ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

RANCANG BANGUN SISTEM KOLABORASI PADA LINGKUNGAN *CROSS PLATFORM* SECARA *REAL TIME* DENGAN MEKANISME *PUBLISH/SUBSCRIBE* BERBASIS *LIGHTWEIGHT MESSAGING PROTOCOL* (STUDI KASUS : KOLABORASI SENSOR TERDISTRIBUSI)

MUHAMAD LUTHFIE LA ROEHA
NRP 5113 100 113

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

RANCANG BANGUN SISTEM KOLABORASI PADA LINGKUNGAN *CROSS PLATFORM* SECARA *REAL TIME* DENGAN MEKANISME *PUBLISH/SUBSCRIBE* BERBASIS *LIGHTWEIGHT MESSAGING PROTOCOL* (STUDI KASUS : KOLABORASI SENSOR TERDISTRIBUSI)

MUHAMAD LUTHFIE LA ROEHA
NRP 5113 100 113

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

BUILDING REAL TIME COLLABORATION SYSTEM ON CROSS PLATFORM ENVIRONMENT USING LIGHTWEIGHT MESSAGING PROTOCOL BASED PUBLISH/SUBSCRIBE MECHANISM (CASE STUDY: COLLABORATION OF DISTRIBUTED SENSOR)

MUHAMAD LUTHFIE LA ROEHA
NRP 5113 100 113

Supervisor I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN SISTEM KOLABORASI PADA LINGKUNGAN *CROSS PLATFORM* SECARA *REAL TIME* DENGAN MEKANISME *PUBLISH/SUBSCRIBE* BERBASIS *LIGHTWEIGHT MESSAGING PROTOCOL* (STUDI KASUS : KOLABORASI SENSOR TERDISTRIBUSI)

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
MUHAMAD LUTHFIE LA ROEHA
NRP: 5113 100 113

Disetujui oleh Dosen Pembimbing Tugas Akhir

Waskitho Wibisono, S.Kom., M.Eng., Ph.D. .....
NIP: 197410222000031001 (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D .....
NIP: 051100116 (Pembimbing 2)

SURABAYA
Juni 2017

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN SISTEM KOLABORASI PADA LINGKUNGAN *CROSS PLATFORM* SECARA *REAL TIME* DENGAN MEKANISME *PUBLISH/SUBSCRIBE* BERBASIS *LIGHTWEIGHT MESSAGING PROTOCOL* (STUDI KASUS : KOLABORASI SENSOR TERDISTRIBUSI)

Nama	:	**MUHAMAD LUTHFIE LA ROEHA**
NRP	:	**5113 100 113**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Dalam masa transisi menuju masa Internet of Things, pertumbuhan penggunaan jumlah sensor yang dipakai di seluruh dunia berkembang dengan sangat cepat. Salah satu fungsinya yaitu untuk mengenali keadaan di sekitar secara real-time. Sehubungan dengan itu, kumpulan data sensor tersebut juga dapat dimanfaatkan untuk melakukan suatu aksi secara otomatis. Untuk melakukan hal tersebut diperlukan suatu sistem yang dapat melakukan otomatisasi aksi berdasarkan data kolaborasi sensor. Sistem tersebut harus mendukung komunikasi lintas platform secara real-time agar aplikasi monitoring data sensor selalu mendapatkan data paling mutakhir. Sistem juga harus menggunakan sumber daya komputer dan bandwidth seefisien mungkin agar dapat menghemat biaya operasional ketika digunakan pada skala besar.

Pada tugas akhir ini dijelaskan mulai dari tahapan perancangan hingga tahapan implementasi serta evaluasi hasil uji coba untuk memenuhi kebutuhan sistem kolaborasi tersebut.

Terdapat aplikasi Android yang berfungsi untuk monitoring data sensor dan aplikasi Web untuk mengelola kolaborasi aturan sensor. Aturan data sensor tersebut menggunakan operator relasional dan operator logika. Aturan tersebut nantinya digunakan untuk melakukan otomatisasi aksi oleh sistem. Ketika sensor mengirimkan data, sistem menerima data tersebut lalu mendistribusikannya pada aplikasi Android dan Web, serta memproses data tersebut untuk mengevaluasi kolaborasi aturan. Ketika aturan terpenuhi, aksi secara otomatis dijalankan.

Berdasarkan hasil uji coba yang dilakukan, sistem kolaborasi dapat menyediakan layanan untuk monitoring data sensor secara real-time pada platform Android dan Web, mengelola kolaborasi aturan sensor, dan melakukan otomatisasi aksi berdasarkan kolaborasi aturan sensor yang telah dikonfigurasi. Sistem juga mampu mengevaluasi kolaborasi data sensor dan melakukan otomatisasi aksi hingga ± 1000 *data tiap detiknya dengan menggunakan sumber daya RAM hanya sebesar 23.12 MB dan bandwidth sebesar 76 KB.*

***Kata-kunci**: publish/subscribe, MQTT, sensor, otomatisasi aksi, komunikasi lintas platform, real time*

BUILDING REAL TIME COLLABORATION SYSTEM ON CROSS PLATFORM ENVIRONMENT USING LIGHTWEIGHT MESSAGING PROTOCOL BASED PUBLISH/SUBSCRIBE MECHANISM (CASE STUDY: COLLABORATION OF DISTRIBUTED SENSOR)

Name	:	**MUHAMAD LUTHFIE LA ROEHA**
NRP	:	**5113 100 113**
Major	:	**Informatics FTIf**
Supervisor I	:	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D**
Supervisor II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstract

As we are moving towards the Internet of Things (IoT) era, the number of sensors deployed around the world is growing at a rapid pace. These sensors offer real-time sensing of the environment. In advances, these sensor data can be used to enable triggering autonomous reaction to changes in the physical world. To do so requires a system which can enable triggering autonomous reaction based on collaboration of sensor data. The system should support real-time cross platform communication system to enable monitoring applications to get the latest data. The system also should use less resource and bandwidth to increase cost efficiency while being used in a large scale deployment.

This work explain a step by step to design and implement the collaboration system. We build an Android app to monitor sensor data and a Web app to manage rule of sensor collaboration data. These rule can be built using relational and logical operator. It will be used to trigger automated action. When the sensor send their data to the system, the system will distribute it to Android app

and Web app, as well as processing the data to evaluate the rules. When the rules are met by the sensor data, system will invoke an action automatically.

Finally, based on our evaluation, the collaboration system can provide real-time sensor data monitoring, managing rule of sensor collaboration data, and trigger an autonomous action based on configured action. This system is also capable of evaluating collaboration of sensor data and trigger an autonomous action up to ± 1000 *data per second using 23.12 MB of RAM and 76 KB of network data.*

Keywords*: publish/subscribe, MQTT, sensor, automation, internet of things, real time cross platform communication*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Sistem Kolaborasi Pada Lingkungan *Cross Platform* Secara *Real Time* Dengan Mekanisme *Publish/Subscribe* Berbasis *Lightweight Messaging Protocol* (Studi Kasus : Kolaborasi Sensor Terdistribusi)**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Orang tua atas kasih sayang, semangat, dan pelajaran hidup yang diberikan ketika masa hidupnya hingga penulis bisa tetap semangat menjalani hidup ini.
3. Keluarga besar yang senantiasa memberikan do'a, dukungan dari segi moral maupun materi, dan motivasi hingga saat ini.
4. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku dosen pembimbing I yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
5. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku dosen

pembimbing II yang telah meluangkan waktu untuk membimbing, memotivasi, dan membantu penulis ketika penulis kesulitan dalam mengerjakan Tugas Akhir.

6. Bapak Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS saat ini, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.
7. Teman-teman Kontrakan Berprestasi, Daniel Bintar, Daniel Fablius, Dewangga, Donny, Irsyad, Juan, dan Wicak yang telah mengingatkan untuk shalat, membantu penulis dalam menyelesaikan permasalahan dalam mengerjakan tugas akhir, yang selalu menghibur, memotivasi, dan mendukung penulis untuk menyelesaikan tugas akhir ini.
8. Teman-teman Laboratorium NCC dan Administrator NCC yang bersedia direpotkan dan menemani penulis dalam masa pengerjaan tugas akhir ini.
9. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

Muhamad Luthfie La Roeha

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Internet adalah salah satu penemuan yang sangat penting dan berguna dalam sejarah manusia. Pada tahun 2010, jumlah perangkat yang terhubung ke internet sebesar 12.5 miliar, sedangkan populasi penduduk di dunia sebanyak 6.8 miliar. Perbedaan jumlah perangkat dan penduduk hampir mencapai dua kali lipat dan diprediksikan akan terus meningkat drastis[1]. Dalam masa transisi menuju masa *Internet of Things*(IoT), pertumbuhan penggunaan jumlah sensor yang dipakai di seluruh dunia berkembang dengan sangat cepat[2], seperti mengenali keadaan di sekitar menggunakan sensor secara *real time*. Sehubungan dengan itu, selain dapat digunakan untuk mendapatkan informasi dari sensor, kumpulan data sensor tersebut juga dapat dimanfaatkan untuk melakukan suatu aksi secara otomatis[3]. Untuk melakukan hal tersebut dibutuhkan suatu sistem untuk melakukan otomatisasi aksi berdasarkan kolaborasi data sensor.

Sistem kolaborasi adalah sistem yang berfungsi untuk melakukan *monitoring* data sensor pada *platform* Android dan Web. Sistem ini juga dapat melakukan otomatisasi aksi ketika data sensor yang diterima memenuhi aturan yang dibuat oleh pengguna. Sistem tersebut harus dapat berkomunikasi lintas *platform* untuk dapat mendistribusikan data rekaman sensor serta melakukan aksi secara otomatis secara *real time*. Dikarenakan jumlah penggunaan sensor yang semakin banyak, sistem harus menggunakan sumber daya komputer dan *bandwidth* seefisien

mungkin agar dapat menghemat biaya operasional ketika digunakan pada skala besar.

Untuk mengimplementasikan hal tersebut, dibutuhkan suatu metode komunikasi yang mendukung pertukaran data lintas *platform* secara *real time*. Selain itu, dibutuhkan protokol yang tersedia pada banyak *platform*, dapat menyajikan data secara *real time*, dan menggunakan *bandwidth* yang rendah sehingga dapat menghemat biaya operasional ketika digunakan pada skala besar.

Salah satu metode komunikasi yang mendukung pertukaran data lintas *platform* yaitu metode *client/server* menggunakan *web services*. Keuntungan menggunakan *web services* yaitu dapat digunakan pada banyak bahasa pemrograman dan *platform* yang berbeda[4]. *Web services* biasanya berinteraksi menggunakan protokol HTTP dengan format data XML[5]. Tetapi, untuk dapat menyajikan data secara *real time*, setiap kali ada perubahan data, *client* perlu melakukan *request* ulang ke *server* untuk mendapatkan data baru sehingga berdampak boros *resource* karena tiap *request* membutuhkan koneksi *socket* baru dan pesan HTTP yang baru[6]. Maka dari itu, diperlukan metode lain yang dapat menangani komunikasi lintas *platform* secara *real time*, yaitu metode *publish/subscribe*[7]. *Publish/subscribe* merupakan metode yang memisahkan antara pengirim pesan (*publisher*) dengan penerima pesan (*subscriber*). *Publisher* menentukan topik yang nantinya akan menjadi tempat diterbitkannya data. *Subscriber* dapat mendaftar untuk berlangganan data berdasarkan topik yang diminati. Ketika *publisher* menerbitkan data pada topik tertentu, *server* meneruskan data tersebut kepada *subscriber*. *Subscriber* dapat mengakses data yang diterbitkan kapanpun dan dimanapun secara asinkron[8].

Untuk menggunakan metode komunikasi *publish/subscribe*, dibutuhkan protokol yang mendukung metode komunikasi tersebut. Salah satu protokol yang mendukung metode *publish/subscribe* adalah protokol MQTT[3]. *Message Queue*

Telemetry Transport (MQTT) adalah protokol pub/sub murni, yang sangat sederhana dan ringan yang dapat digunakan untuk perangkat yang memiliki keterbatasan, seperti menggunakan *bandwidth* rendah, berjalan pada jaringan yang tidak *reliable* dan berlatensi tinggi[9]. Selain itu, protokol MQTT juga digunakan sebagai infrastruktur *back-end* pada aplikasi *smartphone* untuk mengirimkan notifikasi berskala besar. Contohnya aplikasi *Facebook Messenger* menggunakan protokol MQTT untuk berkomunikasi dengan *server*. [10].

Oleh karena itu, sistem kolaborasi ini dibangun dengan menerapkan metode komunikasi *publish/subscribe* pada lingkungan *cross platform* menggunakan protokol MQTT. Harapannya sistem dapat melakukan fungsi *monitoring* pada lingkungan *cross platform* dan melakukan otomatisasi aksi berdasarkan kolaborasi data sensor secara *real time* menggunakan sumber daya secara efisien dan dapat digunakan pada skala besar.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana membangun sistem kolaborasi untuk *monitoring* data sensor secara *real time* pada lingkungan *cross platform* menggunakan protokol MQTT ?
2. Bagaimana mengevaluasi kolaborasi data sensor yang dikirimkan secara *real time* untuk melakukan otomatisasi aksi ?
3. Bagaimana performa sistem dan kebutuhan sumber daya sistem dalam menangani sejumlah data rekaman sensor, mengevaluasi kolaborasi data sensor dan melakukan otomatisasi aksi ?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Jenis metode komunikasi *publish/subscribe* yang dipakai adalah *topic based publish/subscribe*.
2. Protokol yang digunakan untuk berkomunikasi adalah protokol MQTT.
3. *Device* yang digunakan diantaranya *smartphone* Android, Raspberry Pi, dan komputer.
4. Sensor yang tersedia pada sistem berjumlah 4 sensor yang terdiri dari 1 sensor *motion* menggunakan Raspberry Pi dan 3 sensor simulasi menggunakan *script* python
5. Aksi yang tersedia pada sistem berjumlah 2 aksi, yaitu *push notification* dan mengirim *email*.
6. Penulisan program menggunakan bahasa Java pada *platform* Android; Python pada *platform* Raspberry Pi; Javascript, HTML dan CSS pada *platform* Web.
7. Data sensor bertipe data *integer*.

1.4 Tujuan

Tujuan dari pengerjaan tugas akhir ini adalah membangun sistem kolaborasi untuk *monitoring* data sensor dan otomatisasi aksi pada lingkungan *cross platform* yang berkomunikasi secara *real time* dengan mekanisme *publish/subscribe* menggunakan protokol MQTT

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini antara lain :

1. Menghasilkan sistem yang dapat memonitor data rekaman sensor secara *real time* pada lingkungan *cross platform* dan melakukan aksi secara otomatis berdasarkan kolaborasi data sensor.
2. Sebagai referensi solusi untuk permasalahan komunikasi *cross platform* secara *real time*.

1.6 Metodologi

1.6.1 Studi literatur

Studi literatur yang dilakukan bertujuan untuk mencari informasi mengenai implementasi mekanisme *publish/subscribe* untuk distribusi pesan lintas *platform* menggunakan protokol MQTT pada *platform* Android, Raspberry Pi, dan Web. Selain itu, juga dilakukan studi literatur mengenai implementasi teknologi yang dibutuhkan untuk mengimplementasikan sistem, yaitu mengenai Raspberry Pi, bahasa pemrograman Go, bahasa pemrograman Python, *postgresql*, *graphql*, Android, *react*, *relay*, *ant design* dan *redux*.

1.6.2 Desain dan Perancangan Sistem

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat kasus penggunaan yang ada pada sistem, arsitektur sistem, serta perencanaan implementasi perangkat lunak pada sistem.

1.6.3 Implementasi Sistem

Implementasi merupakan tahap membangun implementasi rancangan sistem yang telah dibuat. Pada tahapan ini merealisasikan apa yang telah didesain dan dirancang pada tahapan sebelumnya, sehingga menjadi sebuah sistem yang sesuai dengan apa yang telah direncanakan.

1.6.4 Uji Coba dan Evaluasi

Pada tahapan ini dilakukan uji coba terhadap sistem yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Selain itu, tahap ini juga akan melakukan uji performa sistem dan melakukan perbandingan dengan protokol lain untuk mengetahui kecepatan, efisiensi penggunaan sumber daya serta evaluasi berdasarkan hasil uji performa tersebut.

1.7 Sistematika Laporan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna bagi pembaca yang berminat melakukan pengambangan lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri atas beberapa bagian seperti berikut:

Bab I Pendahuluan

Bab yang berisi latar belakang, tujuan, manfaat, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika laporan.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang yang digunakan dalam pembuatan tugas akhir ini.

Bab II Desain dan Perancangan

Bab ini berisi tentang analisis dan perancangan sistem yang dibuat, termasuk di dalamnya mengenai analisis kasus penggunaan, desain arsitektur sistem, dan perancangan implementasi sistem.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa pemasangan alat dan kode program yang digunakan untuk mengimplementasikan sistem.

Bab V Uji Coba dan Evaluasi

Bab ini membahas tahap-tahap uji coba serta melakukan evaluasi terhadap sistem yang dibuat.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang memberikan kesimpulan dari hasil percobaan dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran bagi pembaca yang berminat untuk melakukan pengembangan lebih lanjut.

(Halaman ini sengaja dikosongkan)

BAB II

LANDASAN TEORI

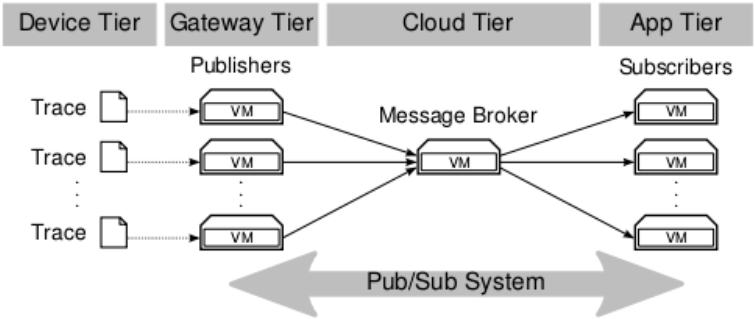
2.1 Publish/Subscribe

Mekanisme *publish/subscribe* adalah suatu *middleware* yang berorientasi pesan yang menyediakan komunikasi yang terdistribusi, asinkron dan tidak terikat antara pengirim pesan dan penerima pesan[3]. Pengirim pesan disebut sebagai *publisher* dan penerima pesan disebut sebagai *subscriber*. *Publisher* menentukan topik yang nantinya akan menjadi tempat diterbitkannya data. *Subscriber* dapat mendaftar untuk berlangganan data berdasarkan topik yang diminati. Ketika *publisher* menerbitkan data pada topik tertentu, *server* meneruskan data tersebut kepada *subscriber*. *Subscriber* dapat mengakses data yang diterbitkan kapanpun dan dimanapun secara asinkron[8].

Terdapat dua jenis *publish/subscribe*, *topic based publish/subscribe* dan *content based publish/subscribe*. *Topic based publish/subscribe* merupakan pola pengiriman pesan *publish/subscribe* dimana *publisher* menyampaikan pesan ke *subscriber* berdasarkan topik yang dipilihnya. *Content based publish/subscribe* merupakan pola pengiriman pesan *publish/subscribe* dimana *publisher* menyampaikan pesan ke *subscriber* berdasarkan isi dari pesan yang ada.

Suatu *middleware pub/sub* menawarkan tiga jenis fitur [11], diantaranya :

1. *Publisher* dan *subscriber* terpisah. Maksudnya, mereka tidak perlu terkoneksi pada waktu yang sama.
2. Pesan tidak ditujukan untuk *subscriber* tertentu, melainkan ditujukan pada target yang simbolis (contohnya *channel*/topik).
3. Pengiriman dan penerimaan pesan dilakukan secara asinkron.



Gambar 2.1: Visualisasi mekanisme *publish/subscribe*[3]

Agar *publisher* dapat mengirim pesan ke *subscriber*, terdapat *message broker* yang berfungsi sebagai penerus pesan yang dikirim oleh *publisher* menuju *subscriber* yang tertarik pada topik yang dikirim.

2.2 Message Queue Telemetry Transport

Message Queue Telemetry Transport (MQTT) [9] adalah protokol pub/sub murni untuk perangkat yang memiliki keterbatasan, menggunakan *bandwidth* rendah, berlatensi tinggi, dan jaringan yang tidak *reliable* yang dikembangkan oleh IBM dan dijadikan standar oleh OASIS. Protokol ini sangat ringan, terbuka (*open source*), sederhana dan didesain agar mudah untuk digunakan. Karakteristik tersebut yang membuat protokol ini ideal untuk digunakan pada banyak situasi dan lingkungan yang terbatas, seperti komunikasi antar mesin (*Machine to Machine communication*) dan *Internet of Things* (IoT) dimana dibutuhkan *bandwidth* yang rendah dan kode yang sesedikit mungkin. Protokol ini berjalan diatas protokol TCP/IP, atau jaringan lain yang menyediakan koneksi yang *lossless* dan *bi-directional*. MQTT memiliki fitur[9], diantaranya:

1. Menggunakan *publish/subscribe message pattern* yang menyediakan distribusi pesan *one-to-many* dan ketidakterikatan antar aplikasi.
2. Tiga tipe *Quality of Services* (QoS):
   (a) ***At most once***, dimana pesan dikirim dengan upaya terbaik dari jaringan TCP/IP. Kehilangan pesan atau duplikasi dapat terjadi.
   (b) ***At least once***, dimana pesan dapat dipastikan diterima walaupun duplikasi dapat terjadi.
   (c) ***Exactly once***, dimana pesan dapat dipastikan diterima tepat satu kali.
3. *Transport overhead* yang kecil dan pertukaran protokol yang diminimalisir untuk mengurangi *traffic* pada jaringan.
4. Suatu mekanisme untuk menginformasikan *subsciber* yang tertarik ketika koneksi terputus secara tidak normal.

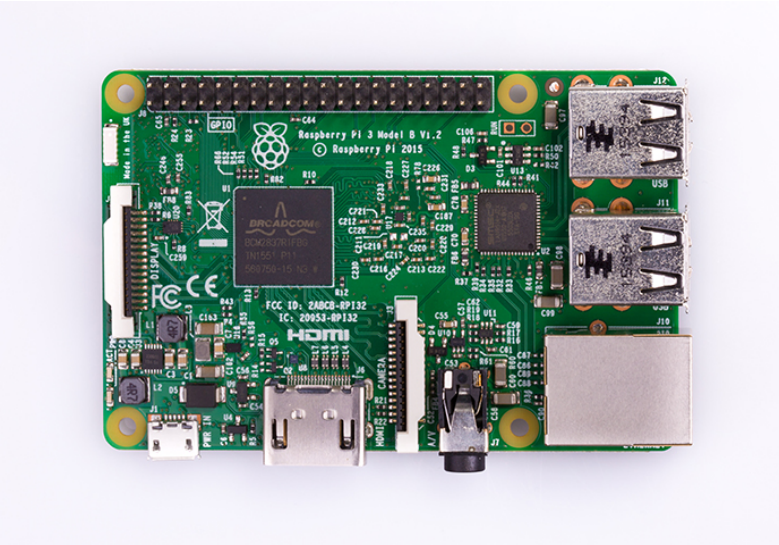
2.3 Raspberry Pi

Raspberry Pi[12] adalah komputer yang berukuran sebesar kartu kredit yang dapat dihubungkan ke layar *monitor* dan *keyboard*. Ini adalah komputer kecil yang dapat digunakan untuk proyek-proyek elektronik seperti yang dilakukan pada komputer biasanya. Hal tersebut seperti membuat dokumen *spreadsheet*, dokumen *word*, berseluncur di *internet*, dan bermain *game*. Terdapat beberapa jenis produk Raspberry Pi, diantaranya :

1. Raspberry Pi 3 Model B, yaitu Raspberry Pi generasi ke 3 yang memiliki spesifikasi tertinggi dan termutakhir. Model Raspberry Pi ini merupakan model utama dari seluruh produk Raspberry Pi.
2. Raspberry Pi Zero, yaitu Raspberry Pi yang ukurannya lebih kecil dari Model utama. Spesifikasi model ini lebih rendah dari yang utama.
3. Raspberry Pi Zero W, yaitu Raspberry Pi yang memiliki

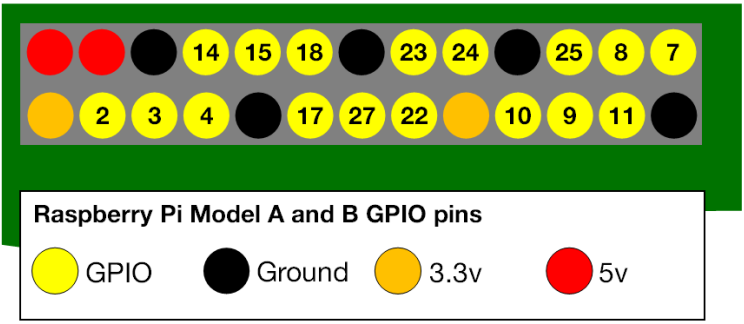
spesifikasi yang sama dengan Raspberry Pi Zero dengan tambahan modul konektifitas seperti WLAN dan Bluetooth.

