	*Epochtime Tap*	*Epochtime Show*	*Epochtime Diff*
1	1634043534282	1634043534295	13
2	1634043590312	1634043590329	17
3	1634043591830	1634043591867	37
4	1634043593229	1634043593241	12
5	1634043594894	1634043594895	1
6	1634043596413	1634043596418	5
7	1634043598161	1634043598165	4
8	1634043599565	1634043599672	107
9	1634043601200	1634043601267	67
10	1634043602600	1634043603005	405
11	1634044079901	1634044079911	10
12	1634044275287	1634044275330	43
13	1634044308840	1634044308908	68
14	1634044309815	1634044309841	26
15	1634044310881	1634044310900	19
16	1634044312365	1634044312419	54
17	1634044313630	1634044313659	29
18	1634044314976	1634044315027	51
19	1634044316587	1634044316745	158
20	1634044318156	1634044318302	146
			63,6 (ms)

Terlihat pada Tabel 5 adalah data pengamatan dari hasil pengujian metode *realtime handshake*. Setiap Percobaan melakukan selisih hitung dari unik time *milisecond* antara waktu penerimaan data (*epochtime tap*) dengan waktu penampilan data di *web* (*epochtime show*). Tingkat keberhasilan metode ini dihitung dari seberapa kecil jeda waktu yang didapat. Penelitian ini menghitung rata rata ke seluruhan percobaan tingkat keberhasilan dengan rumus (3).

$$Rata\ rata\ delay\ tunda\ data = \frac{Total\ Selisih\ Delay\ data}{Banyaknya\ Percobaan} \quad (3)$$

DOI: 10.34148/teknika.v11i1.418

- Untuk pengujian website *realtime* dengan metode *polling* dihitung tingkat kecepatan data / waktu tunda sebesar 669,95 ms.
- Untuk pengujian website *realtime* dengan metode *handshake* dihitung tingkat kecepatan data / waktu tunda sebesar 63,6 ms.

IV. KESIMPULAN DAN REKOMENDASI

Perancangan dan Simulasi proses antrean data pada *multisensor* untuk sistem *telemonitoring* dan multikontrol berbasis *internet of things* telah berhasil untuk melakukan pengujian antrean data dan metode *realtime* terbaik yang digunakan. Berdasarkan penelitian dan pengujian antrean data menggunakan *ActiveMQ* yang dilakukan dengan melakukan perbandingan pengunaan antrean data dan tidak meggunakan antrean data pada saat mikrokontroler melakukan *request* data ke *backend API* maka di dapat hasil, Metode Pengunaan antrean data lebih efektif dengan tingkat keberhasilan data sukses diterima dan diproses sebesar 96 %, dibandingkan dengan tidak menggunakan antrean tingkat keberhasilan data sukses diterima dan diproses sebesar 83 %. Berdasarkan penelitian dan pengujian untuk mencari metode terbaik dalam *realtime web* antara menggunakan pengunaaan metode *polling* dan metode *handshake* di dapat hasil, Metode *handshake* jauh lebih cepat dan tepat dengan jeda waktu antara permintaan *request* data masuk hingga di tampilkan sebesar 63,6 ms, dibandingkan dengan metode *polling* dengan jeda waktu antara permintaan *request* data masuk hingga di tampilkan sebesar 669,95 ms.

Rekomendasi tahap selanjutnya sistem dan perangkat dapat ditambahkan lebih mikrokontroler dan sensor untuk mencoba secara langsung dalam tahap implementasi. Sistem dan perangkat dapat integrasi dengan sistem yang lebih besar menggunakan *API*, seperti sistem statistik dan data analitik. Sistem dan perangkat ini dapat menampung ribuan permintaan data sensor, sehingga masih dapat digunakan dan di implementasikan dengan banyak sensor sensor *Internet of Things* lain nya.

REFERENSI

[1] V. Florentin, “Presiden Jokowi Luncurkan roadmap Revolusi Industri 4.0,” *Tempo*, 04-Apr-2018. [Online]. Available: https://bisnis.tempo.co/read/1076107/presiden-jokowi-luncurkan-roadmap-revolusi-industri-4-0. [Accessed: 04-Apr-2020].

[2] B. Di Martino, A. Esposito, S. A. Maisto, and S. Nacchia, “A semantic IOT framework to support restful devices' API interoperability,” *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, 2017.

[3] K. Guntupally, R. Devarakonda, and K. Kehoe, “Spring boot based REST API to improve data quality report generation for Big Scientific Data: ARM data center example,” *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[4] JiabaoYin, S. He, F. Zhao, and S. Li, “Design and Implementation of Intelligent Load-Balancing Heterogeneous Data Source Middleware Based on ActiveMQ and XML,” *2015 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration*, 2015.

[5] Q. Liu and X. Sun, “Research of web real-time communication based on web socket,” *International Journal of Communications, Network and System Sciences*, vol. 05, no. 12, pp. 797–801, 2012.

[6] H. Djamal, “Radio Frequency Identification (RFID) Dan Aplikasinya,” *TESLA: Jurnal Teknik Elektro* , vol. 14, no. 1, 2014.

[7] R. Y. He, “Design and implementation of web based on Laravel framework,” *Advances in Computer Science Research*, 2015.