4. Sense Hat, yaitu *board* tambahan untuk Raspberry Pi, yang memiliki fungsionalitas untuk merekam data lingkungan. Terdapat beberapa sensor yang disediakan oleh *board* ini, diantaranya sensor *gyroscope*, *accelerometer*, *magnetometer*, *temperature*, *barometer*, dan kelembaban.



Gambar 2.2: Raspberry Pi 3 Model B[12]

Salah satu keunggulan perangkat keras Raspberry Pi yaitu pada bagian pin GPIO(*General Purpose Input/Output*)[13]. Pin tersebut adalah penghubung antara Pi dengan perangkat lainnya. 17 dari 26 pin yang tersedia pada Pi adalah pin GPIO, sisanya merupakan pin daya dan GND. Berikut ilustrasi pin pada Pi.



Gambar 2.3: Pin GPIO pada Raspberry Pi Model B[13]

2.4 Bahasa Pemrograman Go

Bahasa pemrograman Go[14] adalah suatu proyek *open source* yang dikembangkan oleh Google. Go adalah bahasa pemrograman yang ekspresif, ringkas, bersih dan efisien. Go merupakan salah satu bahasa pemrograman tingkat rendah, yang dapat mengakses memori secara langsung. Karena hal itu, Go dapat langsung dikompilasi secara cepat ke kode mesin. Tetapi, walaupun bahasa tingkat rendah, Go memiliki fitur *Garbage Collection*[garbage collection] dan *run-time reflection*. Bahasa tersebut juga sangat cepat, bertipe statis, dan bahasa yang butuh dikompilasi yang terasa seperti bahasa dengan tipe dinamis. Bahasa ini terkenal dengan kemampuan mekanisme konkurensi yang mudah untuk diimplementasikan. Dengan bantuan mekanisme konkurensi, membuat program berskala besar secara vertikal (*multi core*) maupun *horizontal* (menggunakan beberapa komputer yang terhubung oleh jaringan) akan menjadi lebih mudah.

2.5 PostgreSQL

PostgreSQL[15] adalah suatu sistem manajemen basis data relasional (RDBMS) yang *open source*. *PostgreSQL* telah dikembangkan selama lebih dari 15 tahun dan memiliki arsitektur yang terbukti keandalan, integritas data dan kebenaran datanya. *PostgreSQL* dapat dijalankan pada sebagian besar sistem operasi, seperti Linux, UNIX, dan Windows. *PostgreSQL* mendukung penuh fitur RDBMS, seperti *foreign key, joins, views, triggers, dan stored procedure*. *PostgreSQL* juga mendukung sebagian besar tipe data yang digunakan pada RDBMS pada umumnya, seperti *integer, numeric, boolean, char, varchar, date, interval*, dan *timestamp*. *PostgreSQL* juga mendukung tipe data yang unik, seperti BLOB(*Binary Large Object*) dan JSON(*Javascript Object Notation*).

2.6 GraphQL

Graphql[16] adalah suatu bahasa *query* yang didesain untuk membangun API(*Application Programming Interface*) dengan menyediakan sintaks yang intuitif dan fleksibel, serta menyediakan suatu sistem untuk menggambarkan kebutuhan data dan cara untuk berinteraksi dengan datanya.
Graphql tidak mengharuskan menggunakan bahasa pemrograman khusus untuk mengimplementasikan desain bahasa *query* ini, melainkan tiap bahasa pemrograman dapat memanfaatkan kapabilitas mereka untuk mengimplementasikan *graphql* sesuai dengan *type system*, prinsip dan filosofi yang dispesifikasikan oleh *graphql*. *Graphql* memiliki beberapa prinsip dalam mendesain bahasa *query*, diantaranya:

1. Hirarkis: Sebagian besar pengembangan produk perangkat lunak saat ini melibatkan pembuatan dan manipulasi tampilan yang berstruktur hirarki. Untuk memfasilitasi hal

tersebut, *graphql* didesain secara hirarki, sehingga data dapat dibentuk sesuai dengan kebutuhan hirarki produk perangkat lunak yang meminta datanya.

2. Berbasis produk: *graphql* dibuat karena kebutuhan akan *Front End Engineers* yang membuat representasi tampilan suatu produk.
3. *Strong typing*: Setiap *server graphql* mendefinisikan aplikasi *type-system* yang spesifik. Semua *query* dieksekusi menggunakan tipe sistem pada konteks yang ada pada spesifikasi. Ketika *query* dijalankan, *graphql* dapat memastikan bahwa *query* tersebut valid secara sintaks dan sesuai dengan *type-system* yang dispesifikasikan *graphql* sebelum dieksekusi.
4. *Client-specified queries*: Dengan menggunakan *type system graphql*, *server graphql* menampilkan kapabilitas data yang dapat dikonsumsi oleh klien. Sisanya, semua tergantung oleh klien untuk menspesifikasikan bagaimana dan seperti apa klien tersebut mengkonsumsi data yang dibutuhkan.
5. *Introspective*: *Type-system server graphql* harus bisa di-*query*-kan oleh bahasa spesifikasi *graphql* untuk introspeksi[16].

Pada *graphql*, terdapat dua tipe spesial, yaitu:

1. *Query* : berfungsi untuk menyediakan data berdasarkan spesifikasi
2. *Mutation* : berfungsi untuk memodifikasi data yang ada pada server.

2.7 Android

Android[17] merupakan sistem operasi perangkat bergerak yang berbasis Linux Kernel dan saat ini sedang dikembangkan oleh Google. Android menggunakan antarmuka pengguna yang berbasis manipulasi langsung, dan didesain terutama untuk digunakan pada perangkat bergerak dengan layar sentuh. Android adalah sistem operasi *open source*, dan dirilis di bawah lisensi Apache. Kode *open source* dan lisensi perizinan pada Android memungkinkan perangkat lunak untuk dimodifikasi secara bebas dan didistribusikan oleh para pembuat perangkat, operator nirkabel, dan pengembang aplikasi. Selain itu, Android memiliki komunitas pengembang aplikasi dengan jumlah besar yang memperluas fungsionalitas perangkat, umumnya ditulis dalam versi bahasa pemrograman Java. Hingga Juni 2017, Android telah mencapai versi 7.0 atau Nougat.

2.8 React

React[18] adalah *library javascript* yang deklaratif, efisien dan fleksibel untuk membuat tampilan/UI(*User Interface*). *React* membantu para *front-end engineers* agar dapat membuat tampilan yang interaktif dengan mudah. *React* memiliki beberapa fitur unggulan, diantaranya :

1. *One-way data flow* : yaitu suatu konsep yang digunakan *react* dalam mengalirkan data dari *model*. Data selalu mengalir searah, sehingga dapat memudahkan pengembang untuk mengerti kode yang ditulis.
2. *Virtual DOM(Document Object Model)* : *react* membuat komponen virtual yang disimpan pada memori, lalu mengkalkulasi perbedaan antara DOM yang ada pada browser dengan DOM virtual, lalu memperbarui DOM yang ada di browser secara efisien menggunakan hasil

kalkulasi tersebut.

3. JSX : yaitu sintaks Javascript yang dibuat mirip seperti sintaks HTML yang digunakan untuk membuat komponen pada *react*.

React menggunakan konsep modularitas untuk pembuatan komponen antarmuka, sehingga dapat menggunakan ulang komponen yang telah dibuat, memastikan konsistensi tampilan dan mempercepat transisi dari proses desain ke proses implementasi tampilan. Selain itu, banyak *UI library*, seperti *ant design*[1], *semantic-ui*[2], *material-ui*[3] dan lain-lain yang berbasis *react* sehingga dapat mempercepat proses pengembangan tampilan pada aplikasi.

2.9 Relay

Relay[19] adalah kerangka kerja yang dibuat oleh Facebook yang menyediakan fungsionalitas pengambilan data pada *graphql server* untuk aplikasi berbasis *react*. Pada *relay* terdapat beberapa terminologi, diantaranya:

- *Container*, yaitu komponen yang berfungsi sebagai wadah penyimpanan data komponen *react*.
- *Query renderer*, yaitu komponen *relay* yang berfungsi untuk mengeksekusi *query graphql*.
- *Fragment container*, yaitu komponen *relay* yang berfungsi sebagai wadah penyimpanan data *react* yang dapat mendeklarasikan kebutuhan datanya sendiri.
- *Commit mutation*, yaitu metode yang digunakan pada *relay* untuk mengeksekusi *mutation* pada *graphql server*.

[1] Ant Design dapat diakses di ant.design
[2] Semantic UI dapat diakses di react.semantic-ui.com
[3] Material UI dapat diakses di material-ui.com

2.10 Redux

Redux[20] adalah kerangka kerja javascript untuk mengelola dan memelihara *state* aplikasi yang biasanya dibangun dengan kerangka kerja lain untuk membuat aplikasi web. Terdapat beberapa istilah pada kerangka kerja *redux*, diantaranya:

- *State*, yaitu keadaan suatu aplikasi.
- *Store*, yaitu suatu obyek yang menyimpan *state* aplikasi.
- *Actions*, yaitu suatu deskripsi kejadian eksplisit yang terjadi pada aplikasi.
- *Reducers*, yaitu fungsi yang dijalankan untuk mengubah *state* aplikasi ketika *actions* terjadi.

2.11 Ant Design

Ant design[21] merupakan suatu metode desain berkelas *enterprise* yang diimplementasikan menggunakan *framework react*. Awalnya, *Ant design* dibuat oleh Ant UED Team yang memiliki tujuan untuk menyeragamkan spesifikasi antarmuka pengguna untuk proyek internal perusahaan, sehingga dapat mengurangi perbedaan desain dan implementasi tampilan, serta memudahkan pengembangan tampilan sistem serta implementasi *front end*. *Ant design* telah dipakai oleh beberapa perusahaan terkenal yang berasal dari China, seperti Ant Financial, Alibaba dan Didi Chuxing.

Ant design memiliki komponen berkualitas tinggi yang diimplementasikan menggunakan *framework react* yang dapat digunakan untuk membuat antarmuka pengguna yang interaktif. Komponen tersebut diantaranya:

1. Komponen umum, yaitu *button* dan *icon*.
2. *Layout*, yaitu *grid* dan *layout*.
3. Navigasi, yaitu *affix*, *breadrumb*, *backTop*, *dropdown*, *menu*, *pagination*, *steps* dan *tabs*.

4. Entri data, yaitu *autocomplete*, *checkbox*, *form*, *input*, *inputnumber*, *radio*, *select* dan lainnya.
5. Menampilkan data, yaitu *badge*, *card*, *tooltip*, *popover*, *table*, *tag*, *timeline* dan lainnya.
6. *feedback*, yaitu *alert*, *modal*, *message*, *notification*, *progress*, *popconfirm* dan *spin*.

2.12 Python

Python[22] adalah bahasa pemrograman yang interaktif, menggunakan teknik interpretasi, dan berorientasi obyek. Python menyediakan tipe data dinamis tingkat tinggi seperti *list* dan *dictionaries*, modul, *exceptions*, kelas, manajemen memori secara otomatis, dan lain-lain. Sintaks yang digunakan pada bahasa pemrograman Python sangat simpel dan elegan sehingga mudah dipelajari. Seperti bahasa pemrograman lainnya, Python merupakan bahasa pemrograman yang *open-source*, sehingga *developer* dapat memodifikasi dan mendistribusikan ulang, bahkan untuk kebutuhan komersial. Python dapat berjalan di berbagai sistem operasi, seperti Mac, Windows dan Linux. Selain itu, terdapat banyak *library* yang dikembangkan oleh *developer* di seluruh dunia untuk menyelesaikan studi kasus tertentu, seperti *machine learning*[4], *natural language processing*[5], dan klien untuk mekanisme *publish/subscribe*[6].

[4] *Library machine learning* dapat diakses di scikit-learn.org
[5] *Library natural language processing* dapat diakses di nltk.org
[6] *Publish/subscribe client* dapat diakses di eclipse.org/paho/clients/python

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN DAN PERANCANGAN

3.1 Deskripsi Umum Sistem

Sistem yang akan dibuat yaitu sistem yang dapat merekam data lingkungan sesuai dengan sensor yang dipakai serta mengirimkan datanya kepada sistem secara *real-time*. Terdapat *smartphone* Android ditujukan untuk memonitor data sensor yang sedang direkam secara *real-time*. Sedangkan pada platform Web, pengguna dapat menampilkan riwayat rekaman data seluruh sensor beserta melakukan konfigurasi otomatisasi aksi pada sistem kolaborasi. Otomatisasi aksi tersebut dapat dikonfigurasi dengan menyetel aturan pada sensor yang tersedia. Aturan sensor tersebut menggunakan operator relasional(<, <=, ==, >=, dan >) dan operator logika(*and* dan *or*). Aturan yang telah dikonfigurasi tersebut dinamakan *rule expression*. Lalu, ketika nilai pada sensor memenuhi *rule expression*, sistem akan menjalankan aksi tertentu sesuai dengan aksi pilihannya. Selain menjalankan aksi, sistem akan menyimpan rekaman data sensor pada saat itu. Data sensor ketika *rule expression* terpenuhi tersebut dapat dilihat pada *platform* web. Selain itu, sistem ini menggunakan mekanisme *publish/subscribe* agar sistem dapat melakukan komunikasi lintas *platform* secara *real-time*.

3.2 Tabel Istilah

Tabel istilah memuat istilah yang dipergunakan dalam tugas akhir ini disertai keterangan berupa arti ataupun maknanya. Daftar istilah yang digunakan pada tugas akhir ini tertera pada tabel 3.1.

Tabel 3.1: Tabel istilah

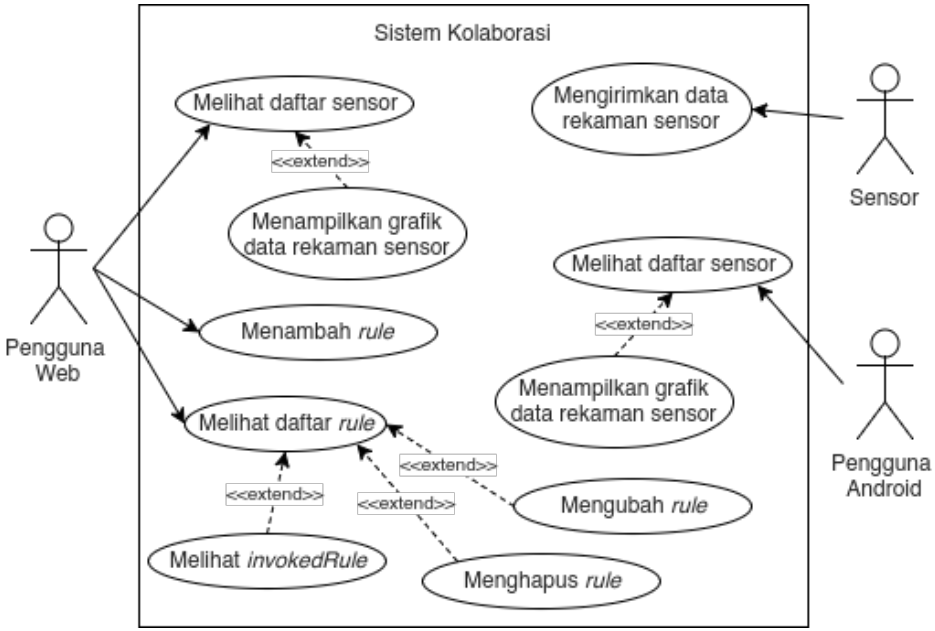
Rule expression	Aturan nilai pada suatu sensor
Rule	Kumpulan *rule expression* yang memicu sistem untuk melakukan otomatisasi aksi

Tabel 3.1: Tabel istilah

Invoked rule	Data yang berisi nama *rule* dan data rekaman sensor ketika suatu *rule* terpenuhi
Action	Aksi yang dapat dilakukan oleh sistem
AutoAI	Komponen sistem yang berfungsi untuk melakukan otomatisasi aksi

3.3 Kasus Penggunaan

Berdasarkan deskripsi umum sistem, terdapat tiga aktor dalam sistem, yaitu perangkat sensor, pengguna aplikasi Android, dan pengguna aplikasi Web. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Diagram kasus penggunaan

Diagram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.2

Tabel 3.2: Daftar kode kasus penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Mengirimkan data rekaman sensor.	Perangkat sensor mengirimkan data hasil rekaman lingkungan menggunakan sensor.
UC-0002	Melihat daftar sensor.	Pengguna Android dapat melihat daftar sensor yang tersedia pada sistem.
UC-0003	Menampilkan grafik data rekaman sensor.	Pengguna Android dapat menampilkan grafik data rekaman sensor yang dipilih.
UC-0004	Melihat daftar sensor.	Pengguna web dapat melihat daftar sensor yang tersedia pada sistem.
UC-0005	Menampilkan grafik data rekaman sensor.	Pengguna web dapat menampilkan grafik data rekaman sensor yang dipilih beserta data riwayat rekamannya.
UC-0006	Menambah *rule*.	Pengguna web dapat menambahkan *rule* untuk melakukan otomatisasi aksi.

Tabel 3.2: Daftar kode kasus penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0007	Melihat daftar *rule*.	Pengguna web dapat melihat daftar *rule* yang terdaftar pada sistem.
UC-0008	Mengubah *rule*.	Pengguna web dapat mengubah *rule* yang terdaftar pada sistem.
UC-0009	Menghapus *rule*.	Pengguna web dapat menghapus *rule* yang terdaftar pada sistem.
UC-0010	Melihat *invokedRule*.	Pengguna web dapat melihat data rekaman sensor beserta waktu ketika *rule* terpenuhi.

Khusus ketika UC-0001 dilakukan, sistem melakukan tiga fungsionalitas lainnya, yaitu menyimpan data rekaman sensor pada basis data, mengevaluasi *rule* yang tersimpan pada basis data, dan melakukan aksi otomatis serta menyimpan data rekaman seluruh sensor ke basis data jika *rule expression* terpenuhi.

3.4 Arsitektur Sistem

Pada subbab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun.

3.4.1 Desain Umum Sistem

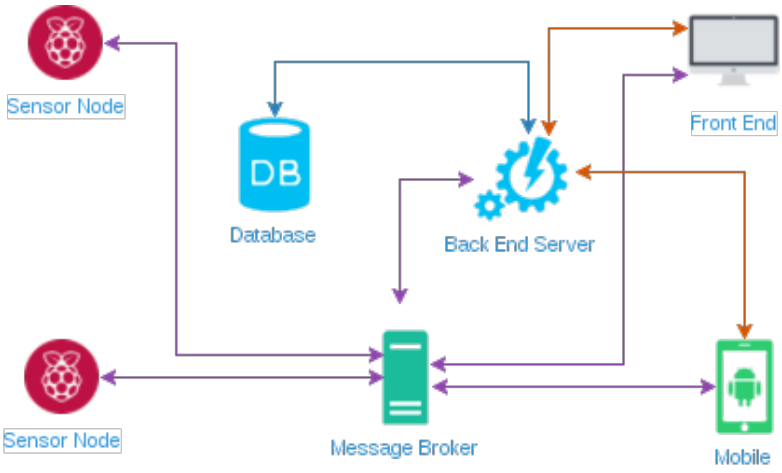
Berdasarkan deskripsi umum sistem dan kasus penggunaan yang telah ditulis pada bab sebelumnya, dapat diperoleh kebutuhan sistem ini, diantaranya :

1. Perekaman data lingkungan menggunakan sensor.
2. Komunikasi lintas *platform* secara *real-time*.
3. *Monitoring* data sensor pada aplikasi Android.
4. Menampilkan data sensor pada aplikasi Web.
5. Manajemen data *rule* pada aplikasi Web.
6. Otomatisasi aksi berdasarkan *rule expression*.

Sistem dibagi menjadi beberapa komponen untuk memenuhi kebutuhan sistem tersebut, diantaranya:

1. *Message broker*
   Berfungsi agar sistem dapat melakukan komunikasi lintas *platform* secara *real-time*.
2. *Node* sensor
   Berfungsi sebagai perekam data lingkungan sesuai dengan sensor yang dipakai.
3. *Backend*
   Berfungsi untuk menjalankan aksi secara otomatis berdasarkan *rule expression*, serta menyimpan dan menyediakan data sensor dan *rule* yang dibutuhkan oleh *end-user* pada *platform* Android maupun Web. Sehingga, dibutuhkan *database* untuk penyimpanan datanya.
4. Aplikasi pada *platform* Android
   Berfungsi untuk memonitor data sensor yang sedang direkam secara *real time*.
5. Aplikasi pada *platform* Web
   Berfungsi untuk menampilkan sensor yang tersedia beserta riwayat rekaman datanya serta manajemen *rule* untuk otomatisasi aksi.

Ilustrasi arsitektur sistem beserta komponen- komponennya secara umum tertera pada Gambar 3.2.



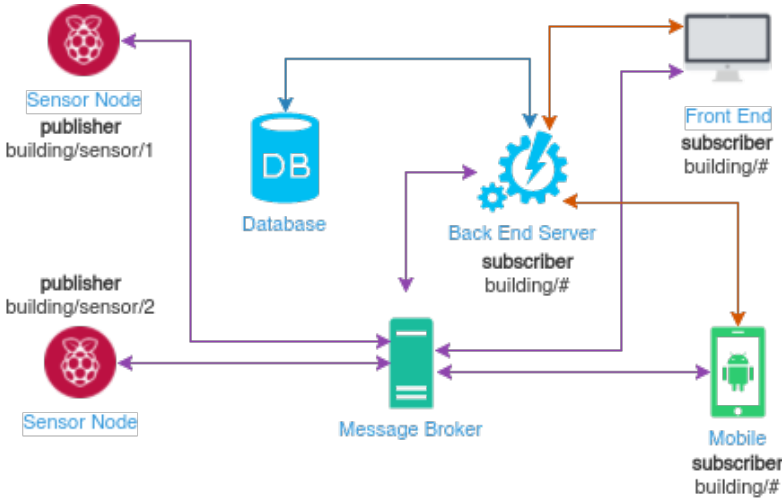
Gambar 3.2: Arsitektur sistem secara umum

3.4.2 Message Broker

Dalam *paper* yang ditulis oleh Eugster et al.[11], untuk menerapkan metode komunikasi *publish/subscribe*, dibutuhkan *neutral mediator* diantara *publisher* dan *subscriber* yang berfungsi sebagai penyimpan informasi *subscriber* beserta topik yang dipilih sebagai penerus pesan yang dikirim oleh *publisher* menuju *subscriber* yang relevan. Untuk menerapkan metode komunikasi *publish/subscribe*, penulis memilih protokol MQTT karena memiliki banyak keunggulan, seperti menggunakan *bandwidth* yang rendah, sangat ringan, dapat berjalan pada jaringan yang tidak *reliable*[9]. Pada protokol MQTT, *neutral mediator* tersebut dinamakan *message broker*[23]. Lalu, teknologi yang dipakai sebagai MQTT *broker* yaitu *mosquitto*[1]. Penulis memilih *mosquitto* sebagai MQTT *broker* karena teknologi ini bersifat *open source* dan merupakan proyek inkubasi yang aktif dikembangkan oleh Eclipse Foundation.

[1] Mosquitto dapat diunduh di https://mosquitto.org

Perancangan topik untuk dapat menggunakan metode komunikasi *publish/subscribe* tertera pada Gambar 3.3



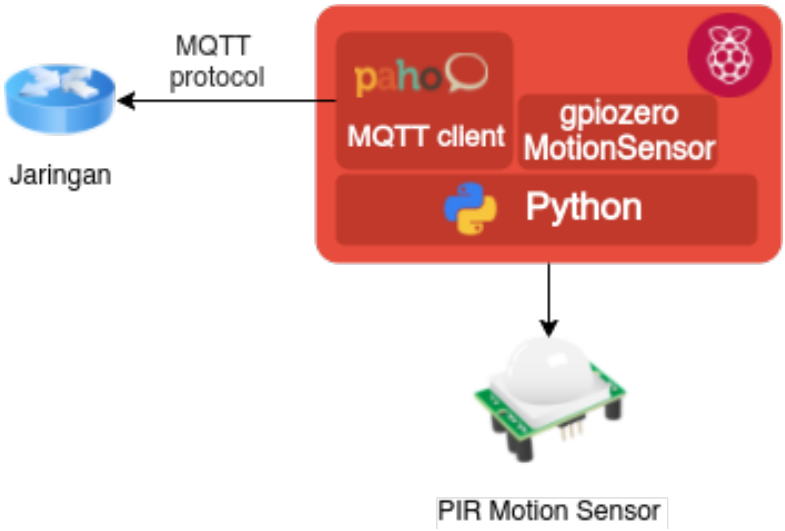
Gambar 3.3: Perancangan topik pada sistem

3.4.3 Perancangan *Node* Sensor

Node sensor adalah komponen pada sistem yang berfungsi untuk merekam data lingkungan. Data tersebut tergantung dengan sensor yang dipakai. Pada tugas akhir ini, *PIR motion sensor* dipakai untuk merekam data lingkungan. Sensor tersebut dipasang pada Raspberry Pi agar dapat mengirim datanya ke *subscriber*. Karena untuk dapat berkomunikasi dengan protokol MQTT membutuhkan koneksi TCP, Raspberry Pi yang dipilih yaitu Raspberry Pi 1 Model B.

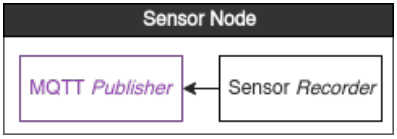
Lalu, dari sisi perangkat lunak, bahasa pemrograman yang dipilih penulis untuk mengimplementasikan kode untuk mengirim data dari sensor ke *subscriber* yaitu bahasa pemrograman Python. Bahasa tersebut dipilih karena terdapat

library untuk merekam data sensor gerakan. *Library* yang digunakan yaitu *gpiozero.MotionSensor*. Selain itu, agar perangkat dapat berkomunikasi menggunakan protokol MQTT, dibutuhkan *library* tambahan yaitu *eclipse paho python client*. Sehingga arsitektur komponen pada *node* sensor menjadi seperti Gambar 3.4.



Gambar 3.4: Arsitektur *node* sensor

Pada tugas akhir ini, *node* sensor membutuhkan dua fungsi utama untuk mengirim data rekaman lingkungan ke sistem. Fungsi pertama yaitu *SensorRecorder* yang berfungsi untuk merekam data lingkungan. Fungsi kedua yaitu *MQTTPublisher* yang berguna untuk mengirim data rekaman sensor ke sistem. Ilustrasi arsitektur perangkat lunak *node* sensor tertera pada Gambar 3.5.



Gambar 3.5: Arsitektur perangkat lunak *node* sensor

3.4.4 Perancangan *Backend*

Menurut *Oxford Dictionaries*[24], *backend* (dalam istilah komputer) adalah suatu bagian dari sistem atau aplikasi yang tidak diakses langsung oleh pengguna, yang biasanya digunakan untuk menyimpan dan memanipulasi data. Berdasarkan desain umum sistem, komponen *backend* memiliki fungsi untuk menyelesaikan permasalahan otomatisasi aksi, menyediakan data yang dibutuhkan oleh aplikasi Web dan Android, serta tempat penyimpanan data. Karena kebutuhan yang cukup luas, *backend* dibagi lagi menjadi beberapa sub komponen, diantaranya:

1. Basis data
   Berfungsi sebagai tempat penyimpanan segala data yang digunakan dan dibutuhkan oleh sistem.
2. *Graphql server*
   Graphql server berfungsi untuk menyediakan data pada aplikasi Web dan Android yang dipakai oleh *end-user*. Sehingga pengguna dapat mengakses data sensor serta memanajemen data *rule* secara terpisah.
3. *Auto Action Invoker*
   Berfungsi sebagai *engine* untuk melakukan otomatisasi aksi berdasarkan *rule* dan data sensor yang diterima.

Pada tugas akhir ini, penulis memilih *postgresql* sebagai *RDBMS* yang dipakai untuk penyimpanan data yang dibutuhkan oleh sistem. Lalu, bahasa Go dipilih sebagai bahasa pemrograman yang digunakan untuk mengimplementasikan komponen *backend*. Bahasa Go dipilih karena fitur

pemrograman paralel yang mudah untuk digunakan, serta merupakan bahasa tingkat rendah, sehingga dapat berjalan dengan sangat cepat dan efisien. Oleh karena itu, arsitektur sistem pada bagian *backend* dibuat menyesuaikan dengan bahasa yang dipilih sehingga kode yang dihasilkan dapat dengan mudah dikembangkan.

Berdasarkan *The Clean Architecture*[25] yang ditulis oleh Uncle Bob, terdapat beberapa lapisan pada arsitektur sistem, yaitu lapisan entitas/domain sebagai obyek utama pada sistem, lapisan kasus penggunaan/implementasi sistem sebagai implementasi kasus penggunaan yang terjadi pada sistem, lapisan antarmuka sebagai penghubung antara lapisan implementasi dengan *library* yang digunakan, dan lapisan kerangka kerja/*framework*. Sistem ini hanya menggunakan tiga lapisan karena tidak menggunakan kerangka kerja. Tiga lapisan tersebut yaitu lapisan entitas/domain yang dirancang pada bagian desain basis data, lapisan antarmuka yang digunakan sebagai penghubung antara kasus penggunaan pada komponen *backend* dengan penyimpanan data, dan lapisan implementasi sebagai implementasi dari kebutuhan sistem pada komponen *backend* yang dirancang pada sub komponen *graphql* dan *Auto Action Invoker*.

3.4.4.1 Desain Basis Data

Komponen basis data berfungsi sebagai tempat penyimpanan segala data yang digunakan dan dibutuhkan oleh sistem agar sistem dapat menjalankan fungsinya. Berdasarkan deskripsi umum sistem, terdapat lima entitas yang perlu disimpan dan tersedia pada sistem ini, yaitu sensor, data sensor, *rule*, *action*, dan *invoked rule*. Berikut detail analisis tiap entitasnya:

1. Entitas sensor
   Entitas ini berfungsi untuk menyimpan daftar sensor yang

didaftarkan pada sistem. Hal ini dilakukan agar sistem dapat memperoleh daftar sensor dan dapat memproses otomatisasi aksi berdasarkan sensor yang terdaftar. Terdapat beberapa atribut yang dibutuhkan sensor, diantaranya:

Tabel 3.3: Analisis entitas sensor

Nama	**Tujuan**
ID	ID sensor
Name	Nama sensor
Status	Status sensor(aktif/tidak aktif)

2. Entitas data sensor
   Entitas ini berfungsi untuk menyimpan data hasil rekaman sensor. Hal ini dilakukan agar sistem dapat memperoleh data hasil rekaman sensor. Berikut atribut yang dibutuhkan pada entitas data sensor.

Tabel 3.4: Analisis entitas data sensor

Nama	**Tujuan**
ID	ID unik tiap data
sensorID	ID sensor
val	data hasil rekaman sensor
time	waktu perekaman data

3. Entitas *action*
   Entitas ini berfungsi untuk menyimpan aksi otomatis yang tersedia pada sistem. Berikut atribut yang dibutuhkan pada entitas *action*:

Tabel 3.5: Analisis entitas *action*

Nama	**Tujuan**
ID	ID *action*

Tabel 3.5: Analisis entitas *action*

Nama	Tujuan
Name	Nama *action*
CallbackFn	Fungsi yang dijalankan pada server

4. Entitas *rule*
   Entitas ini berfungsi untuk menyimpan *rule expression* yang dapat dikonfigurasi oleh pengguna menggunakan aplikasi Web. Entitas ini juga memiliki relasi dengan entitas *action* karena aksi akan secara otomatis dilakukan ketika *rule expression* terpenuhi. Berikut atribut yang dibutuhkan pada entitas *rule*:

Tabel 3.6: Analisis entitas *rule*

Nama	Tujuan
ID	ID *rule*
actionID	ID *action* yang akan dijalankan
Name	Nama *rule*
Index	Urutan *rule* yang ditampilkan
Status	Status *rule*(aktif/tidak aktif)
RuleExpression	Konfigurasi aturan data rekaman sensor

5. Entitas *invoked rule*
   Entitas ini berfungsi untuk menyimpan data sensor dan *rule* ketika *rule* tersebut terpenuhi. Entitas ini memiliki relasi dengan entitas *rule*. Berikut atribut yang dibutuhkan pada entitas *invoked rule*:

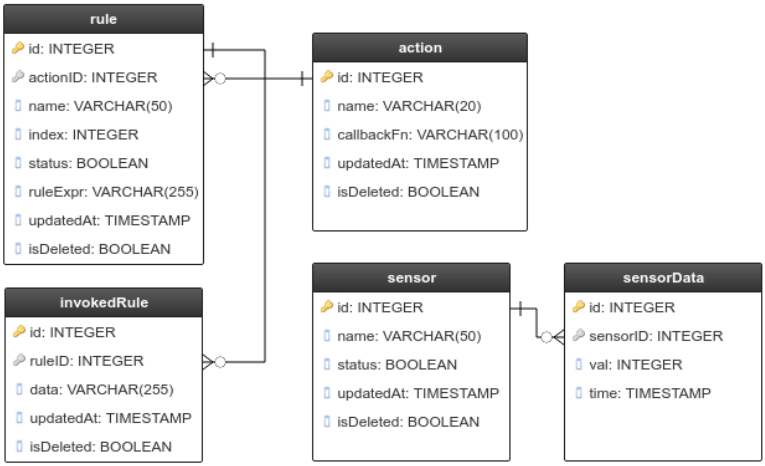
Tabel 3.7: Analisis entitas *invoked rule*

Nama	Tujuan
ID	ID unik tiap data
ruleID	ID *rule* yang terpenuhi

Tabel 3.7: Analisis entitas *invoked rule*

Nama	Tujuan
Data	Data sensor ketika *rule* terpenuhi

Pada setiap entitas ditambahkan kolom *createdAt* sebagai catatan waktu data dibuat, *updatedAt* untuk waktu terakhir data diubah dan *isDeleted* agar data yang dihapus tetap tersimpan pada basis data. Berikut hasil rancangan basis data sistem ini:



Gambar 3.6: Rancangan basis data

3.4.4.2 Desain GraphQL Server

Agar sistem pada aplikasi Android dan Web dapat menampilkan data yang disimpan pada basis data, aplikasi pada *platform* tersebut perlu mengakses basis data. Tetapi, aplikasi tersebut tidak bisa mengakses langsung basis data dikarenakan arsitektur sistem yang terpisah, sehingga dibutuhkan penghubung antara aplikasi dengan basis data, yaitu *middleware*.

Pada sistem ini, penulis memilih *graphql* sebagai jenis *middleware* yang digunakan. Karena dengan menggunakan *graphql*, aplikasi dapat meminta data sesuai kebutuhan yang dispesifikasikan sehingga tidak terjadi *overfetching* serta dapat meminta banyak data dari sumber berbeda hanya dengan satu *request* ke *server graphql*.

Graphql server dibuat dengan cara mendefinisikan tipe obyek dan *field* yang ada pada tiap tipe obyeknya. Lalu tiap tipe tersebut disusun menjadi suatu skema yang dapat diakses oleh aplikasi lain.

Pada sistem ini, aplikasi Web dan Android membutuhkan data berikut untuk ditampilkan:

1. Daftar sensor, terdiri dari *field* ID, nama sensor, dan data sensor.
2. Detail sensor, terdiri dari *field* ID, nama sensor, dan data sensor.
3. Data sensor terdiri dari *field* val dan time.
4. Daftar *rule*, terdiri dari *field* ID, nama, *rule expression* dan *action* yang dipilih.
5. Daftar *invoked rule* terdiri dari *field* ID, nama *rule*, dan data sensor ketika *rule* tersebut terpenuhi.
6. Daftar *action*, terdiri dari *field* ID dan nama *action*.

Lalu, agar *rule expression* dapat dikonfigurasi secara dinamis, *rule* harus dapat dimodifikasi menggunakan aplikasi pada platform Web, sehingga dibutuhkan tipe *mutation* pada *graphql server*. Berikut *mutation* yang dibuat:

1. Menambahkan *rule*, dengan *input*: nama, *rule expression* dan *action* yang dipilih.
2. Mengubah *rule*, dengan *input*: ID yang diubah, nama, *rule expression* dan *action* yang dipilih.
3. Menghapus *rule*, dengan *input*: ID yang akan dihapus.

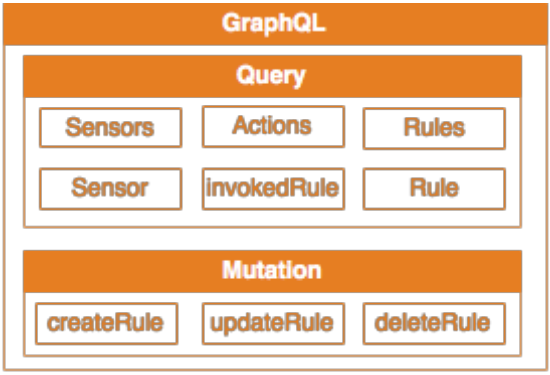
Berikut skema *graphql* yang dibuat berdasarkan analisis tersebut:

```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  sensors: [Sensor]
  sensor(id: ID!): Sensor
  rules: [Rule]
  rule(id: ID!): Rule
  invokedRules(ruleid: ID!): [InvokedRule]
  actions: [Action]
}

type Mutation {
  createRule(rule: RuleInput): Rule
  updateRule(input: RuleUpdateInput): Rule
  deleteRule(input: RuleDeleteInput): Rule
}

type Sensor {
  id: ID!
  name: String!
  sensordata(limit: Int): [SensorData]
}

type SensorData {
  val: Int!
  time: Int!
}

type Rule {
  id: ID!
  name: String!
  index: Int!
  status: Boolean
  rule: String!
  actionID: Int!
}

input RuleInput {
  name: String
  index: Int
  status: Boolean
  rule: String
  actionID: Int
}
```

```
48
49  input RuleUpdateInput {
50    id: ID!
51    rule: RuleInput
52  }
53
54  input RuleDeleteInput {
55    id: ID!
56  }
57
58  type InvokedRule {
59    id: ID!
60    rulename: String
61    data: String
62    time: String!
63  }
64
65  type Action {
66    id: ID!
67    name: String
68  }
```

Kode Sumber 3.1: Skema *graphql*

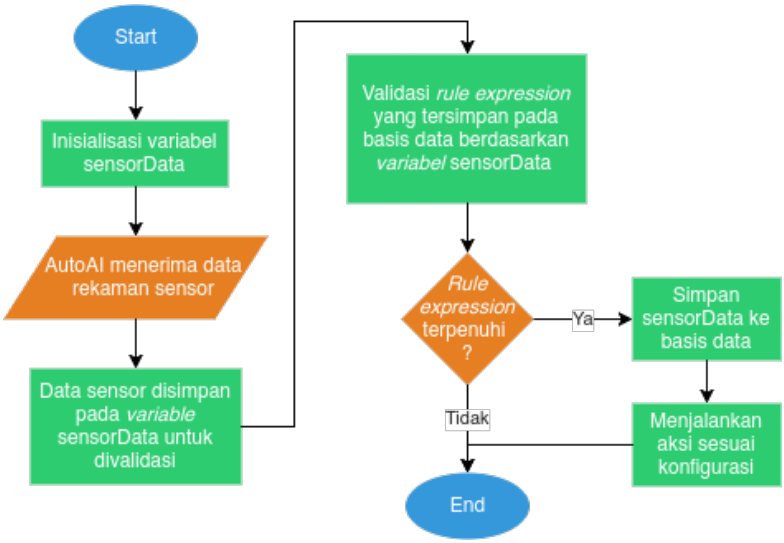
Ilustrasi desain *graphql* tertera pada Gambar 3.7



Gambar 3.7: Ilustrasi desain *graphql*

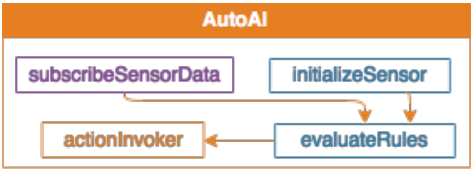
3.4.4.3 Desain *Auto Action Invoker*

Auto Action Invoker(AutoAI) merupakan *engine* untuk mengevaluasi *rule expression* dan melakukan aksi berdasarkan data lingkungan yang diterima. Berikut diagram alur untuk melakukan otomatisasi aksi berdasarkan *rule expression*:



Gambar 3.8: Diagram alur AutoAI

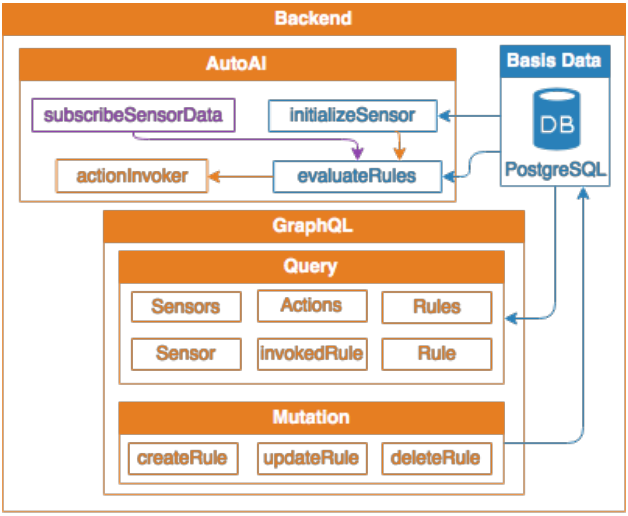
Berdasarkan diagram alur pada Gambar 3.8, AutoAI membutuhkan empat fungsi utama, yaitu *InitializeSensor* yang berfungsi untuk menginisialisasi variabel sensor dari basis data, *subscribeSensorData* yang berfungsi untuk menerima data rekaman sensor dari MQTT *broker*, *evaluateRules* untuk mengevaluasi *rule expression* ketika data rekaman sensor diterima oleh AutoAI, dan *actionInvoker* yang berfungsi untuk melakukan aksi ketika *rule* terpenuhi. Ilustrasi arsitektur perangkat lunak AutoAI tertera pada Gambar 3.9.



Gambar 3.9: Arsitektur perangkat lunak AutoAI

3.4.4.4 Arsitektur Perangkat Lunak pada Komponen *Backend*

Berdasarkan hasil analisis pada tiap sub komponen yang terdapat pada *backend*, dihasilkan arsitektur perangkat lunak pada komponen *backend*. Ilustrasi arsitektur perangkat lunak pada komponen *backend* tertera pada Gambar 3.10.



Gambar 3.10: Arsitektur perangkat lunak komponen *Backend*

3.4.5 Desain Aplikasi pada *Platform* Android

Pada sistem ini, aplikasi Android berguna untuk menampilkan data hasil rekaman sensor secara *real-time*. Aplikasi ini juga menampilkan data berupa grafik untuk memvisualisasikan data sensor yang direkam. Dalam membangun aplikasi ini, penulis membagi metode perancangan menjadi tiga bagian, diantaranya:

1. analisis kebutuhan aplikasi.
2. desain antarmuka aplikasi.
3. alur kerja aplikasi.

3.4.5.1 Analisis Kebutuhan Aplikasi Android

Berdasarkan deskripsi umum aplikasi Android, terdapat beberapa hal yang dibutuhkan untuk membuat aplikasi ini, diantaranya:

1. *Graphql client*, untuk mengambil daftar sensor yang tersedia pada *server*.
2. MQTT *client*, untuk menerima data rekaman sensor.
3. *Chart library*, untuk menampilkan grafik pada aplikasi Android.

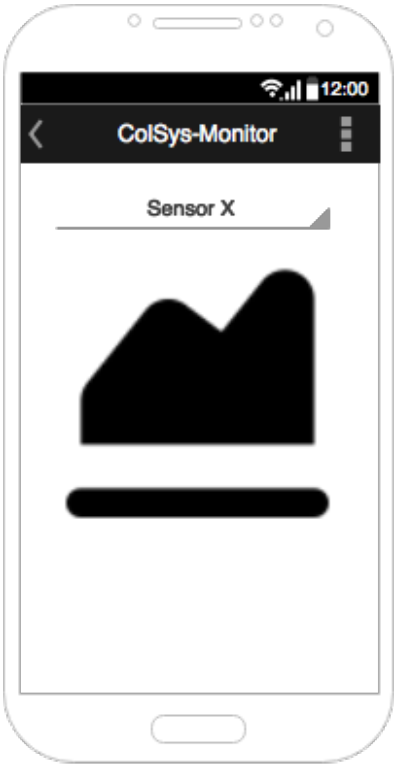
Sehingga, dipilihlah *library* dibawah ini untuk memenuhi kebutuhan aplikasi ini:

1. Apollo Android *client*, sebagai *graphql client*.
2. Android Paho *client*, sebagai MQTT *client*.
3. MPAndroidChart, sebagai *library* untuk menampilkan grafik.

3.4.5.2 Desain Antarmuka Aplikasi Android

Pada sistem ini, pengguna dapat memilih sensor apa yang ingin dimonitor. Lalu, akan tampil data hasil rekaman sensor yang telah dipilih secara *real-time*. Sehingga dibutuhkan dua objek tampilan pada aplikasi Android, yaitu *spinner/combobox*

dan *chart* menggunakan *mpandroidchart*. Desain antarmuka aplikasi Android untuk sistem ini tertera pada Gambar 3.11

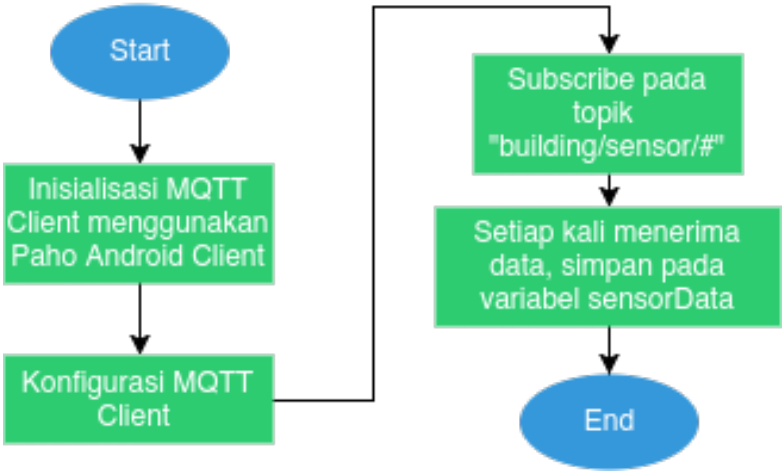


Gambar 3.11: Mockup antarmuka pada Android

3.4.5.3 Alur Kerja Aplikasi Android

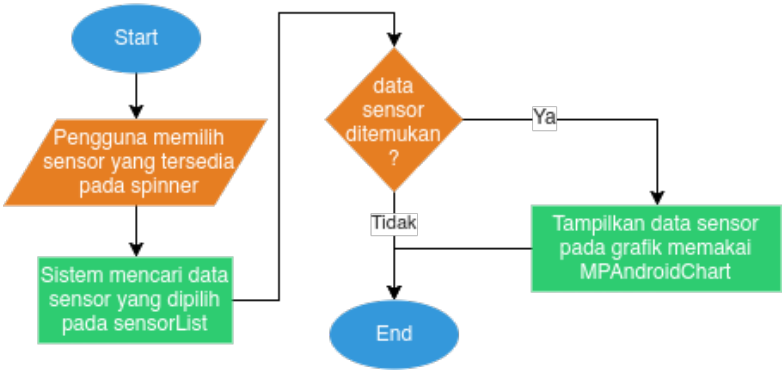
Terdapat beberapa alur kerja pada aplikasi android, diantaranya:

1. Alur kerja menerima data hasil rekaman sensor



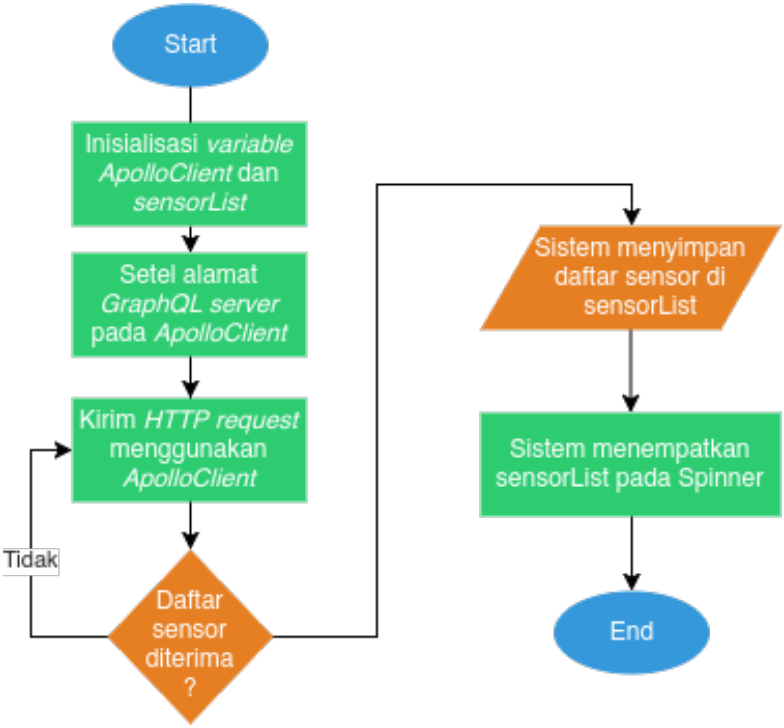
Gambar 3.12: Alur kerja menerima data hasil rekaman sensor

2. Alur kerja menampilkan grafik berdasarkan pilihan pengguna



Gambar 3.13: Alur kerja menampilkan grafik berdasarkan pilihan pengguna

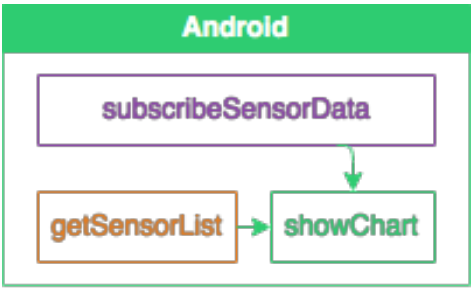
3. Alur kerja mengambil daftar sensor dari *graphql server*



Gambar 3.14: Alur kerja mengambil daftar sensor dari *graphql Server*

Berdasarkan alur kerja yang telah dibuat, dihasilkan arsitektur perangkat lunak umum pada aplikasi Android. Aplikasi android membutuhkan tiga fungsi utama. Fungsi pertama yaitu *getSensorList* yang berfungsi untuk mengambil daftar sensor dari *graphql server*. Fungsi kedua yaitu *subscribeSensorData* yang berfungsi untuk menerima data rekaman sensor dari MQTT *broker*. Fungsi selanjutnya yaitu *showChart* yang berfungsi untuk menampilkan grafik data rekaman sensor ketika pengguna memilih sensor pada *spinner*.

Ilustrasi arsitektur perangkat lunak pada aplikasi Android tertera pada Gambar 3.15.



Gambar 3.15: Arsitektur perangkat lunak aplikasi Android

3.4.6 Desain Aplikasi pada *Platform* Web

Pada aplikasi Web ini, pengguna dapat melihat daftar sensor dan daftar *rule* yang tersimpan pada *database*. Lalu, pengguna juga dapat melihat data rekaman sensor secara *real-time* dalam bentuk grafik. Selain itu, pengguna juga dapat menambah, mengubah dan menghapus *rule*. Pada tiap *rule* terdapat nama *rule* dan *rule expression* berdasarkan sensor yang dipilih serta *action* yang dilakukan ketika *rule expression* terpenuhi.

Untuk membangun aplikasi Web tersebut, penulis membagi menjadi beberapa bagian, diantaranya:

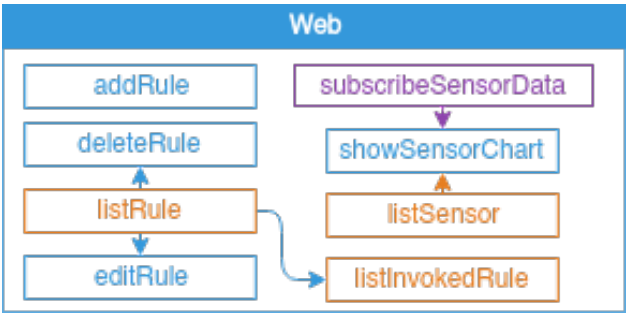
1. Analisis kebutuhan pada aplikasi Web
2. Desain antarmuka
3. Perancangan komponen pada aplikasi Web

3.4.6.1 Analisis Kebutuhan pada Aplikasi Web

Analisis kebutuhan pada subbab ini dibagi menjadi dua bagian, yaitu analisis teknologi dan analisis arsitektur perangkat lunak. Berdasarkan deskripsi umum aplikasi Web, teknologi yang dibutuhkan untuk aplikasi Web, diantaranya (1) *library*

untuk mempermudah pembuatan tampilan. (2) CSS *framework* untuk membuat tampilan menjadi interaktif dan elegan. (3) *graphql client*, untuk mengambil daftar sensor dan memodifikasi data *rule* yang tersedia pada *graphql server*. (4) MQTT *client*, agar dapat menerima data sensor secara *real-time*. (5) *chart library*, untuk menampilkan data sensor dalam bentuk grafik.

Untuk memenuhi kebutuhan diatas, penulis memilih beberapa teknologi, diantaranya (1) *React*, sebagai *library* dasar pembuatan tampilan. (2) *Ant design*, sebagai komponen CSS untuk membuat tampilan menjadi interaktif dan elegan. (3) *Relay*, sebagai *graphql client* karena konsep berbasis komponen yang diadopsi dari *React* dan kemudahan dalam penggunaan. (4) MQTT.js, sebagai MQTT *client* untuk menerima data sensor. (5) *Recharts*, sebagai *library* untuk menampilkan data sensor dalam bentuk grafik. Berdasarkan kasus penggunaan dan deskripsi umum aplikasi Web, terdapat beberapa fungsi yang perlu diimplementasikan. Ilustrasi arsitektur perangkat lunak pada aplikasi Web tertera pada Gambar 3.16.

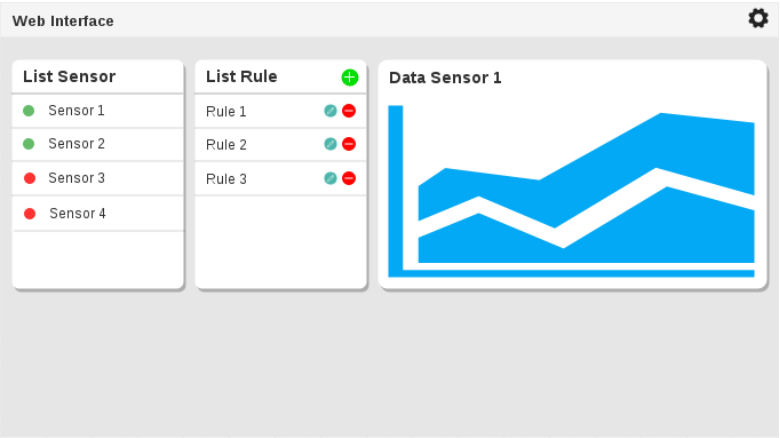


Gambar 3.16: Arsitektur perangkat lunak aplikasi Web

3.4.6.2 Desain Antarmuka

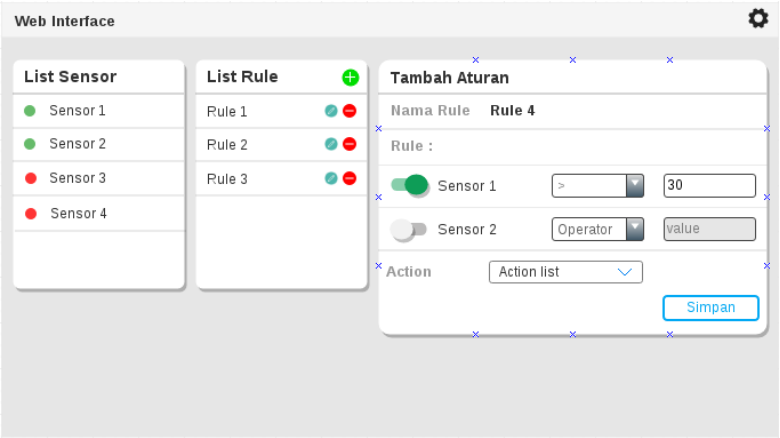
Berikut antarmuka tampilan untuk memenuhi kebutuhan tersebut.

1. Antarmuka halaman *dashboard*
   Halaman *dashboard* berfungsi untuk menampilkan daftar sensor, daftar *rule* beserta data sensor secara *real-time*. Desain antarmuka komponen *dashboard* tertera pada Gambar 3.17.



Gambar 3.17: Halaman antarmuka *Dashboard*

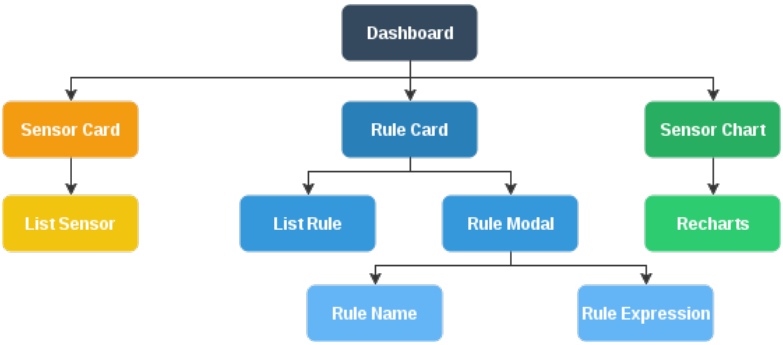
2. Antarmuka manajemen *rule*
   Antarmuka manajemen *rule* berfungsi untuk menambah atau mengubah data *rule* yang ada. Pada bagian ini, *rule expression* dapat dikonfigurasi menggunakan sensor yang tersedia dengan operator relasional (<, <=, ==, >= dan >) dan operator logika(*and* dan *or*). Desain antarmuka komponen manajemen *rule* tertera pada Gambar 3.18.



Gambar 3.18: Halaman manajemen *rule*

3.4.6.3 Perancangan Komponen pada Aplikasi Web

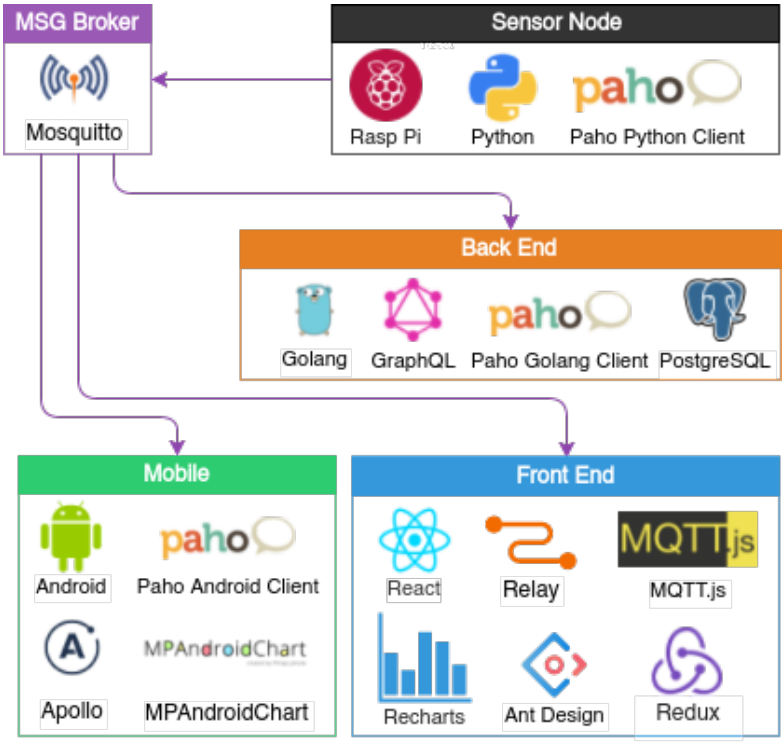
Karena teknologi dasar yang dipilih adalah React, maka perancangan juga menerapkan konsep yang dipakai oleh React, yaitu konsep berbasis komponen. Hasil pohon komponen berdasarkan desain antarmuka yang telah dibuat tertera pada Gambar 3.19.



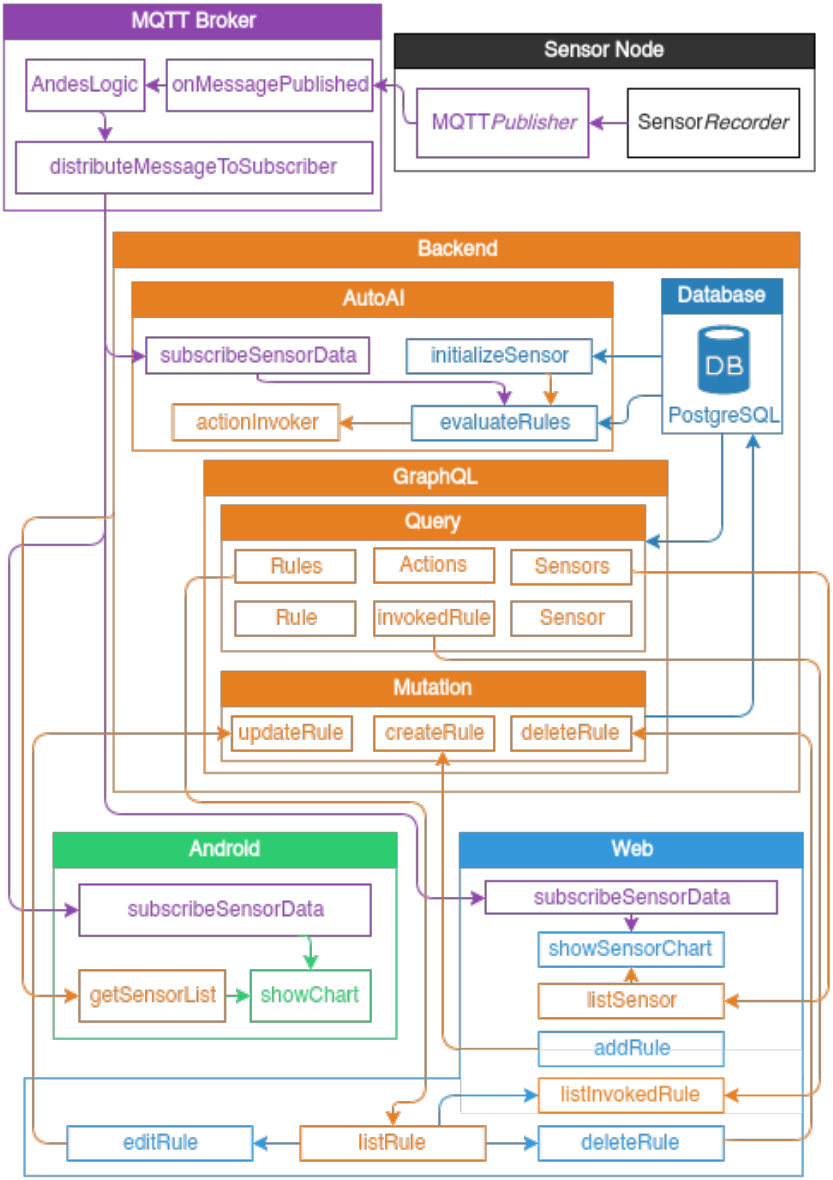
Gambar 3.19: Rancangan komponen pada aplikasi Web

3.4.7 Arsitektur Umum Sistem

Berdasarkan hasil analisis dan perancangan pada sub bab sebelumnya, diperoleh arsitektur umum sistem ini. Terdapat dua jenis arsitektur sistem, yaitu arsitektur teknologi sistem dan arsitektur perangkat lunak sistem. Visualisasi arsitektur teknologi sistem tertera pada Gambar 3.20 dan arsitektur perangkat lunak sistem tertera pada Gambar 3.21



Gambar 3.20: Arsitektur sistem beserta teknologi yang digunakan



Gambar 3.21: Arsitektur perangkat lunak sistem

BAB IV

IMPLEMENTASI

Pada bab ini dibahas mengenai implementasi pembangunan sistem. Pembahasan dilakukan di setiap komponen yang ada, yaitu: implementasi *node* sensor, implementasi *backend*, implementasi aplikasi Android dan implementasi aplikasi Web.

4.1 Lingkungan Implementasi

4.1.1 Perangkat Keras

Implementasi pembangunan sistem diterapkan pada perangkat keras dengan spesifikasi sebagai berikut:

1. Intel Core i5-4210U 1.70GHz
2. RAM DDR3 4 GB

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

1. Sistem Operasi Ubuntu 16.04 LTS 64 Bit
2. Python versi 2.7
3. Mosquitto versi 1.4.10 untuk MQTT *broker*
4. PostgreSQL versi 9.6 untuk sistem manajemen basis data
5. Golang versi 1.8 untuk implementasi *backend*
6. Android Studio versi 2.3 untuk pengembangan aplikasi Android
7. NodeJS versi 6.10.3 untuk pengembangan aplikasi Web
8. Yarn versi 0.23 untuk manajemen paket installasi untuk pengembangan aplikasi Web.

4.2 Implementasi *Message Broker*

Message broker yang digunakan pada sistem ini untuk menangani metode komunikasi *publish/subscribe* yaitu

mosquitto[1]. Diperlukan beberapa tahap untuk dapat menggunakan *mosquitto*, yaitu tahap pemasangan dan konfigurasi. Untuk melakukan pemasangan pada sistem operasi Ubuntu, jalankan Kode Sumber 4.1 pada terminal:

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

Kode Sumber 4.1: *Command* untuk installasi *mosquitto*

Lalu, agar *message broker* dapat diakses pada aplikasi Web, perlu dilakukan konfigurasi tambahan dengan menambahkan Kode Sumber 4.2 pada `/etc/mosquitto/mosquitto.conf`.

```
listener 9001
protocol Websockets
```

Kode Sumber 4.2: Konfigurasi tambahan *mosquitto*

Setelah itu, jalankan *mosquitto* menggunakan Kode Sumber 4.3.

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

Kode Sumber 4.3: *Command* untuk menjalankan MQTT *broker*

4.3 Implementasi *Node* Sensor

Implementasi *node* sensor diterapkan menggunakan Raspberry Pi 1 Model B dan PIR *motion sensor*. Pertama, lakukan installasi sistem operasi untuk Raspberry Pi dengan cara mengunduh *file* NOOBS[2]. Lalu, siapkan kartu SD dan *extract file* hasil unduhan tersebut ke dalam kartu SD. Pasang kartu SD tersebut pada Raspberry Pi, lalu ikuti langkah pemasangan sistem operasi hingga Raspberry Pi siap digunakan. Setelah berhasil melakukan pemasangan sistem operasi, hubungkan PIR *motion sensor* dengan Raspberry Pi dengan menggunakan *jumper*. Pada sensor PIR terdapat 3 pin, yaitu VCC, OUT dan

[1]Mosquitto dapat diunduh di https://mosquitto.org/download/
[2]NOOBS dapat diunduh di https://www.raspberrypi.org/downloads/noobs/

Gambar 4.1: Implementasi Raspberry Pi dengan sensor PIR

GND. Hubungkan VCC ke 5V, OUT ke GPIO pin 4, dan GND ke GND pada pin di perangkat Raspberry Pi menggunakan *female to female jumper*. Sehingga konfigurasi alat menjadi seperti pada Gambar 4.1.

Pada bagian perangkat lunak, *node* sensor perlu melakukan beberapa hal untuk dapat mengirimkan data rekaman sensor ke sistem, yaitu merekam data lingkungan dan mempublikasi data yang direkam kepada MQTT *broker*. Agar dapat mengakses data rekaman sensor, dibutuhkan *library gpiozero.MotionSensor*. Lalu, konfigurasikan kode pin sehingga sesuai dengan konfigurasi perangkat keras, yaitu pada pin 4. Setelah itu, lakukan perulangan tak terbatas sehingga program dapat merekam data pergerakan secara terus menerus. Kode untuk merekam data lingkungan menggunakan sensor PIR(*Passive Infrared Sensor*) tertera pada Kode Sumber 4.4.

```
from gpiozero import MotionSensor

pir = MotionSensor(4)
while True:
  pir.wait_for_motion()
  publish(1)
  pir.wait_for_no_motion()
  publish(0)
```

Kode Sumber 4.4: Merekam data pergerakan menggunakan sensor PIR

Setelah mendapatkan data hasil rekaman sensor, *node* sensor mempublikasikan data tersebut kepada MQTT *broker* menggunakan *library eclipse paho python client*. Data yang dikirim yaitu ID, data sensor dan waktu perekaman. Data tersebut perlu diubah menjadi format JSON agar dapat dipublikasikan. Data sensor dipublikasikan pada topik ”building/sensor/*id*”. Potongan kode untuk mempublikasikan data rekaman sensor tertera pada Kode Sumber 4.5.

```
client = mqtt.Client()
client.connect(brokerAddress, 1883, 60)
sensorTopic = "building/sensor/" + sensorID

def publish(val):
  now = int(time.time())
  sensorData = {'sensorID':sensorID, 'val':val, 'time':now}
  sensorDataJSON = json.dumps(sensorData)
  client.publish(sensor_topic, sensorDataJSON)
  print("Publish", sensorData, "to", sensor_topic)
```

Kode Sumber 4.5: Publikasi data rekaman sensor

4.4 Implementasi *Backend*

Pada bab ini dibahas mengenai implementasi sistem pada komponen *backend*. Berdasarkan analisis dan perancangan sistem pada bab 3.4.4, terdapat tiga sub komponen pada komponen *backend*, yaitu komponen basis data, komponen *graphql*, dan komponen AutoAI.

4.4.1 Implementasi Basis Data

Berdasarkan hasil perancangan basis data pada bab 3.4.4.1, terdapat lima entitas yang diimplementasikan menjadi suatu tabel pada basis data *postgresql*. Detail implementasi tiap entitas adalah sebagai berikut:

1. Tabel sensor

```
CREATE TABLE sensor (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  status BOOL,
  updatedAt TIMESTAMPTZ DEFAULT now(),
  isDeleted BOOL DEFAULT false
);
```

Kode Sumber 4.6: *Query* untuk membuat tabel sensor

2. Tabel data sensor

```
CREATE TABLE sensorData (
  id SERIAL PRIMARY KEY,
  sensorID INT,
  val TEXT,
  time INT
);
```

Kode Sumber 4.7: *Query* untuk membuat tabel data sensor

3. Tabel *action*

```
CREATE TABLE action (
  id SERIAL PRIMARY KEY,
  name VARCHAR(20),
  callbackFn TEXT,
  updatedAt TIMESTAMPTZ DEFAULT now(),
  isDeleted BOOL DEFAULT false
);
```

Kode Sumber 4.8: *Query* untuk membuat tabel *action*

4. Tabel *rule*

```
CREATE TABLE rule (
  id SERIAL PRIMARY KEY,
  actionID INT,
  name VARCHAR(50),
  index INT,
  status BOOL,
  updatedAt TIMESTAMPTZ DEFAULT now(),
  isDeleted BOOL DEFAULT false
);
```

Kode Sumber 4.9: *Query* untuk membuat tabel *rule*

5. Tabel *invoked rule*

```
CREATE TABLE invokedRule (
  id SERIAL PRIMARY KEY,
  ruleID INT,
  data TEXT,
  updatedAt TIMESTAMPTZ DEFAULT now(),
  isDeleted BOOL DEFAULT false
);
```

Kode Sumber 4.10: *Query* untuk membuat tabel *invoked rule*

Pada bahasa Go, program tidak dapat langsung membaca obyek yang disimpan pada basis data, sehingga dibutuhkan entitas/domain untuk merepresentasikan obyek tersebut. Oleh karena itu, dibuatlah lapisan domain yang merepresentasikan obyek yang disimpan pada basis data. Implementasi tertera pada Kode Sumber 4.11.

```
  type Rule struct {
    ID int
    Name string
    Index int
    Status bool
    Rule string
    Action Action
  }
```

Kode Sumber 4.11: Representasi obyek *rule* menggunakan bahasa Go

Lalu, agar obyek pada basis data dapat diakses oleh sub komponen *backend* lainnya, sistem perlu menyediakan lapisan antarmuka penghubung yang berfungsi untuk mengambil data pada basis data dan menyimpannya ke dalam domain. Untuk mempermudah penyusunan *query*, penulis menggunakan *package squirrel*[3]. Lalu, *package* yang digunakan untuk mengakses *postgresql* yaitu *pgx*[4]. Beberapa fungsi yang berhubungan dengan basis data yang dibutuhkan oleh sistem yaitu fungsi untuk mengambil data, menambah data, mengubah dan menghapus data. Untuk mengambil data dari basis data, gunakan *squirrel* dengan mendefinisikan *field* yang ingin diambil, lalu cari data yang belum dihapus dan urutkan berdasarkan waktu terakhir diubah. Lalu, setelah *query* disusun, jalankan *query* tersebut menggunakan *package pgx*. Setelah itu, simpan obyek yang telah diambil ke dalam domain yang telah dibuat. Implementasi tertera pada Kode Sumber 4.12.

```
1  func Rules() ([]*domain.Rules) {
2    query, params, err := squirrel.
3      Select("id, name, index, status, rule, actionID").
4      From("rule").
5      Where("isDeleted=?", false).
6      OrderBy("updatedAt DESC").
7      ToSql()
8
9    rows, err := pgx.Query(query, params...)
10   defer rows.close()
11
12   var rules []*domain.Rule
13   for rows.Next() {
14     var r domain.Rule
15     err = rows.
16       Scan(&r.ID, &r.Name, &r.Index, &r.Status, &r.Rule, &r.
             actionID)
17     rules = append(rules, &r)
18   }
19
```

[3]Squirrel dapat diakses di github.com/Masterminds/squirrel
[4]Pgx dapat diakses di github.com/jackc/pgx

```
20    return sensors
21  }
```

Kode Sumber 4.12: Mengambil data *rule* dari basis data

Lalu, untuk melakukan fungsi tambah, terdapat satu *parameter* pada fungsi yang merepresentasikan domain yang akan dimasukkan ke dalam basis data. Domain tersebut diubah menjadi bentuk *map*, lalu diproses dengan *package squirrel*. Setelah *query* berhasil disusun, eksekusi *query* tersebut menggunakan *package pgx*. Implementasinya tertera pada Kode Sumber 4.13.

```
1   func CreateRule(newRule *domain.Rule) {
2     newRule := squirrel.Eq{
3       "name": newRule.Name,
4       "index": newRule.Index,
5       "status": newRule.Status,
6       "rule": newRule.Rule,
7       "actionID": newRule.Action.ID,
8     }
9
10    query, params, _ := squirrel.Insert("rule").
11      SetMap(newRuleMap).
12      ToSql()
13
14    _, err := pgx.Exec(query, params...)
15  }
```

Kode Sumber 4.13: Menambah *rule* ke basis data

Pada fungsi ubah data, terdapat dua *parameter*, yaitu ID yang berfungsi untuk mengetahui data mana yang ingin diubah, dan domain data yang berisi data baru yang akan dimasukkan ke dalam sistem. Setelah itu, ubah data domain ke bentuk *map* dan cari data pada basis data yang memiliki ID yang sama dengan ID yang dimasukkan. Lalu, eksekusi *query* menggunakan *pgx*. Implementasinya tertera pada Kode Sumber 4.14.

```
func UpdateRule(ID int, updatedRule *domain.Rule) {
  updatedRule := squirrel.Eq{
    "name": updatedRule.Name,
    "index": updatedRule.Index,
    "status": updatedRule.Status,
    "rule": updatedRule.Rule,
    "actionID": updatedRule.Action.ID,
  }

  query, params, _ := squirrel.Update("rule").
    SetMap(updatedRuleMap).
    Where("id=?", ID).
    ToSql()

  _, err := pgx.Exec(query, params...)
}
```

Kode Sumber 4.14: Mengubah *rule* pada basis data

Pada fungsi hapus data hanya terdapat satu *parameter*, yaitu ID yang berfungsi untuk mengetahui data mana yang ingin dihapus. Agar data yang dihapus tetap berada pada basis data, maka yang dilakukan pada fungsi hapus hanya mengubah *field isDeleted* menjadi *true*. Implementasinya tertera pada Kode Sumber 4.15.

```
func DeleteRule(ID int) {
  query, params, _ := squirrel.Update("rule").
    SetMap(sq.Eq{"isDeleted":true}).
    Where("id=?", ID).
    ToSql()

  _, err := pgx.Exec(query, params...)
}
```

Kode Sumber 4.15: Menghapus *rule* pada basis data

4.4.2 Implementasi *GraphQL*

Untuk dapat mengimplementasikan *graphql* menggunakan bahasa Go, dibutuhkan *package* tambahan yang berfungsi untuk menyediakan fitur-fitur *graphql*. *Package* yang dipilih oleh

penulis yaitu *graphql-go*[5]. *Graphql-go* membutuhkan skema yang merepresentasikan data yang akan disediakan oleh sistem. Skema tersebut dibuat pada Kode Sumber 3.1. Lalu *package* tersebut menggunakan tipe data *struct* untuk merepresentasikan suatu tipe obyek pada skema. *Struct* tersebut memiliki daftar fungsi dengan nama yang sama dengan *field* yang ada pada tipe obyek tersebut. Kode untuk merepresentasikan tipe obyek *rule* tertera pada Kode Sumber 4.16.

```
1  type ruleResolver struct {
2    s *domain.Rule
3  }
4
5  func (s *ruleResolver) ID() graphql.ID {
6    return relay.MarshalID("rule", s.s.ID)
7  }
8
9  func (s *ruleResolver) Name() string {
10   return s.s.Name
11 }
12
13 func (s *ruleResolver) Index() int32 {
14   return int32(s.s.Index)
15 }
16
17 func (s *ruleResolver) Status() *bool {
18   return &s.s.Status
19 }
20
21 func (s *ruleResolver) Rule() string {
22   return s.s.Rule
23 }
24
25 func (s *ruleResolver) ActionID() int32 {
26   return int32(s.s.Action.ID)
27 }
```

Kode Sumber 4.16: Tipe obyek *rule* menggunakan graphql-go

Package graphql-go menyediakan *struct* `Resolver` yang berfungsi untuk memproses tipe obyek *query*. Untuk dapat menyediakan data menggunakan *graphql*, program perlu

[5] graphql-go dapat diakses di github.com/neelance/graphql-go

mengakses basis data, lalu mengembalikan hasil pemrosesannya kepada `Resolver` tersebut. Kode untuk menyediakan daftar *rules* tertera pada Kode Sumber 4.17.

```
1  func (r *Resolver) Rules() *[]*ruleResolver {
2    var rules []*ruleResolver
3    rulesData := postgres.Rules()
4    for i := range rulesData {
5      rules = append(rules, &ruleResolver{rulesData[i]})
6    }
7
8    return &rules
9  }
```

Kode Sumber 4.17: Menyediakan daftar rules menggunakan *package* graphql-go

Ketika terdapat *parameter* pada *query* tersebut, maka tambahkan juga *parameter* pada fungsi untuk menyediakan data dengan tipe *struct* yang berisi *parameter* tersebut. Kode untuk mencari *rule* yang spesifik berdasarkan ID tertera pada Kode Sumber 4.18.

```
1  func (r *Resolver) Rule(args *struct{ ID graphql.ID }) *
       ruleResolver {
2    s := postgres.Rule(unmarshalID(args.ID))
3    return &ruleResolver{s}
4  }
```

Kode Sumber 4.18: Mencari *rule* spesifik berdasarkan ID

Untuk melakukan *mutation* atau memodifikasi data menggunakan *graphql* juga serupa, buat fungsi yang bernama sama dengan nama *mutation*, lalu tambahkan *parameter* sesuai dengan isi *parameter* pada skema *graphql*. Kode untuk menambah *rule* tertera pada Kode Sumber 4.19.

```
1  func (r *Resolver) CreateRule(args *struct { Rule *ruleInput }) *
       ruleResolver {
2    ruleData := ruleInputToDomain(args.Rule)
3    s := postgres.CreateRule(ruleData)
4    return &ruleResolver{s}
5  }
```

Kode Sumber 4.19: Menambah *rule* pada *graphql server*

4.4.3 Implementasi *Auto Action Invoker*

Berdasarkan perancangan pada subbab 3.4.4.3, terdapat beberapa langkah untuk mengimplementasikan AutoAI. Langkah pertama yaitu melakukan inisialisasi variabel sensorData. Karena data seluruh sensor tidak dikirimkan dalam waktu yang bersamaan, sehingga butuh variabel untuk menyimpan seluruh data rekaman sensor. Selain itu, dibutuhkan variabel mutex agar tidak terjadi *race condition* ketika data rekaman sensor datang secara bersamaan. Kode untuk melakukan inisialisasi variabel sensorData tertera pada Kode Sumber 4.20.

```
var mx = &sync.RWMutex{}
var sensorData map[string]interface{}

func prepareSensor() {
  sensors := postgres.Sensors()
  sensorData = make(map[string]interface{})
  for i := range sensors {
    sensorID := "s" + strconv.Itoa(sensors[i].ID)
    sensorData[sensorID] = false
  }
}
```

Kode Sumber 4.20: Inisialisasi variabel sensorData pada AutoAI

Setelah itu, agar AutoAI dapat menerima data rekaman sensor secara *real-time*, dibutuhkan *package* tambahan. *Package* yang digunakan yaitu *paho.mqtt.golang*. Untuk menggunakan *package* tersebut dibutuhkan konfigurasi sehingga dapat berkomunikasi dengan MQTT *broker*. Konfigurasinya tertera pada Kode Sumber 4.21.

```
broker := flag.String("broker", brokerAddress, "The broker URI.")
password := flag.String("password", "", "password (opsional)")
user := flag.String("user", "", "User (opsional)")
id := flag.String("id", "AutoAI-Backend", "ID client")
flag.Parse()

connOpts = &MQTT.ClientOptions{
    ClientID: *id,
    Username: *user,
```

```
10      Password: *password,
11      MaxReconnectInterval: 1 * time.Second,
12      KeepAlive: 0,
13  }
14
15  connOpts.AddBroker(*broker)
```

Kode Sumber 4.21: Konfigurasi Paho MQTT Golang

Lalu, setelah berhasil melakukan konfigurasi, buat *client* baru untuk berlangganan pada topik dimana data rekaman sensor dipublikasikan, yaitu ”building/sensor/#”. Selain berlangganan, siapkan juga fungsi untuk memproses data ketika data rekaman sensor diterima. Implementasi tertera pada Kode Sumber 4.22.

```
1   client := MQTT.NewClient(connOpts)
2   if token := client.Connect(); token.Wait() && token.Error() != nil
        {
3       return token.Error()
4   }
5   subscribeSensorData := func(client MQTT.Client, message MQTT.
        Message) {
6       mx.Lock()
7     defer mx.Unlock()
8     receivedSensorData := domain.SensorData{}
9     json.Unmarshal(message.Payload(), &receivedSensorData)
10    sensorData["s" + receivedSensorData.SensorID] =
          receivedSensorData.Val
11    go evaluateRules(receivedSensorData.SensorID)
12    go postgres.RecordSensorData(&receivedSensorData)
13
14  }
15
16  if token := client.Subscribe(topic, byte(qos), prosesDataSensor);
        token.Wait() && token.Error() != nil {
17      return token.Error()
18  }
```

Kode Sumber 4.22: *Subscribe* topik dan memproses data rekaman

Ketika menerima data rekaman sensor, sistem mencari daftar *rule* pada basis data yang memiliki *rule expression* yang menggunakan data sensor tersebut sebagai kriterianya. Pada tiap *rule* tersebut, lakukan validasi *rule expression* menggunakan

package conditions berdasarkan nilai sensor yang tersimpan di variabel sensorData. Jika kondisi terpenuhi, simpan sensorData tersebut ke dalam *invoked rule* dan lakukan aksi berdasarkan konfigurasi *rule*. Implementasi tertera pada Kode Sumber 4.23.

```
1  func evaluateRules(sensorID string) {
2    rules := postgres.RulesBySensor(sensorID)
3    for i := range rules {
4      go actionInvoker(rules[i])
5    }
6  }
7
8  func actionInvoker(rule *domain.Rule) {
9    p := conditions.NewParser(strings.NewReader (rule.Rule))
10   expr, err := p.Parse()
11
12   mx.RLock()
13   r, err := conditions.Evaluate(expr, sensorData)
14   savedData := sensorData
15   mx.RUnlock()
16   if r == true {
17     jsonRes, _ := json.Marshal(savedData)
18     invokedRule := domain.InvokedRule{
19       Rule: *rule,
20       Data: string(jsonRes),
21     }
22     go actions.Invoke(&rule.Action, &invokedRule)
23     go postgres.CreateInvokedRule(&invokedRule)
24   }
25 }
```

Kode Sumber 4.23: Validasi *rule expression* dan melakukan *action* otomatis

4.5 Implementasi Aplikasi pada *Platform* Android

4.5.1 Pemasangan *library* untuk Aplikasi Android

Berdasarkan subbab 3.4.5.1, terdapat tiga *library* yang dibutuhkan untuk membangun aplikasi Android, yaitu *apollo android client*[6], *android paho client*, dan *mpandroidchart*. Agar

[6]Informasi mengenai *apollo android* client dapat dilihat di github.com/apollographql/apollo-android

dapat menggunakan fungsi *library* tersebut, tambahkan Kode Sumber 4.24 *file* `build.gradle` tingkat *module* untuk memasang ketiga *library* tersebut.

```
compile 'com.github.PhilJay:MPAndroidChart:v3.0.2'

compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
compile 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'

compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.jakewharton.retrofit:retrofit2-rxjava2-adapter:1.0.0'
compile 'com.apollographql.android:converter-pojo:0.1.0'
compile 'io.reactivex.rxjava2:rxandroid:2.0.1'
```

Kode Sumber 4.24: Konfigurasi file build.gradle untuk pemasangan *library*

Setelah itu, tambahkan Kode Sumber 4.25 pada *file* `build.gradle` tingkat *project* di bagian *dependency* untuk memasang *apollo android client*.

```
. . .
dependencies {
  . . .
  classpath 'com.apollographql.apollo:gradle-plugin:0.3.0'
}
. . .
```

Kode Sumber 4.25: Konfigurasi file `build.gradle` untuk pemasangan *library*

Tambahkan juga Kode Sumber 4.26 di bagian *repositories* agar dapat menggunakan MQTT *client* dan *mpandroidchart*.

```
. . .
repositories {
  maven { url "https://jitpack.io" }
  maven { url "https://repo.eclipse.org/content/repositories/paho
      -releases/" }
}
. . .
```

Kode Sumber 4.26: Konfigurasi file build.gradle untuk pemasangan *library*

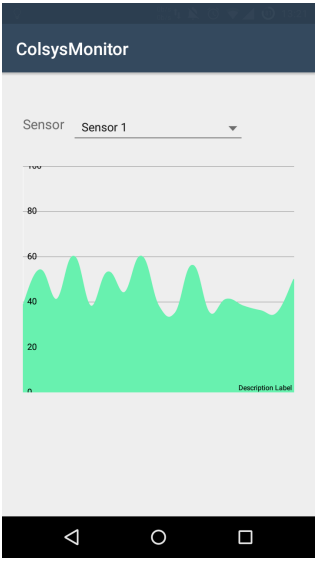
4.5.2 Implementasi Antarmuka Aplikasi Android

Berdasarkan bab sebelumnya, dibutuhkan dua obyek tampilan pada aplikasi Android, yaitu *spinner* dan *chart* menggunakan *mpandroidchart*. Tambahkan Kode Sumber 4.27 pada file `activity_main.xml` untuk menambahkan dua obyek tersebut.

```
<Spinner
 android:id="@+id/sensorID"
 style="@style/Base.Widget.AppCompat.Spinner.Underlined"
 android:layout_height="wrap_content"
 android:layout_marginEnd="8dp"
 android:layout_marginLeft="8dp"
 android:layout_marginRight="8dp"
 android:layout_marginStart="8dp"
 app:layout_constraintHorizontal_bias="0.0"
 app:layout_constraintRight_toRightOf="parent"
 android:layout_marginTop="-16dp" />

<com.github.mikephil.charting.charts.LineChart
 android:id="@+id/sensorChart"
 android:layout_height="261dp"
 android:layout_marginEnd="8dp"
 android:layout_marginLeft="8dp"
 android:layout_marginRight="8dp"
 android:layout_marginStart="8dp"
 android:layout_marginTop="27dp"
 app:layout_constraintHorizontal_bias="0.0"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintRight_toRightOf="parent"
 app:layout_constraintTop_toBottomOf="@+id/sensorID" />
```

Kode Sumber 4.27: Implementasi antarmuka pada Android

Hasil implementasi antarmuka aplikasi Android tertera pada Gambar 4.2



Gambar 4.2: Implementasi antarmuka Android

4.5.3 Implementasi Alur Kerja Aplikasi Android

Berdasarkan bab 3.4.5.3, terdapat tiga alur kerja pada aplikasi Android. Subbab ini akan menjelaskan implementasi tiap-tiap alurnya.

4.5.3.1 Implementasi Mengambil Daftar Sensor dari *Graphql* Server

Langkah pertama untuk dapat mengambil daftar sensor yaitu inisialisasi variabel *ApolloClient* beserta alamat *graphql server* untuk mengambil data dari *server* dan *sensorList* untuk menyimpan data hasil *request* dari *graphql server*. Implementasi tertera pada Kode Sumber 4.28.

```
private void getSensorList() {
  OkHttpClient client = new OkHttpClient.Builder().build();
  ApolloClient apolloClient = ApolloClient.builder()
    .serverUrl("http://10.151.32.111:8080/graphql")
    .okHttpClient(client)
    .build();
  List<Sensor> sensorList = new ArrayList<>();
  . . .
```

Kode Sumber 4.28: Mengambil daftar sensor dari *graphql server*

Lalu, kirim HTTP *request* menggunakan *apollo*, jika tidak berhasil, lakukan lagi fungsi untuk mengambil daftar sensor. Jika berhasil, simpan daftar sensor pada variabel *sensorList*.

```
apolloClient.newCall(new SensorQuery()).enqueue(
  new ApolloCall.Callback<SensorQuery.Data>() {
  @Override
  public void onResponse(@Nonnull Response<SensorQuery.Data>
       response) {
    int index = 0;
    for (SensorQuery.Data.Sensor sensorData : response.data().
         sensors()) {
      Sensor newSensor = new Sensor(sensorData);
      sensorList.add(newSensor);
      sensorIndex.put(decodeID(newSensor.getId()), index++);
    }
    Sensor sensorHint = new Sensor("-- Pilih Sensor --");
    sensorList.add(sensorHint);
    MainActivity.this.runOnUiThread(new Runnable() {
      @Override
      public void run() {
        initSpinner();
        addSensorsOnSpinner(sensorList);
      }
    });
  }

  @Override
  public void onFailure(@Nonnull ApolloException e) {
    System.out.println(e.getMessage());
    getSensorList();
  }
});
```

Kode Sumber 4.29: Mengirim HTTP *request* dan menyimpannya pada variabel *sensorList*

Lalu, setelah menerima daftar sensor, tempatkan daftar sensor pada *spinner*. Implementasi tertera pada Kode Sumber 4.30.

```
1  public void addSensorsOnSpinner (List<Sensor> sensors) {
2    ArrayAdapter<Sensor> dataAdapter = new ArrayAdapter<Sensor>(this,
3      android.R.layout.simple_spinner_item, sensors)
4    sSpinner.setAdapter(dataAdapter);
5  }
```

Kode Sumber 4.30: Menempatkan daftar sensor ke *spinner*

4.5.3.2 Implementasi Menerima Data Hasil Rekaman Sensor

Langkah pertama yang dilakukan yaitu inisialisasi MQTT *client* menggunakan *paho android client*. Lalu konfigurasikan agar MQTT *client* dapat terhubung ke MQTT *broker*. Setelah itu, lakukan koneksi ke *server*. Lalu ketika berhasil, lakukan langganan pada topik ”bulding/sensor/#” sehingga dapat menerima seluruh data rekaman sensor. Implementasi tertera pada Kode Sumber 4.31.

```
1  private void initMQTT() {
2    MqttAndroidClient mqttClient = new MqttAndroidClient(
3      getApplicationContext(), brokerAddress, "Colsys-Android");
4    MqttConnectOptions mqttOpts = new MqttConnectOptions();
5    mqttClient.connect(mqttOpts, null, new IMqttActionListener() {
6      @Override
7      public void onSuccess(IMqttToken asyncActionToken) {
8        int qos = 0
9        mqttClient.subscribe("building/sensor/\#", qos, null)
10     }
11   }
12   . . .
```

Kode Sumber 4.31: Konfigurasi MQTT *client* agar terhubung ke MQTT *broker*

Lalu, setiap kali menerima data rekaman sensor, simpan data tersebut pada variabel *sensorData* berdasarkan sumber sensornya. Karena data rekaman berbentuk JSON, maka perlu

diubah menjadi *JSONObject*, lalu ambil data rekaman sensor beserta waktu perekamannya. Setelah itu, simpan ke dalam sensorData berdasarkan sumber sensornya. Implementasi tertera pada Kode Sumber 4.32.

```
  . . .
@Override
public void messageArrived(String topic, MqttMessage message)
    throws Exception {
  String messageStr = new String(message.getPayload());
  JSONObject sensorData = new JSONObject(messageStr);
  String sensorID = sensorData.getString("sensorID");
  int sensorVal = sensorData.getInt("val");
  long timestamp = sensorData.getInt("time");
  Sensor selectedSensor = sensorList.get(getSensorIndex(sensorID));
  selectedSensor.addData(sensorVal, timestamp);
}
```

Kode Sumber 4.32: Menyimpan data rekaman sensor

4.5.3.3 Implementasi Menampilkan Grafik Berdasarkan Pilihan Pengguna

Agar dapat menampilkan grafik berdasarkan sensor pilihan pengguna. Pengguna memilih sensor yang tersedia pada *spinner*. Aplikasi akan menerima *index* sensor pilihan pengguna. *Index* tersebut akan digunakan untuk mencari sensor pada variabel *sensorList*. Implementasi tertera pada Kode Sumber 4.33.

```
sSpinner = (Spinner) findViewById(R.id.sensorID);
sSpinner.setOnItemSelectedListener(
  new AdapterView.OnItemSelectedListener() {
  @Override
  public void onItemSelected(
    AdapterView<?> parent, View view, int index, long id
  ) {
    if (index >= 0 && index < sensorList.size() - 1) {
      Sensor selectedSensor = sensorList.get(index);
    }
  }
    . . .
```

Kode Sumber 4.33: Menerima sensor pilihan pengguna

Lalu, setelah mendapatkan sensor yang dipilih, hapus seluruh data rekaman sensor yang berada pada grafik, lalu tambahkan data rekaman sensor yang dipilih ke grafik. Pada saat ini, data yang terdapat pada grafik merupakan data rekaman yang dipilih oleh pengguna. Implementasi tertera pada Kode Sumber 4.34.

```
1     . . .
2  sensorChart = (LineChart) findViewById(R.id.sensorChart);
3  sensorChart.clear()
4  ArrayList<Entry> chartData = new ArrayList<>();
5  for (SensorData data : selectedSensor.getData()) {
6    chartData.add(new Entry(data.getTime(), data.getValue());
7  }
8  chartDataSet = new LineDataSet(chartData, selectedSensor.getName())
       ;
9  LineData lineData = new LineData(chartDataSet);
10 sensorChart.setData(lineData);
11    . . .
```

Kode Sumber 4.34: Menampilkan data rekaman sensor pada grafik

4.6 Implementasi Aplikasi pada *Platform* Web

Berdasarkan pada bab 3.4.6, terdapat beberapa hal yang diimplementasikan pada aplikasi Web, yaitu implementasi pemasangan kebutuhan teknologi, implementasi antarmuka, dan implementasi fungsionalitas Web. Pada subbab selanjutnya akan menjelaskan mengenai ketiga implementasi tersebut.

4.6.1 Implementasi Pemasangan Kebutuhan Teknologi

Teknologi yang dibutuhkan pada aplikasi Web yaitu *react, ant design, relay,* MQTT.js dan *recharts*. Agar dapat memasang keseluruhan teknologi tersebut, *yarn* menyediakan fitur untuk melakukan installasi teknologi yang dibutuhkan dengan cara membuat *file* bernama `package.json` yang berisi daftar kode teknologi yang dibutuhkan. Isi `package.json` tertera pada Kode Sumber 4.35.

```
{
  "name": "colsys-web",
  "version": "0.1.0",
  "devDependencies": {
    "react-scripts": "1.0.6",
    "babel-plugin-relay": "^1.0.1"
  },
  "dependencies": {
    "react": "^15.5.4",
    "react-dom": "^15.5.4",
    "react-redux": "^5.0.5",
    "antd": "^2.10.1",
    "react-relay": "^1.0.0",
    "mqtt": "^2.7.2",
    "recharts": "^0.22.4",
    "redux": "^3.6.0"
  }
  . . .
}
```

Kode Sumber 4.35: File package.json untuk pembuatan aplikasi Web

Setelah itu, lakukan installasi teknologi dengan menjalankan *command yarn install*. *Yarn* akan secara otomatis melakukan installasi teknologi.

4.6.2 Implementasi Fungsionalitas Web

Subbab ini akan membahas mengenai implementasi fungsionalitas yang telah dirancang pada bab 3.4.6.1

4.6.2.1 Implementasi *Dashboard*

Pada *dashboard* dibutuhkan data daftar *rule*, daftar sensor, dan sensor yang dipilih untuk ditampilkan data rekamannya secara *real-time*. Oleh karena itu, sistem perlu mengambil daftar *rule* dan sensor dari *graphql server*. Selain itu, sistem juga perlu menyimpan *state* sensor yang dipilih oleh pengguna agar dapat menampilkan data rekaman sensor secara *real-time* sesuai dengan sensor yang dipilih pengguna.

Daftar *rule* dan sensor dapat diambil menggunakan *relay* dengan cara mendeklarasikan komponen *queryrenderer* yang disediakan oleh *relay*. Lalu, tuliskan spesifikasi data yang dibutuhkan oleh *dashboard* pada properti *query*. Agar tidak terjadi *overfetching*, detail kebutuhan tiap daftar sensor dan *rule* akan didefinisikan pada komponen yang menampilkan data tersebut. Implementasi tertera pada Kode Sumber 4.36.

```
import {
  QueryRenderer,
  graphql,
} from 'react-relay';
  . . .
<QueryRenderer
  environment={environment}
  query={graphql`
    query DashboardQuery {
        sensors {
            ...SensorCard_sensors
        }
        rules {
            ...RuleCard_rules
        }
    }
  `}
  . . .
```

Kode Sumber 4.36: Mengambil daftar *rule* dan sensor dari *graphql server*

State sensor pilihan disimpan menggunakan *redux*. Untuk dapat menyimpan dan memodifikasinya, siapkan tempat penyimpanan(*store*) *state*, lalu buatlah *reducers* untuk memilih sensor untuk ditampilkan. Implementasi tertera pada Kode Sumber 4.37.

```
const initialState = {
  selectedSensor: null,
}
const colsys = (state = initialState, action) => {
  switch(action.type) {
    case 'SELECT_SENSOR':
      let showSensor;
      if (state.selectedSensor === action.sensor) {
```

```
            showSensor = initialState.selectedSensor
        } else {
            showSensor = action.sensor
        }
        return {
            ...state,
            selectedSensor: showSensor,
        }
      default:
        return state;
  . . .
```

Kode Sumber 4.37: *Reducer* untuk memilih sensor yang ditampilkan pada grafik

Ketika aksi *redux* 'SELECT_SENSOR' dilakukan, *state* sensor pilihan akan berubah menjadi sensor yang dipilih oleh pengguna. Jika sensor pilihan dipilih lagi, maka *dashboard* akan menyembunyikan grafik sensornya.

4.6.2.2 Implementasi Daftar Sensor

Daftar sensor membutuhkan data id sensor dan nama sensor untuk ditampilkan pada antarmuka daftar sensor. Gunakan *relay* Untuk dapat mendapatkan data tersebut dengan cara memanggil fungsi *createfragmentcontainer*. Implementasi tertera pada Kode Sumber 4.38.

```
imoprt { createFragmentContainer, graphql } from 'react-relay';
 . . .
export default createFragmentContainer(
  SensorCard,
  graphql`
    fragment SensorCard_sensors on Sensor @relay(plural: true) {
      id, name,
  } `,
);
 . . .
```

Kode Sumber 4.38: *Request* data daftar sensor menggunakan *relay*

Ketika salah satu sensor diklik, maka akan muncul grafik data rekaman sensor secara *real-time*. Sehingga butuh aksi *redux*

untuk mengubah *state* sensor yang dipilih berdasarkan pilihan pengguna. Gunakan fungsi *dispatch* untuk melakukan aksi pada *redux*, lalu definisikan tipe aksinya dengan tipe yang sama pada saat mendeklarasikan *reducers*. Implementasi tertera pada Kode Sumber 4.39.

```
changeSensor: (sensor) => {
  dispatch({
    type: 'SELECT_SENSOR',
    sensor
  })
```

Kode Sumber 4.39: Mengubah *state* sensor pilihan

4.6.2.3 Implementasi Daftar *Rule*

Daftar sensor membutuhkan data id *rule*, nama *rule*, *rule expression* dan id *action* untuk ditampilkan pada antarmuka daftar *rule*. Gunakan *relay* Untuk dapat mendapatkan data tersebut dengan cara memanggil fungsi *createfragmentcontainer*. Implementasi tertera pada Kode Sumber 4.40.

```
export default createFragmentContainer(
  RuleCard,
  graphql`
    fragment RuleCard_rules on Rule @relay(plural: true) {
      id,  name, rule, actionID
  }`,
);
```

Kode Sumber 4.40: *Request* data daftar *rule* menggunakan *relay*

Selain dapat melihat daftar *rule*, pengguna juga dapat menambah *rule*, mengubah *rule* yang terdaftar pada sistem, serta menghapus *rule* yang terdaftar. Pengguna juga dapat menampilkan *rule* yang telah terpenuhi beserta waktunya yang disebut *invoked rule*. Maka dibutuhkan aksi *redux* untuk melakukan hal tersebut. Implementasi tertera pada Kode Sumber 4.41.

```
addRule: () => {
  dispatch({
    type: 'ADD_RULE',
  })
}
editRule: (rule) => {
  dispatch({
    type: 'EDIT_RULE',
    rule: rule
  })
},
deleteRule: (rule) => {
  removeRuleMutation.commit(rule.id)
},
showInvoked: (rule) =>  {
  dispatch({
    type: 'SELECT_RULE',
    rule
  })
}
```

Kode Sumber 4.41: Daftar aksi pada *rule*

4.6.2.4 Implementasi Tambah *Rule*

Agar dapat menambahkan data *rule* yang telah diisi oleh pengguna, aplikasi Web perlu melakukan *mutation* pada *graphql server*. *Relay* menyediakan API untuk melakukan *mutation* dengan memanggil fungsi *commitmutation*. *Commitmutation* dipanggil dengan konfigurasi *query graphql createRule* dengan variabel *rule* yang berasal dari form yang diisi oleh pengguna. Lalu, agar data yang telah dimasukkan berhasil ditampilkan, *query graphql* perlu meminta kembali *field* yang dibutuhkan untuk ditampilkan. Implementasi *commitmutation* tambah *rule* tertera pada Kode Sumber 4.42.

```
commitMutation(
environment,
{
  graphql`
  mutation addRuleMutation($input: RuleInput) {
```

```
6       createRule(rule: $input) {
7         id, name, rule, actionID
8       }
9     }`,
10    variables: {
11        input: { ...rule }
12    },
13    . . .
14  }
```

Kode Sumber 4.42: Implementasi *mutation* Tambah *Rule*

4.6.2.5 Implementasi Ubah *Rule*

Proses ubah *rule* juga dilakukan dengan cara memanggil fungsi *commitmutation*. *Commitmutation* dipanggil dengan konfigurasi *query graphql updateRule* dengan variabel ID yang berasal dari *rule* yang ingin diubah oleh pengguna dan data *rule* yang berasal dari form yang diisi oleh pengguna. Implementasi *commitmutation* untuk proses ubah *rule* tertera pada Kode Sumber 4.43.

```
1   commitMutation(
2   environment,
3   {
4     graphql`
5     mutation changeRuleMutation($input: RuleUpdateInput) {
6       updateRule(rule: $input) {
7         id, name, rule, actionID
8       }
9     }`,
10    variables: {
11        input: { id: ruleID, rule: rule }
12    },
13    . . .
```

Kode Sumber 4.43: Implementasi *mutation* Ubah *Rule*

4.6.2.6 Implementasi Hapus *Rule*

Proses hapus *rule* juga dilakukan dengan cara memanggil fungsi *commitmutation*. *Commitmutation* dipanggil dengan

konfigurasi *query graphql deleteRule* dengan variabel ID yang berasal dari *rule* yang ingin dihapus oleh pengguna. Implementasi *commitmutation* untuk proses hapus *rule* tertera pada Kode Sumber 4.44.

```
commitMutation(
environment,
{
  graphql`
  mutation removeRuleMutation($input: RuleDeleteInput) {
    deleteRule(input: $input) {
      id
    } }`,
  variables: {
      input: { id: ruleID }
  },
  . . .
```

Kode Sumber 4.44: Implementasi *mutation* Hapus *Rule*

4.6.2.7 Implementasi Tampil *InvokedRule*

Daftar *invoked rule* dapat diambil menggunakan *relay* dengan cara mendeklarasikan komponen *queryrenderer* yang disediakan oleh *relay*. Tuliskan spesifikasi data yang dibutuhkan *invoked rule* pada properti *query* dengan variabel ruleID. Variabel ruleID berasal dari *rule* yang dipilih pada daftar *rule*. Implementasi untuk menampilkan *invoked rule* tertera pada Kode Sumber 4.45.

```
import {
  QueryRenderer,
  graphql,
} from 'react-relay';
  . . .
<QueryRenderer
  environment={environment}
  query={graphql`
    query InvokedRuleCardQuery($id: ID!) {
      invokedRules(ruleid: $id) {
        id, rulename, data, time
      }
```

```
13    } `}
14    variables={{ id: rule.id }}
15    . . .
```

Kode Sumber 4.45: Menampilkan *invoked rule*

4.6.2.8 Implementasi *Real Time Chart*

Grafik data rekaman sensor secara *real-time* dibuat dengan menyiapkan *state* data rekaman sensor pilihan pada *store redux* untuk menyimpan data rekaman sensor yang akan ditampilkan pada grafik. Setelah itu, siapkan juga *reducer* untuk menambahkan data rekaman sensor pada *state* tersebut. Selain menambahkan data, jika data melebihi 10 rekaman, data tersebut dipotong sehingga hanya menampilkan 10 data rekaman sensor terakhir. Implementasi tertera pada Kode Sumber 4.46.

```
1  const initialState = {
2    animation: true, data: []
3  }
4    . . .
5  const chart = (state = initialState, action) => {
6    switch(action.type) {
7      case 'ADD_SENSOR_DATA':
8        const newSensorData = generateChartData(action.sensorData)
9        return {
10         ...state,
11         data: [ ...state.data.slice(-10), newSensorData ]
12       };
13   . . .
```

Kode Sumber 4.46: *Store redux* untuk *real-time* chart

Setelah menyiapkan *store redux* dan *reducer* untuk mengubah data rekaman sensor yang akan ditampilkan, langkah selanjutnya yaitu menerima data rekaman sensor secara *real-time*. Hal tersebut dapat dipenuhi dengan melakukan *subscribe* ke topik sensor yang dipilih oleh pengguna. Sebelum dapat melakukan *subscription*, aplikasi Web perlu terhubung pada MQTT *broker*. Implementasi untuk menghubungkan

aplikasi Web dengan MQTT *broker* tertera pada Kode Sumber 4.47.

```
1  import { connect } from 'mqtt';
2  mqtt = connect(brokerAddress, { clientId: 'colsysWeb-' + Date.now
       ()});
```

Kode Sumber 4.47: Konfigurasi MQTT pada aplikasi Web

Setelah berhasil terhubung pada MQTT *broker*, aplikasi Web dapat melakukan *subscription* pada topik sensor yang dipilih oleh pengguna. Topik tempat dipublikasikannya data rekaman sensor yaitu ”building/sensor/id”. Selain itu, ketika sebelumnya sudah melakukan *subscribe* pada sensor berbeda, lakukan *unsubscribe* pada sensor tersebut. Implementasi untuk melakukan *unsubscribe* dan *subscribe* topik tertera pada Kode Sumber 4.48.

```
1  . . .
2  const topicName = "building/sensor/"
3  mqtt.unsubscribe(topicname + selectedSensor.id)
4  mqtt.subscribe(topicname + sensor.id)
5  . . .
```

Kode Sumber 4.48: *Unsubscribe* dan *Subscribe* data rekaman sensor

Ketika menerima data rekaman sensor hasil berlangganan topik, data tersebut dimasukan kedalam *store redux* dengan cara melakukan aksi ’ADD_SENSOR_DATA’ menggunakan *dispatcher*. Karena data yang dikirim oleh sensor berupa JSON, maka perlu diproses dulu menggunakan `JSON.parse`. Implementasi tertera pada Kode Sumber 4.49.

```
1  . . .
2  mqtt.on('message', (topic, payload) => {
3    store.dispatch({
4      type: 'ADD_SENSOR_DATA',
5      sensorData: JSON.parse(payload.toString()),
6    });
7  })
8  . . .
```

Kode Sumber 4.49: Memasukkan data rekaman sensor ke dalam *store redux*

Ketika data rekaman sensor dikirim oleh *node* sensor, MQTT pada aplikasi Web akan menerima pesan tersebut dan memanggil fungsi diatas. Lalu, aksi 'ADD_SENSOR_DATA' dipanggil untuk memasukkan data rekaman sensor yang diterima ke dalam *store redux*. Ketika *state* pada *store redux* itu berubah, antarmuka *chart* akan otomatis berubah.

4.6.3 Implementasi Antarmuka Web

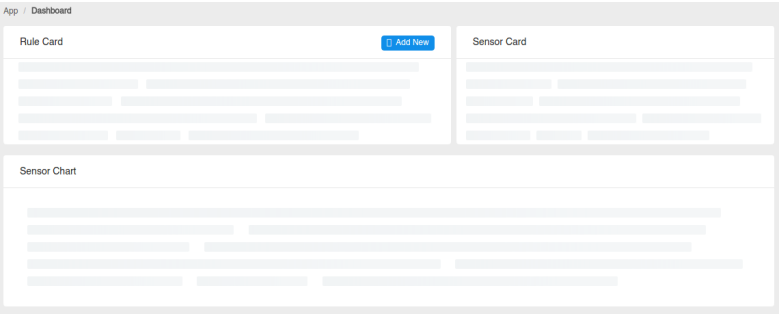
Subbab ini membahas mengenai implementasi antarmuka yang telah dirancang pada subbab 3.4.6.3. Implementasi antarmuka Web menggunakan *package react* untuk membangun aplikasi Web. Lalu untuk mempercepat proses pengembangan dalam membuat tampilan, *ant design* digunakan sebagai komponen dasar membangun aplikasi Web ini. Recharts digunakan sebagai *package* untuk memenuhi kebutuhan menampilkan grafik.

4.6.3.1 Implementasi *Dashboard*

Antarmuka *dashboard* terdiri dari dua komponen utama dan dua komponen opsional. Komponen utamanya yaitu *SensorCard* yang berfungsi untuk menampilkan daftar sensor dan *RuleCard* yang berfungsi untuk menampilkan daftar *rule*. Komponen opsionalnya yaitu *SensorChart* yang ditampilkan ketika *state* sensor pilihan tidak kosong dan *InvokedRuleCard* yang ditampilkan ketika *rule* pilihan tidak kosong. Antarmuka *dashboard* menggunakan komponen *grid* yang disediakan *ant design* untuk membagi area untuk setiap komponen yang ada pada *dasbhoard*. komponen *ant design* yang digunakan yaitu *row* dan *col*. Implementasi untuk mengimplementasikan antarmuka *dashboard* tertera pada Kode Sumber 4.50.

```
import { Col, Row } from 'antd';
class Dashboard extends Component {
 render() { return (
     <Row>
       <Col md={24}>
         <Row gutter={8}>
           <Col md={14}> <RuleCard {...this.props.sensors} /> </Col
               >
           <Col md={10}> <SensorCard {...this.props.rules} /> </Col
               >
         </Row>
       </Col>
       <Col md={24}>
         {this.props.selectedSensor !== null && <SensorChartCard
             />}
         {this.props.selectedRule !== null && <InvokedRuleCard />}
       </Col>
     </Row>
); }; }
```

Kode Sumber 4.50: Implementasi antarmuka *dashboard*



Gambar 4.3: Implementasi antarmuka *Dashboard*

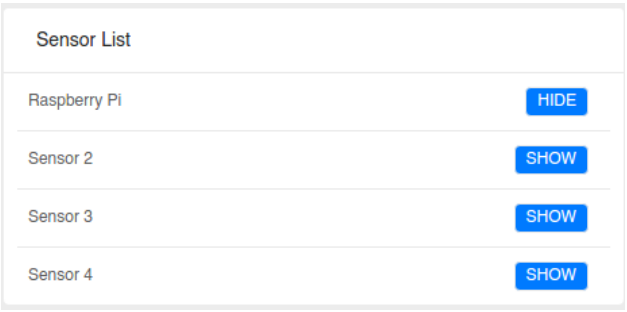
4.6.3.2 Implementasi *Sensor Card*

SensorCard berfungsi untuk menampilkan daftar sensor serta melakukan aksi untuk menampilkan grafik data rekaman sensor secara *real-time*. Daftar sensor dan fungsi *changeSensor* untuk memilih sensor berasal dari hasil implementasi fungsionalitas

daftar sensor. Untuk merepresentasikan komponen ini, dibutuhkan tiga komponen *ant design*, yaitu *card* sebagai pembungkus komponen *SensorCard*, *table* untuk menampilkan daftar sensor dan *tag* sebagai representasi tombol untuk memilih sensor. Implementasi tertera pada Kode Sumber 4.51.

```
import { Card, Table, Tag } from 'antd';
const { Column } = Table;
const SensorCard = ({ sensors, changeSensor }) => (
 <Card>
   <Table rowKey='id' dataSource={sensors}>
     <Column title='Name'  dataIndex='name' key='name' />
     <Column title='Status' dataIndex='status' key='status'
       onCellClick={(sensor) => {
         changeSensor(sensor, selectedSensor.trueid)
       }}
       render={(status, sensor) => {
         return <Tag className='background color-green'>SHOW</Tag>;
       }}
     />
   </Table>
 </Card>
);
```

Kode Sumber 4.51: Implementasi antarmuka *SensorCard*



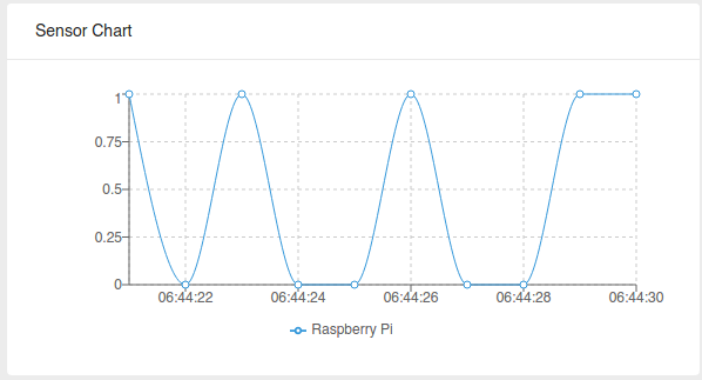
Gambar 4.4: Implementasi antarmuka *Sensor Card*

4.6.3.3 Implementasi *Sensor Chart*

SensorChart berfungsi untuk menampilkan grafik data rekaman sensor secara *real-time*. Data rekaman sensor diperoleh dari *store redux* pada implementasi fungsionalitas *real-time chart*. Komponen ini menggunakan *package recharts* untuk merepresentasikan datanya. Komponen yang dipakai yaitu *linechart* sebagai jenis grafik, *line* untuk representasi garis pada grafik, *xaxis* dan *yaxis* untuk representasi garis x dan y, dan *legend* untuk menampilkan detail data rekamannya. Implementasi tertera pada Kode Sumber 4.52.

```
import { LineChart, Line, XAxis, YAxis, Legend } from 'recharts';
export default class SensorChart extends Component {
  render() { return (
      <LineChart data={this.props.sensorDatas} >
        <XAxis dataKey="name"/> <YAxis/>
        <CartesianGrid strokeDasharray="3 3"/> <Legend />
        <Line type="monotone"  dataKey='data'
          name={this.props.sensorName} stroke='#3498db' />
      </LineChart>
) } }
```

Kode Sumber 4.52: Implementasi antarmuka *SensorChart*



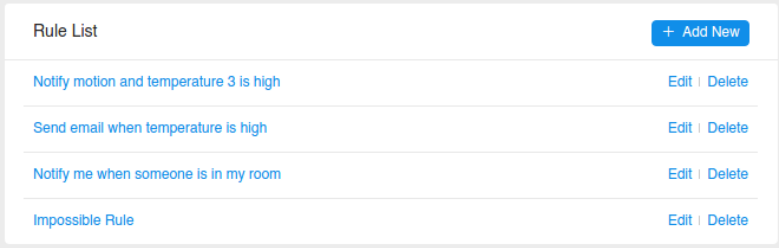
Gambar 4.5: Implementasi antarmuka *Sensor Chart*

4.6.3.4 Implementasi *Rule Card*

RuleCard berfungsi untuk menampilkan daftar *rule* serta melakukan aksi untuk menambah *rule*, mengubah *rule*, menghapus *rule*, dan menampilkan *invoked rule*. Daftar *rule* dan fungsi *addRule*, *editRule*, *deleteRule*, dan *invoked rule* berasal dari hasil implementasi fungsionalitas daftar *rule*. Untuk merepresentasikan komponen ini, dibutuhkan tiga komponen *ant design*, yaitu *card* sebagai pembungkus komponen *RuleCard*, *table* untuk menampilkan daftar *rule* dan *button* sebagai tombol untuk melakukan aksi tambah *rule*. Implementasi tertera pada Kode Sumber 4.53.

```
1  import { Button, Card, Table } from 'antd';
2  const { Column } = Table;
3  const RuleCard = ({rules, addRule, editRule,
4    deleteRule, showInvoked }) => (
5    <Card bordered={false}
6      extra={ <Button onClick={() => addRule()} icon='plus'>
7    Add New</Button> } >
8      <Table rowKey='id' dataSource={rules}>
9        <Column title='Name' dataIndex='name' key='name'
10         render={(ruleName, rule) => (
11           <span> <a href="#" onClick={() => showInvoked(rule)}>
12           {ruleName}</a> </span>
13         )}
14       />
15       <Column title='Action'  key='action' dataIndex='id'
16         render={(ruleID, rule) => (
17           <span>
18             <a href='#' onClick={() => editRule(ruleID)} >Edit</a>
19             <span className='ant-divider' />
20             <a href='#' onClick={() => deleteRule(rule)} >Delete</a
                   >
21           </span>
22       )} />
23     </Table>
24   .  .  .
```

Kode Sumber 4.53: Implementasi antarmuka *Rule Card*



Gambar 4.6: Implementasi antarmuka *Rule Card*

Pada komponen ini juga terdapat sub komponen *RuleForm* yang berfungsi untuk modifikasi data *rule*. Komponen ini ditampilkan menggunakan komponen *modal* dari *ant design*. Implementasi untuk menyiapkan *RuleForm* tertera pada Kode Sumber 4.54.

```
import { Modal } from 'antd';
import RuleForm from 'containers/RuleForm';
  .  .  .
  <Modal title={title} visible={visibility} okText='Save'>
    <RuleForm form={form} sensors={this.props.sensors}
      actions={this.props.actions} item={rule}
    />
  </Modal>
  .  .  .
```

Kode Sumber 4.54: Implementasi antarmuka modal untuk *RuleForm*

4.6.3.5 Implementasi *Rule Form*

RuleForm berfungsi sebagai antarmuka untuk menambah atau mengubah *rule*. Data *rule*, sensor, *action* dan fungsi untuk menambah atau mengubah *rule* berasal dari properti yang dikirimkan oleh komponen induknya (*RuleCard*). Untuk membuat *form* menggunakan *ant design*, langkah pertama yaitu dengan mendeklarasikan komponen *form*, lalu diisi dengan komponen *Form.Item*. Kode untuk mendeklarasikan *field* tertera pada Kode Sumber 4.55.

```
import { Form } from 'antd';
<Form layout='horizontal'>
  <Form.Item hasFeedback> . . . </Form.Item>
</Form>
```

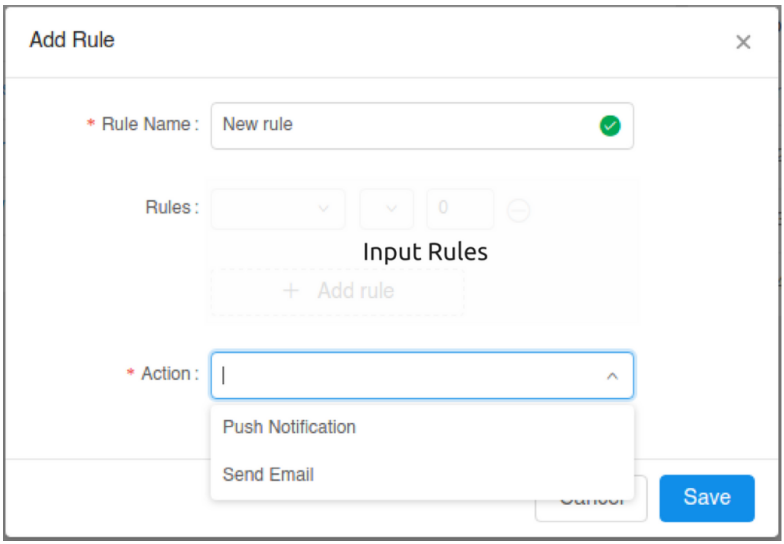
Kode Sumber 4.55: Membuat form menggunakan komponen *ant design*

Lalu, panggil fungsi *getFieldDecorator* untuk mendeklarasikan nama *field* dan representasi komponen untuk *field* tersebut.

Terdapat beberapa *field* yang dibutuhkan untuk membuat atau mengubah *rule*. *Field* pertama yaitu nama *rule*. *Field* ini direpresentasikan menggunakan komponen *input*. *Field* kedua yaitu *action*. *Field* ini berisi daftar *action* yang tersedia pada sistem. *Field* ini direpresentasikan menggunakan komponen select. *Field* yang terakhir yaitu *rule expression*. *Field* ini cukup kompleks sehingga akan dibuat pada subbab selanjutnya. Pada bagian ini, komponen tersebut hanya perlu dimasukkan kedalam komponen *form*. Implementasi untuk mengimplementasikan ketiga *field* tersebut tertera pada Kode Sumber 4.56.

```
<Form.Item label='Rule Name' hasFeedback>
  {getFieldDecorator('name', {}) (<Input />) }
</Form.Item>
{inputRules}
<Form.Item label='Action' hasFeedback>
  {getFieldDecorator('action', {}) (
  <Select placeholder='Please select an action'>
      {this.props.actions.map((action) =>
        <Option key={action.id}
          value={String(action.trueid)}>
        {action.name}</Option>
      )}
    </Select>
  ) }
</Form.Item>
```

Kode Sumber 4.56: Implementasi *field* pada *RuleForm*



Gambar 4.7: Implementasi Antarmuka *Rule Form*

4.6.3.6 Implementasi *Rule Expression*

Rule expression terdiri dari aturan beberapa sensor. Aturan tersebut menggunakan operator relasional untuk membandingkan data rekaman sensor dengan konfigurasi pengguna. Selain itu, aturan tiap sensor juga dapat dibandingkan dengan aturan lainnya menggunakan operator logika ”AND” dan ”OR”. Berdasarkan penjelasan tersebut, dapat disimpulkan bahwa dibutuhkan empat *field* untuk tiap aturannya.

Field pertama yaitu nama sensor. *Field* ini direpresentasikan menggunakan komponen *select*. *Field* kedua yaitu operator relasional. *Field* ini berisi daftar operator yang digunakan sebagai pembanding. *Field* ini direpresentasikan menggunakan komponen *select*. *Field* ketiga yaitu nilai pembanding. *Field* ini direpresentasikan menggunakan komponen *inputNumber*. *Field* yang terakhir yaitu operator logika. *Field* ini berfungsi untuk

membandingkan aturan sensor dengan aturan selanjutnya. Implementasi untuk mengimplementasikan *rule expression* tertera pada Kode Sumber 4.57.

```
<span>
  <Select placeholder='Please select a sensor'>

  </Select>
  <Select placeholder='operator'>
    <Option value="<">&lt;</Option> <Option value="<=">&le;</Option
        >
    <Option value="==">=</Option> <Option value=">=">&ge;</Option>
    <Option value=">">&gt;</Option>
  </Select>
  <InputNumber placeholder='value' />
  <Select placeholder='AND/OR'>
    <Option value="AND">AND</Option> <Option value="OR">OR</Option>
  </Select>
</span>
```

Kode Sumber 4.57: Implementasi antarmuka aturan tiap sensor

Agar dapat membuat *rule expression*, aturan sensor tersebut dapat ditambah atau dikurangi sesuai dengan keinginan pengguna. Sehingga dibutuhkan tombol untuk menambah aturan dan mengurangi aturan. Selain itu, ketika mengubah *rule*, perulangan perlu dilakukan untuk tiap aturan yang ada di basis data. Sehingga komponen *rule expression* menjadi seperti yang tertera pada Kode Sumber 4.58.

```
  . . .
const inputRules = ruleList.map((ruleDetail, index) => {
 return (
   <Form.Item>
    {getFieldDecorator(`rule-${ruleDetail.key}`, {})
    (<InputRule sensor={this.props.sensors} />)}
    <Icon className="dynamic-delete-button" type="minus-circle-o"
        />
   </Form.Item>
 )
})
  . . .
```

Kode Sumber 4.58: Implementasi antarmuka *rule expression*

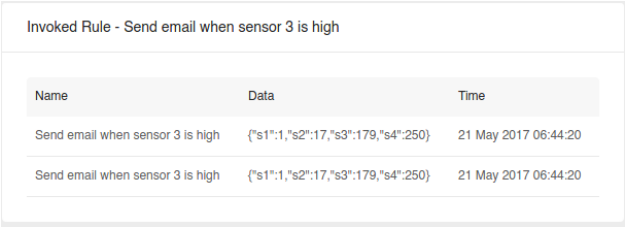
Gambar 4.8: Implementasi antarmuka *rule expression*

4.6.3.7 Implementasi *InvokedRuleCard*

InvokedRuleCard berfungsi untuk menampilkan daftar *invoked rule* dari *rule* yang dipilih. Daftar *invoked rule* berasal dari hasil implementasi fungsionalitas tampil *invoked rule*. Untuk merepresentasikan komponen ini, dibutuhkan dua komponen *ant design*, yaitu *card* sebagai pembungkus komponen *InvokedRuleCard* dan *table* untuk menampilkan daftar *invoked rule*. Data yang ditampilkan pada tabel tersebut yaitu nama *rule*, data rekaman sensor yang terpenuhi dan waktu ketika *rule* terpenuhi. Implementasi tertera pada Kode Sumber 4.59.

```
import { Card, Table, Tag } from 'antd';
const { Column } = Table;
const InvokedRuleCard = ({ rule }) => (
 <Card>
   <Table rowKey='id' dataSource={invokedRules}>
     <Column title='Name' dataIndex='rulename' key='rulename' />
     <Column title='Data' dataIndex='data' key='data' />
     <Column title='Time' dataIndex='time' key='time' />
   </Table>
 </Card>
);
```

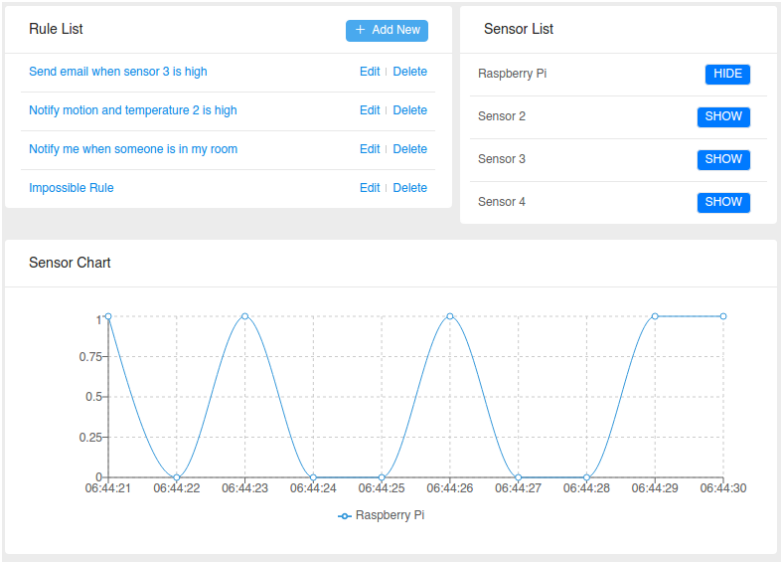
Kode Sumber 4.59: Implementasi antarmuka daftar *invoked rule*



Gambar 4.9: Implementasi antarmuka daftar *invoked rule*

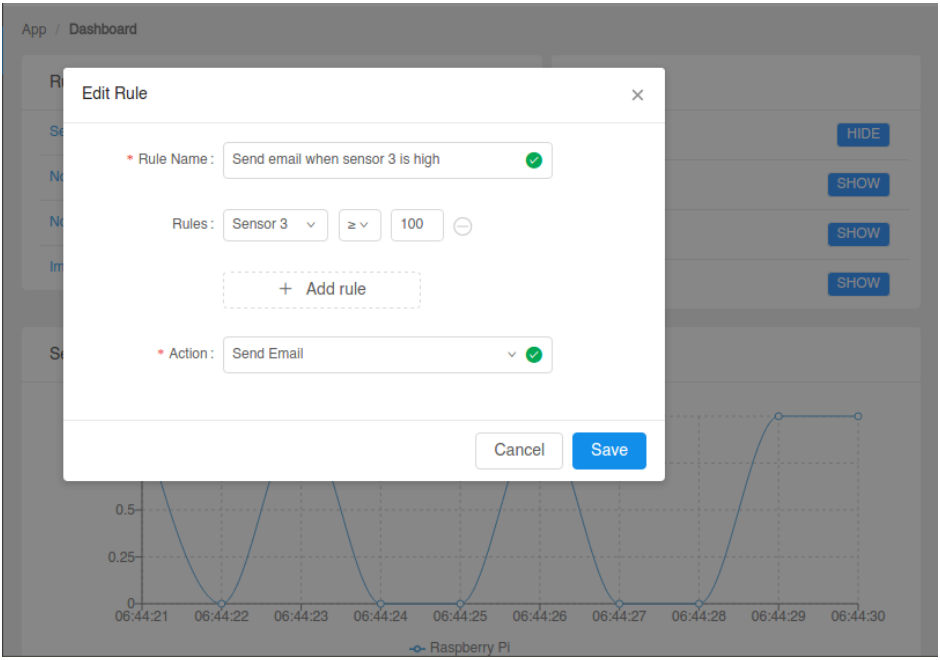
4.6.3.8 Implementasi Antarmuka Aplikasi Web

Berikut hasil implementasi antarmuka aplikasi Web secara keseluruhan. Ketika aplikasi Web dibuka, terdapat daftar *rule* dan daftar sensor. Lalu ketika tombol *show* pada sensor diklik, maka akan menampilkan grafik yang di*update* secara *real-time*.



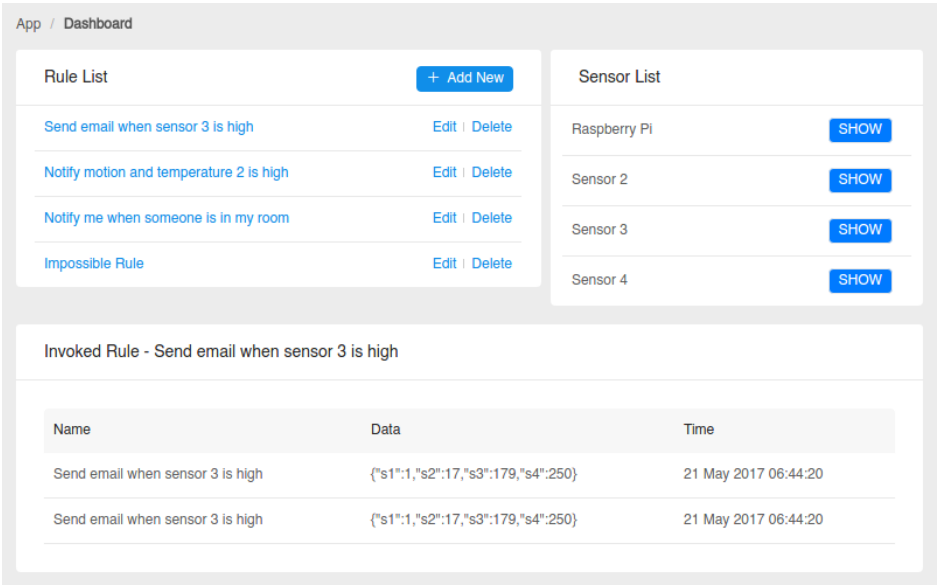
Gambar 4.10: Implementasi antarmuka Web - *Real-Time Chart*

Lalu, ketika tombol tambah atau *edit* diklik, maka akan memunculkan form untuk memodifikasi *rule*.



Gambar 4.11: Implementasi antarmuka Web - modifikasi *rule*

Ketika *rule* diklik, maka akan menampilkan daftar *invoked rule*.



Gambar 4.12: Implementasi antarmuka Web - daftar *InvokedRule*

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

Hal-hal yang akan dibahas pada bab ini adalah lingkungan uji coba perangkat lunak, uji coba fungsionalitas perangkat lunak, dan uji coba performa perangkat lunak. Uji coba akan dilakukan dengan beberapa skenario.

5.1 Lingkungan Uji Coba

Subbab lingkungan uji coba menjelaskan mengenai lingkungan tempat pengujian perangkat lunak. Lingkungan uji coba pada kasus ini terdiri dari empat komponen. Komponen pertama yaitu satu *server* untuk menjalankan *graphql server*, MQTT *broker*, AutoAI, dan Web *server*. Komponen kedua yaitu Raspberry Pi untuk menjalankan *node* sensor. Komponen ketiga yaitu *smartphone* Android untuk menjalankan aplikasi Android. Komponen terakhir yaitu laptop untuk mengakses aplikasi Web dan uji coba performa. Spesifikasi untuk tiap komponen adalah sebagai berikut:

- Server:
  - Perangkat keras:
    * Intel Core i5-4210U 1.70GHz
    * RAM DDR3 4 GB
    * Ethernet interface Express Gigabit Ethernet Controller speed=100Mbit/s
  - Perangkat lunak:
    * Sistem Operasi Ubuntu 16.04 LTS 64 Bit
    * Python versi 2.7
    * Mosquitto versi 1.4.10
    * PostgreSQL versi 9.6
    * Golang versi 1.8
    * Htop versi 2.0.1
    * NodeJS versi 6.10.3
    * Yarn versi 0.23

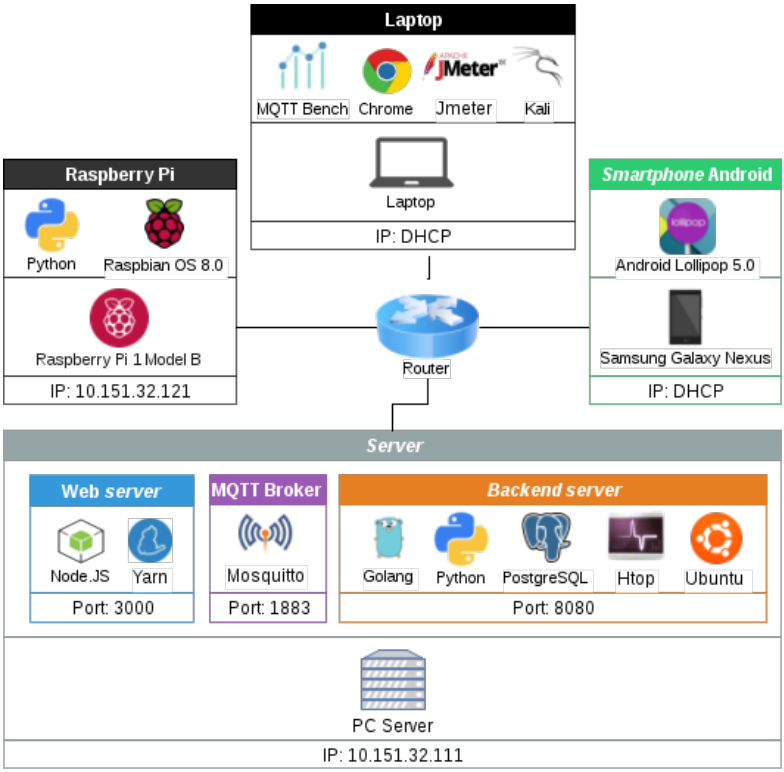
- Raspberry Pi:
  - Perangkat keras:
    * Raspberry Pi 1 Model B
  - Perangkat lunak:
    * Sistem Operasi Raspbian 8.0
    * Python versi 2.7
- *Smartphone* Android:
  - Perangkat keras:
    * Samsung Galaxy Nexus
    * Processor TI OMAP 4460
    * RAM 1 GB
  - Perangkat lunak:
    * Sistem Operasi Android 5.0 Lollipop
- Laptop:
  - Perangkat keras:
    * Processor Intel Pentium 2117U 1.80GHz
    * RAM 4 GB
  - Perangkat lunak:
    * Sistem Operasi Kali Linux 2016.2
    * Peramban Google Chrome 53
    * MQTT-bench[1] untuk uji performa MQTT *broker*
    * Apache JMeter[2] untuk uji coba performa kasus pembanding

Agar tiap komponen dapat berkomunikasi, dibutuhkan konfigurasi alamat IP statis pada beberapa komponen, yaitu:

- MQTT *broker* memiliki alamat IP 10.151.32.111:1883
- Web *server* memiliki alamat IP 10.151.32.111:3000
- GraphQL *server* memiliki alamat IP 10.151.32.111:8080
- *Node* sensor memiliki alamat IP 10.151.32.121

[1] MQTT-bench dapat diakses di https://github.com/takanorig/mqtt-bench
[2] Apache JMeter dapat diunduh di http://jmeter.apache.org/

Visualisasi arsitektur lingkungan uji coba tertera pada Gambar 5.1.



Gambar 5.1: Arsitektur Uji Coba

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan tentang skenario uji coba perangkat lunak yang telah dibangun. Skenario uji coba dibagi menjadi dua bagian, yaitu uji coba fungsionalitas dan uji coba performa perangkat lunak.

5.2.1 Skenario Uji Fungsionalitas

Uji coba fungsionalitas bertujuan untuk menguji apakah fungsi-fungsi utama pada perangkat lunak berhasil di implementasikan dan berjalan sesuai dengan yang diharapkan. Uji coba ini didasarkan pada kasus penggunaan yang telah dibuat pada subbab 3.3. Selain itu terdapat uji coba fungsionalitas tambahan yaitu melakukan otomatisasi aksi ketika data rekaman sensor dikirimkan ke AutoAI dan evaluasi *rule*. Skenario uji coba ditampilkan dalam bentuk tabel.

5.2.1.1 Uji Melihat Daftar Sensor pada Aplikasi Android

Tabel 5.1: Prosedur uji coba melihat daftar sensor pada aplikasi Android

ID	UJ-01
Nama	Uji Coba Melihat Daftar Sensor pada Aplikasi Android
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan daftar sensor pada aplikasi Android
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Aplikasi Android dijalankan 3. Pengguna menekan spinner
Masukan	-
Keluaran	Daftar sensor pada aplikasi Android
Hasil Harapan	Daftar sensor ditampilkan pada aplikasi Android

5.2.1.2 Uji Menampilkan Grafik Data Rekaman Sensor pada Aplikasi Android

Tabel 5.2: Prosedur uji coba menampilkan grafik data rekaman sensor pada aplikasi Android

ID	UJ-02
Nama	Uji Coba Menampilkan Grafik Data Rekaman Sensor pada Aplikasi Android
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan grafik data rekaman sensor secara *real-time* pada aplikasi Android
Kondisi Awal	Perangkat lunak dijalankan dan aplikasi Android sudah terpasang pada *smartphone* android
Skenario	1. Perangkat lunak dijalankan 2. Fungsi merekam data sensor pada raspberry pi dan python dijalankan 3. Aplikasi Android dijalankan 4. Pengguna memilih sensor pada spinner
Masukan	Sensor pilihan pengguna
Keluaran	Grafik data rekaman sensor
Hasil Harapan	Grafik data rekaman sensor ditampilkan secara *real-time*

5.2.1.3 Uji Melihat Daftar Sensor pada Aplikasi Web

Tabel 5.3: Prosedur uji coba melihat daftar sensor pada aplikasi web

ID	UJ-03
Nama	Uji Coba Melihat Daftar Sensor pada Aplikasi Web
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan daftar sensor pada aplikasi web
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop
Masukan	-
Keluaran	Daftar sensor pada aplikasi web

Tabel 5.3: Prosedur uji coba melihat daftar sensor pada aplikasi web

Hasil Harapan	Daftar sensor ditampilkan pada aplikasi web

5.2.1.4 Uji Menampilkan Grafik Data Rekaman Sensor pada Aplikasi Web

Tabel 5.4: Prosedur uji coba menampilkan grafik data rekaman sensor pada aplikasi web

ID	UJ-04
Nama	Uji Coba Menampilkan Grafik Data Rekaman Sensor pada Aplikasi Web
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan grafik data rekaman sensor secara *real-time* pada aplikasi web
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Fungsi merekam data sensor pada Raspberry Pi dan Python dijalankan 3. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 4. Pengguna menekan tombol show pada sensor pilihan
Masukan	1. Data rekaman lingkungan dari sensor 2. Sensor pilihan pengguna
Keluaran	Grafik data rekaman sensor
Hasil Harapan	Grafik data rekaman sensor ditampilkan secara *real-time*

5.2.1.5 Uji Menambah *Rule*

Tabel 5.5: Prosedur uji coba menambah *rule*

ID	UJ-05
Nama	Uji Coba Menambah *Rule*
Tujuan Uji Coba	Menguji fungsionalitas untuk menambah *rule*
Kondisi Awal	Perangkat lunak dijalankan

Tabel 5.5: Prosedur uji coba menambah *rule*

Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 3. Pengguna menekan tombol *add new* pada *RuleCard* 4. Pengguna mengisi *form* untuk menambahkan *rule* 5. Pengguna menekan tombol *save* pada *RuleForm*
Masukan	Data *rule* baru
Keluaran	*Rule* baru yang berhasil disimpan
Hasil Harapan	1. *Rule* berhasil disimpan ke dalam sistem 2. AutoAI melakukan otomatisasi aksi pada *rule* yang dimasukkan

5.2.1.6 Uji Melihat Daftar *Rule*

Tabel 5.6: Prosedur uji coba melihat daftar *rule*

ID	UJ-06
Nama	Uji Coba Melihat Daftar *Rule*
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan daftar *rule*
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop
Masukan	-
Keluaran	Daftar *rule* pada aplikasi Web
Hasil Harapan	Daftar *rule* ditampilkan pada aplikasi Web

5.2.1.7 Uji Mengubah *Rule*

Tabel 5.7: Prosedur uji coba mengubah *rule*

ID	UJ-07
Nama	Uji Coba Mengubah *Rule*

Tabel 5.7: Prosedur uji coba mengubah *rule*

Tujuan Uji Coba	Menguji fungsionalitas untuk mengubah *rule* yang tersimpan pada sistem
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 3. Pengguna menekan tombol *edit* pada *rule* yang ingin diubah 4. Pengguna mengubah data *rule* pada *form* 5. Pengguna menekan tombol *save* pada *RuleForm*
Masukan	Data perubahan *rule*
Keluaran	Data *rule* berubah
Hasil Harapan	1. Data *rule* berubah pada sistem 2. AutoAI melakukan otomatisasi aksi berdasarkan konfigurasi *rule* baru

5.2.1.8 Uji Menghapus *Rule*

Tabel 5.8: Prosedur uji coba menghapus *rule*

ID	UJ-08
Nama	Uji Coba Menghapus *Rule*
Tujuan Uji Coba	Menguji fungsionalitas untuk menghapus *rule* yang tersimpan pada sistem
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 3. Pengguna menekan tombol *delete* pada *rule* yang ingin dihapus
Masukan	*Rule* yang ingin dihapus
Keluaran	Data *rule* terhapus
Hasil Harapan	1. Data *rule* terhapus pada sistem 2. AutoAI tidak melakukan otomatisasi aksi pada *rule* tersebut

5.2.1.9 Uji Melihat *Invoked Rule*

Tabel 5.9: Prosedur uji coba melihat *invoked rule*

ID	UJ-09
Nama	Uji Coba Melihat *Invoked Rule*
Tujuan Uji Coba	Menguji fungsionalitas untuk menampilkan daftar *invoked rule*
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 3. Pengguna memilih *rule* untuk melihat *invoked rule*
Masukan	*rule* yang ingin menampilkan daftar *invokedrule*
Keluaran	Daftar *invoked rule* pada aplikasi Web
Hasil Harapan	Daftar *invoked rule* ditampilkan pada aplikasi web

5.2.1.10 Uji Mengirimkan Data Rekaman Sensor

Tabel 5.10: Prosedur uji coba mengirimkan data rekaman sensor

ID	UJ-10
Nama	Uji Coba Mengirimkan Data Rekaman Sensor
Tujuan Uji Coba	Menguji fungsionalitas untuk mengirimkan data rekaman sensor
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Raspberry Pi menjalankan *script* python untuk mengirimkan data rekaman lingkungan
Masukan	-
Keluaran	Data rekaman lingkungan
Hasil Harapan	1. Data rekaman lingkungan dikirimkan ke MQTT *broker* 2. Data rekaman tersimpan pada basis data

5.2.1.11 Uji Melakukan Otomatisasi Aksi

Pada uji coba ini, aksi yang akan dilakukan ketika *rule* terpenuhi adalah mengirimkan *push notification* ke aplikasi web dan Android. Kode sumber untuk melakukan *push notification* menggunakan metode *publish/subscribe* mengacu pada lampiran 1.1.

Tabel 5.11: Prosedur uji coba melakukan otomatisasi aksi

ID	UJ-11
Nama	Uji Coba Melakukan Otomatisasi Aksi
Tujuan Uji Coba	Menguji fungsionalitas untuk melakukan otomatisasi aksi berdasarkan *rule* yang tersimpan pada sistem
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Pengguna mengakses laman 10.151.32.111:3000 melalui web *browser* Google Chrome menggunakan laptop 3. Pengguna menekan tombol *add new* pada *RuleCard* 4. Pengguna mengisi *form* untuk menambahkan *rule* dan memilih aksi *push notification* 5. Pengguna menekan tombol *save* pada *RuleForm* 6. Raspberry Pi dan sensor simulasi menjalankan *script* Python untuk mengirimkan data rekaman lingkungan yang memenuhi *rule* yang telah ditambahkan ke sistem
Masukan	Data *rule* baru Data rekaman sensor yang memenuhi *rule*
Keluaran	*Push Notification* pada aplikasi Web dan Android
Hasil Harapan	1. *Rule* berhasil terpenuhi 2. Aplikasi Web menerima notifikasi bahwa *rule* terpenuhi 2. Aplikasi Android menerima notifikasi bahwa *rule* terpenuhi

5.2.1.12 Uji Evaluasi Rule

Uji coba ini bertujuan untuk mengecek bahwa evaluasi *rule* menghasilkan hasil yang sama dengan hasil yang sebenarnya. Skenario uji coba evaluasi *rule* tertera pada Tabel 5.12

Tabel 5.12: Skenario uji coba evaluasi *rule*

Rule Expression	**Data Rekaman Sensor**	**Hasil Harapan**
Raspberry Pi == 1	Raspberry Pi: 1	Benar
Raspberry Pi == 1 AND Sensor 2 > 20	Raspberry Pi: 1 Sensor 2: 17	Salah
Sensor 3 > 40 OR Sensor 4 < 50	Sensor 3: 45 Sensor 4: 55	Benar
Raspberry Pi == 1 OR Sensor 2 > 20 OR Sensor 3 < 40	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45	Benar
Sensor 2 > 20 AND Sensor 3 > 40 OR Sensor 4 >= 60	Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah
Raspberry Pi == 1 AND Sensor 3 >= 37 AND Sensor 4 > 52	Raspberry Pi: 1 Sensor 3: 45 Sensor 4: 55	Benar
Raspberry Pi == 0 OR Sensor 2 >= 25 AND Sensor 3 > 37 AND Sensor 4 >= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah
Raspberry Pi == 0 AND Sensor 2 <= 25 OR Sensor 3 > 37 AND Sensor 4 <= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah
Raspberry Pi == 1 AND Sensor 2 > 25 OR Sensor 3 >= 37 AND Sensor 4 >= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Benar
Raspberry Pi == 0 AND Sensor 2 < 25 OR Sensor 3 <= 37 AND Sensor 4 <= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah

5.2.2 Skenario Uji Performa

Uji coba performa merupakan pengujian yang bertujuan untuk menguji tingkat kehandalan suatu perangkat lunak, baik dalam segi kecepatan (*speed*), ketahanan (*robustness*), maupun efisiensi penggunaan sumber daya. Uji coba ini juga mengukur penggunaan *bandwidth* saat sensor melakukan pengiriman data ke *server*. Pada sistem ini, terdapat dua komponen yang diuji performanya, yaitu MQTT *broker* dan AutoAI. Penjelasan setiap komponen dari uji coba performa dijelaskan pada subbab berikut.

5.2.2.1 Uji Performa Evaluasi Kolaborasi Data Sensor

Pengujian evaluasi kolaborasi data sensor dilakukan untuk mengetahui performa dan penggunaan sumber daya sistem ketika mengevaluasi aturan kolaborasi data sensor. Prosedur uji performa evaluasi kolaborasi data sensor tertera pada Tabel 5.13.

Tabel 5.13: Prosedur uji performa evaluasi kolaborasi data sensor

ID	UP-01
Nama	Uji Performa Kolaborasi Data Sensor
Tujuan Uji Coba	1. Menguji kecepatan evaluasi *rule* 2. Menguji penggunaan sumber daya
Parameter	Jumlah *rule* = {100, 200, 500, 1000, 2000, 5000, 10000}
Frekuensi	5 kali percobaan
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. Basis data diisi dengan *rule* sejumlah *parameter* 3. HTOP dijalankan pada *server* dengan *filter* `AutoAI` 4. *Node* sensor dijalankan untuk mengirimkan data ke AutoAI
Hasil Uji Coba	1. Waktu rata-rata sejumlah *rule* dievaluasi dalam satuan *milisecond*(ms) 2. Penggunaan RAM oleh AutoAI tiap sejumlah *rule* 3. Penggunaan CPU oleh AutoAI tiap sejumlah *rule*

5.2.2.2 Uji Performa AutoAI

Pengujian AutoAI dilakukan untuk mengetahui kecepatan, penggunaan sumber daya dan kapabilitas AutoAI dalam menangani data rekaman sensor yang dikirim oleh *node* sensor serta memproses *rule* berdasarkan data rekaman sensor yang masuk. Uji performa ini bertujuan untuk mengetahui performa AutoAI dalam menangani data rekaman sensor yang diterima dalam waktu bersamaan serta melakukan otomatisasi aksi berdasarkan *rule* yang tersimpan pada sistem. Prosedur uji performa AutoAI tertera pada Tabel 5.14.

Tabel 5.14: Prosedur uji performa AutoAI

ID	UP-02
Nama	Uji Performa AutoAI
Tujuan Uji Coba	1. Menguji kecepatan pemrosesan otomatisasi aksi 2. Menguji penggunaan sumber daya 3. Mengetahui kapabilitas AutoAI
Parameter	Kecepatan pengiriman pesan = ± 20.000/detik Aksi = *Push notification* Pesan = {100, 200, 500, 1000, 2000, 5000, 10000}
Frekuensi	5 kali percobaan
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. HTOP dijalankan pada *server* dengan *filter* `AutoAI` 3. Wireshark dijalankan pada laptop dengan konfigurasi *capture filter* `host 10.151.32.111` 4. Mqtt-bench dijalankan pada laptop dengan *parameter* pesan dari jumlah terendah hingga jumlah tertinggi
Hasil Uji Coba	1. Waktu rata-rata AutoAI memproses *rule* berdasarkan data rekaman sensor yang diterima 2. Penggunaan RAM oleh AutoAI tiap sejumlah pesan 3. Penggunaan CPU oleh AutoAI tiap sejumlah pesan 4. Penggunaan *Bandwidth* oleh AutoAI tiap sejumlah pesan

Sebagai perbandingan, penulis melakukan uji performa menggunakan protokol HTTP yang melakukan proses AutoAI, yaitu menyimpan data rekaman sensor dan menyimpan data *invokedRule*. Uji performa ini dilakukan untuk membandingkan performa sistem yang dibuat menggunakan teknologi yang dirancang pada tugas akhir ini dengan protokol HTTP dan bahasa pemrograman Python. Kode sumber sistem pembanding tertera pada Lampiran 1.2. Prosedur uji performa pembanding tertera pada Tabel 5.15.

Tabel 5.15: Prosedur uji performa pembanding AutoAI

ID	UP-03
Nama	Uji Performa pembanding AutoAI
Tujuan Uji Coba	1. Menguji kecepatan pemrosesan pembanding AutoAI 2. Menguji penggunaan sumber daya pembanding AutoAI 3. Membandingkan performa antara AutoAI dengan sistem pembanding
Parameter	Pesan = {100, 200, 500, 1000, 2000, 5000, 10000}
Frekuensi	5 kali percobaan
Kondisi Awal	Perangkat lunak dijalankan
Skenario	1. Perangkat lunak dijalankan 2. HTOP dijalankan pada *server* dengan *filter* `Python http.py` 3. Wireshark dijalankan pada laptop dengan konfigurasi *capture filter* `host 10.151.32.111` 4. Apache JMeter dijalankan pada laptop dengan *parameter* pesan dari jumlah terendah hingga jumlah tertinggi
Hasil Uji Coba	1. Waktu rata-rata pembanding AutoAI memproses *rule* berdasarkan data rekaman sensor yang diterima 2. Penggunaan RAM oleh pembanding AutoAI tiap sejumlah pesan 3. Penggunaan CPU oleh pembanding AutoAI tiap sejumlah pesan 4. Penggunaan *bandwidth* oleh pembanding AutoAI tiap sejumlah pesan

5.3 Hasil Uji Coba dan Evaluasi

Berikut disampaikan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Hasil pengujian fungsionalitas pada sistem yang sudah dibangun tertera pada Tabel 5.16 dan Tabel 5.17.

Tabel 5.16: Hasil uji coba fungsionalitas

Kode	Hasil Harapan	Hasil
UJ-01	Daftar sensor ditampilkan pada aplikasi Android	Terpenuhi
UJ-02	Grafik data rekaman sensor ditampilkan secara *real-time*	Terpenuhi
UJ-03	Daftar sensor ditampilkan pada aplikasi web	Terpenuhi
UJ-04	Grafik data rekaman sensor ditampilkan secara *real-time*	Terpenuhi
UJ-05	*Rule* berhasil disimpan ke dalam sistem	Terpenuhi
	AutoAI melakukan otomatisasi aksi pada *rule* yang dimasukkan	Terpenuhi
UJ-06	Daftar *rule* ditampilkan pada aplikasi web	Terpenuhi
UJ-07	Data *rule* berubah pada sistem	Terpenuhi
	AutoAI melakukan otomatisasi aksi pada *rule* baru	Terpenuhi
UJ-08	Data *rule* terhapus pada sistem	Terpenuhi
	AutoAI tidak melakukan otomatisasi aksi pada *rule* yang dihapus	Terpenuhi
UJ-09	Daftar *invoked rule* ditampilkan pada aplikasi web	Terpenuhi
UJ-10	Data rekaman lingkungan dikirimkan ke MQTT *broker*	Terpenuhi
	Data rekaman lingkungan tersimpan pada basis data	Terpenuhi
UJ-11	*Rule* berhasil terpenuhi	Terpenuhi
	Aplikasi web menerima notifikasi bahwa *rule* terpenuhi	Terpenuhi
	Aplikasi Android menerima notifikasi bahwa *rule* terpenuhi	Terpenuhi

Tabel 5.17: Hasil uji coba evaluasi *rule*

Rule Expression	Data Rekaman Sensor	Hasil Harapan	Hasil
Raspberry Pi == 1	Raspberry Pi: 1	Benar	Benar
Raspberry Pi == 1 AND Sensor 2 > 20	Raspberry Pi: 1 Sensor 2: 17	Salah	Salah
Sensor 3 > 40 OR Sensor 4 < 50	Sensor 3: 45 Sensor 4: 55	Benar	Benar
Raspberry Pi == 1 OR Sensor 2 > 20 OR Sensor 3 < 40	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45	Benar	Benar
Sensor 2 > 20 AND Sensor 3 > 40 OR Sensor 4 >= 60	Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah	Salah
Raspberry Pi == 1 AND Sensor 3 >= 37 AND Sensor 4 > 52	Raspberry Pi: 1 Sensor 3: 45 Sensor 4: 55	Benar	Benar
Raspberry Pi == 0 OR Sensor 2 >= 25 AND Sensor 3 > 37 AND Sensor 4 >= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah	Salah
Raspberry Pi == 0 AND Sensor 2 <= 25 OR Sensor 3 > 37 AND Sensor 4 <= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah	Salah
Raspberry Pi == 1 AND Sensor 2 > 25 OR Sensor 3 >= 37 AND Sensor 4 >= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Benar	Benar
Raspberry Pi == 0 AND Sensor 2 < 25 OR Sensor 3 <= 37 AND Sensor 4 <= 52	Raspberry Pi: 1 Sensor 2: 17 Sensor 3: 45 Sensor 4: 55	Salah	Salah

5.3.2 Uji Performa

Berdasarkan skenario uji coba pada subbab 5.2.2, pengujian dilakukan pada proses kolaborasi data sensor dan ketika menangani data rekaman sensor. Subbab ini akan membahas

hasil dan evaluasi dari tiap uji coba tersebut.

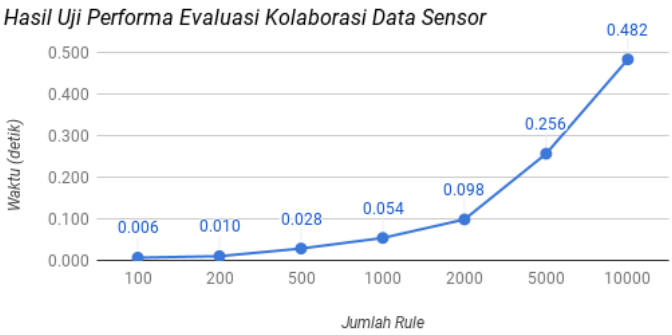
5.3.2.1 Uji Performa Evaluasi Kolaborasi Data Sensor

Terdapat tiga data hasil dari uji performa ini, yaitu kecepatan evaluasi sejumlah *rule*, penggunaan RAM, dan penggunaan CPU yang tertera pada Tabel 5.18.

Tabel 5.18: Hasil uji performa evaluasi kolaborasi data sensor

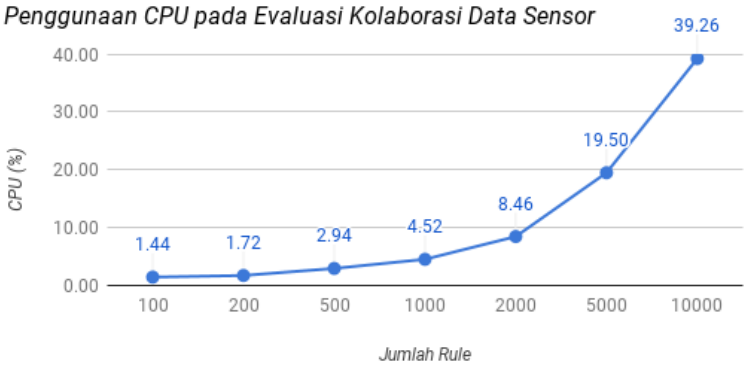
No	Jumlah *Rule*	Waktu	CPU (%)	RAM (MB)
1	100	0.006 detik	1.44%	8.31 MB
2	200	0.010 detik	1.72%	9.13 MB
3	500	0.028 detik	2.94%	10.94 MB
4	1000	0.054 detik	4.52%	11.5 MB
5	2000	0.098 detik	8.46%	12.35 MB
6	5000	0.256 detik	19.5%	12.82 MB
7	10000	0.482 detik	39.26%	13.2 MB

Berdasarkan hasil uji performa pada Tabel 5.18, sistem mampu memproses 1000 *rule* hanya dalam waktu 0.054 detik. Grafik hasil uji performa kecepatan tertera pada Gambar 5.2.



Gambar 5.2: Hasil uji kecepatan evaluasi *rule*

Hasil uji coba performa kedua yaitu penggunaan sumber daya CPU oleh sistem. Berdasarkan Tabel 5.18, sumber daya CPU telah digunakan sebesar 4.52% ketika memproses 1000 *rule*. Begitu juga dengan penggunaan sumber daya RAM. Sistem hanya menggunakan 11.5 MB RAM ketika memproses 1000 *rule*. Hal ini menunjukkan bahwa kecepatan pemrosesan *rule* dipengaruhi oleh jumlah *rule*. Grafik dari hasil uji penggunaan sumber daya CPU tertera pada Gambar 5.3.



Gambar 5.3: Hasil uji performa evaluasi *rule* (Penggunaan CPU)

5.3.2.2 Uji Performa AutoAI

Berdasarkan skenario uji performa AutoAI, terdapat tiga hasil uji coba, yaitu waktu rata-rata evaluasi *rule* ketika mendapatkan sejumlah data rekaman sensor, penggunaan sumber daya RAM, penggunaan CPU, dan penggunaan *bandwidth* yang tertera pada Tabel 5.19.

Tabel 5.19: Hasil uji performa AutoAI

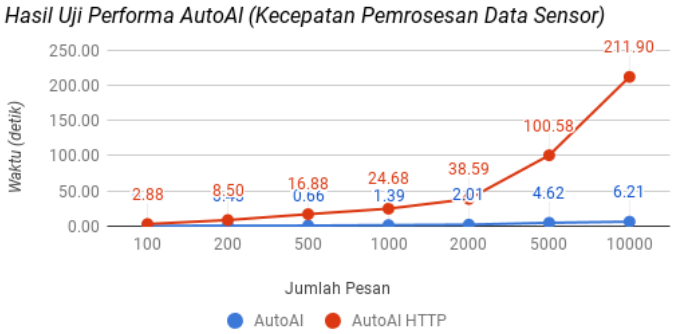
No	Jumlah Pesan	Waktu	CPU (%)	RAM (MB)	*Bandwidth (KB)*
1	100	0.35 detik	8.42%	12.58 MB	9 KB
2	200	0.43 detik	13.18%	13.62 MB	17 KB
3	500	0.66 detik	26.96%	17.66 MB	39 KB
4	1000	1.39 detik	32.10%	23.12 MB	76 KB
5	2000	2.01 detik	59.18%	32.64 MB	148 KB
6	5000	4.62 detik	76.04%	58.72 MB	369 KB
7	10000	6.21 detik	132%	104.62 MB	737 KB

Lalu, berdasarkan skenario, terdapat sistem pembanding yang melakukan proses seperti AutoAI yang dibuat menggunakan protokol HTTP dan bahasa Python. Hasil dari uji coba sistem tersebut tertera pada Tabel 5.20.

Tabel 5.20: Hasil uji performa sistem pembanding AutoAI

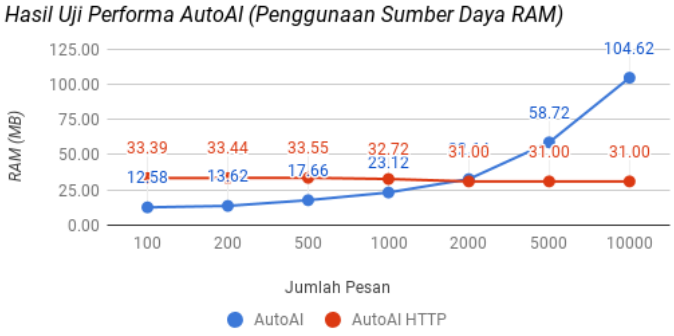
No	Jumlah Pesan	Waktu	CPU (%)	RAM (MB)	*Bandwidth (KB)*
1	100	2.88 detik	16.43%	33.39 MB	55 KB
2	200	8.50 detik	17.05%	33.44 MB	111 KB
3	500	16.88 detik	18.10%	33.55 MB	276 KB
4	1000	24.68 detik	25.73%	32.72 MB	553 KB
5	2000	38.59 detik	26.20%	31.00 MB	1096 KB
6	5000	100.58 detik	26.70%	31.00 MB	2754 KB
7	10000	211.90 detik	26.40%	31.00 MB	5638 KB

Grafik perbandingan antara AutoAI dengan sistem pembanding tertera pada Gambar 5.4 (kecepatan pemrosesan data sensor), Gambar 5.5 (penggunaan CPU), Gambar 5.6 (penggunaan RAM), dan Gambar 5.7 (penggunaan *bandwidth*).



Gambar 5.4: Perbandingan kecepatan AutoAI dengan AutoAI versi HTTP

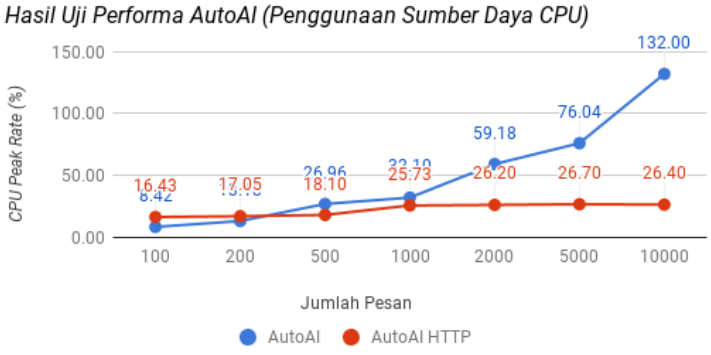
Gambar 5.4 menunjukkan bahwa terdapat perbedaan kecepatan yang sangat signifikan antara AutoAI dengan sistem pembanding AutoAI. AutoAI dapat memproses data rekaman sensor dengan sangat cepat karena menggunakan fitur pemrograman paralel bahasa Go, yaitu `goroutine`. Dengan menggunakan `goroutine`, sistem tidak perlu menunggu proses yang tidak saling berkaitan, sehingga proses dapat dijalankan secara bersamaan.



Gambar 5.5: Perbandingan penggunaan RAM AutoAI dengan HTTP AutoAI

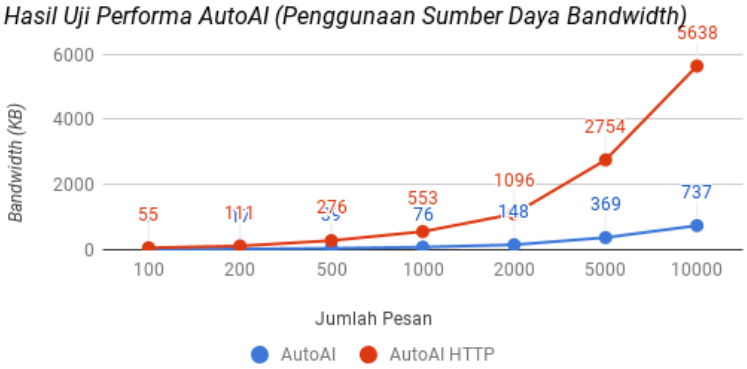
Gambar 5.5 menunjukkan bahwa penggunaan RAM pada AutoAI cenderung rendah hingga pesan yang dikirim secara bersamaan berjumlah 2000. Penggunaan RAM yang bertambah seiring bertambahnya jumlah pesan disebabkan oleh semakin banyaknya proses paralel yang berjalan secara bersamaan, sehingga membutuhkan sumber daya RAM tiap proses paralel. Berbeda dengan HTTP AutoAI yang hanya menggunakan satu proses, sehingga penggunaan RAM cenderung konstan.

Tetapi, terdapat anomali pada saat pesan yang dikirim secara bersamaan berjumlah 5000. RAM yang dibutuhkan sangat tinggi, seharusnya ketika proses paralel selesai, Go akan secara otomatis menghapus memori yang tidak terpakai. Hal ini disebabkan oleh kapasitas koneksi pada RDBMS *postgresql*. Pada konfigurasi normal, *postgresql* hanya dapat menangani maksimal 100 koneksi untuk menjalankan *query*. Sedangkan data rekaman sensor yang diterima oleh sistem mencapai 5000. Sehingga ketika seluruh koneksi dipakai oleh `goroutine` untuk menjalankan *query*, `goroutine` lainnya perlu menunggu. Hal ini menyebabkan *memory leak* hingga `goroutine` tersebut mendapatkan koneksi dan menyelesaikan fungsinya.



Gambar 5.6: Perbandingan penggunaan CPU AutoAI dengan HTTP AutoAI

Gambar 5.6 juga menunjukkan grafik penggunaan sumber daya CPU yang relatif sama dengan penggunaan RAM. Hal ini juga disebabkan oleh semakin banyaknya proses paralel yang berjalan secara bersamaan, sehingga membutuhkan sumber daya CPU untuk menjalankan fungsi tersebut.



Gambar 5.7: Perbandingan penggunaan *bandwidth* AutoAI dengan HTTP AutoAI

Gambar 5.7 menunjukkan bahwa penggunaan *bandwidth* menggunakan metode komunikasi *publish/subscribe* berbasis *lightweight messaging protocol*/MQTT jauh lebih efisien dibandingkan menggunakan metode *client/server* dengan protokol HTTP. Protokol MQTT menggunakan *bandwidth* yang sangat rendah, yakni hanya menggunakan 737 KB untuk mengirim 10000 pesan. Hal itu karena spesifikasi MQTT yang memanfaatkan tiap *bit* pada *header* dengan sangat efisien dan hanya melakukan *handshake* sebanyak satu kali untuk membuat koneksi antara MQTT *broker* dengan *subscriber*. Lalu ketika *publisher* mengirim pesan, MQTT *broker* hanya perlu mengirim pesan tanpa melakukan *handshake* lagi[26].

BAB VI

KESIMPULAN DAN SARAN

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. *Monitoring* data sensor pada lingkungan *cross platform* secara *real time* dilakukan dengan memanfaatkan mekanisme *publish/subscribe* yang disediakan oleh protokol MQTT. *Node* sensor berfungsi sebagai *publisher* dan aplikasi pada *platform* Android dan Web berfungsi sebagai *subscriber*. Selain itu, dibutuhkan perantara yang berfungsi untuk meneruskan data dari *publisher* ke *subscriber*, yaitu MQTT *broker*.
2. Evaluasi kolaborasi data sensor dilakukan dengan melakukan *subscription* pada topik dipublikasikannya data *node* sensor untuk menerima data rekaman sensor secara *real time*, lalu melakukan evaluasi *rule* tiap sensor menggunakan *rule engine* berdasarkan data yang diterima.
3. Sistem mampu menangani 1000 data yang dikirimkan oleh sensor tiap detiknya dan memprosesnya untuk melakukan otomatisasi aksi. Sumber daya yang digunakan untuk menangani data tersebut juga relatif rendah, yakni menggunakan RAM sebesar 23.12 MB, CPU sebesar 32.10% dan *bandwidth* sebesar 76 KB.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Menggunakan teknologi basis data yang bervariasi sesuai dengan kebutuhan sistem, agar tidak terjadi *bottleneck* dan *resource leak* pada saat pemrosesan otomatisasi aksi dan penyimpanan data rekaman sensor.
- Implementasi lingkungan *grid* untuk menjalankan komputasi paralel yang dilakukan oleh sistem.
- Aksi yang dilakukan berhubungan dengan *hardware* lainnya sebagai aktuator.
- Implementasi *rule expression* dengan aturan yang deskriptif.
- Menambahkan fitur *login* agar otomatisasi aksi lebih personal.
- Implementasi sekuritas SSL/TLS pada MQTT *broker*.
- Konfigurasi *rule* menggunakan suara.
- Penentuan *rule* secara otomatis menggunakan *machine learning*.

DAFTAR PUSTAKA

[1] D. Evans, "The internet of things," *How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG)*, vol. 1, hal. 1–12, 2011.

[2] C. Perera, A. Zaslavsky, P. Christen, dan D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, hal. 414–454, 2014.

[3] D. Happ, N. Karowski, T. Menzel, V. Handziski, dan A. Wolisz, "Meeting IoT platform requirements with open pub/sub solutions," *Annals of Telecommunications*, hal. 1–12, 2015.

[4] C. Ferris dan J. Farrell, "What are web services?" *Communications of the ACM*, vol. 46, no. 6, hal. 31, 2003.

[5] "Web Services Glossary." [Daring]. Tersedia pada: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice. [Diakses: 12-Des-2016].

[6] C. A. Gutwin, M. Lippold, dan T. C. Graham, "Real-time groupware in the browser: testing the performance of web-based networking," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. ACM, 2011, hal. 167–176.

[7] R. Rajkumar, M. Gagliardi, dan L. Sha, "The real-time publisher/subscriber inter-process communication model for distributed real-time systems: design and implementation," in *Real-Time Technology and Applications Symposium, 1995. Proceedings*. IEEE, 1995, hal. 66–75.

[8] S. Oh, J.-H. Kim, dan G. Fox, "Real-time performance analysis for publish/subscribe systems," *Future Generation Computer Systems*, vol. 26, no. 3, hal. 318–323, 2010.

[9] D. Locke, "Mq telemetry transport (mqtt) v3. 1 protocol specification," *IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html*, 2010.

[10] C. Doblander, S. Zimmermann, K. Zhang, dan H.-A. Jacobsen, "Demo Abstract: MOS: A Bandwidth-Efficient Cross-Platform Middleware for Publish/Subscribe," in *Proceedings of the Posters and Demos Session of the 17th International Middleware Conference on ZZZ*. ACM, 2016, hal. 27–28.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, dan A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, hal. 114–131, 2003.

[12] "Raspberry Pi FAQs - Frequently Asked Questions." [Daring]. Tersedia pada: https://www.raspberrypi.org/help/faqs/. [Diakses: 4-Mei-2017].

[13] "GPIO: Raspberry Pi Models A and B - Raspberry Pi Documentation." [Daring]. Tersedia pada: https://www.raspberrypi.org/documentation/usage/gpio/. [Diakses: 6-Mei-2017].

[14] "The Go Programming Language." [Daring]. Tersedia pada: https://golang.org/. [Diakses: 16-Mar-2017].

[15] "PostgreSQL: About." [Daring]. Tersedia pada: https://www.postgresql.org/about/. [Diakses: 6-Mei-2017].

[16] "GraphQL." [Daring]. Tersedia pada: http://facebook.github.io/graphql/. [Diakses: 10-Apr-2017].

[17] “Android.” [Daring]. Tersedia pada: https://www.android.com/intl/id_id/. [Diakses: 10-Jun-2017].

[18] “React - A JavaScript library for building user interfaces.” [Daring]. Tersedia pada: https://facebook.github.io/react/. [Diakses: 6-Mei-2017].

[19] “Relay - A JavaScript framework for building data-driven React applications.” [Daring]. Tersedia pada: http://facebook.github.io/relay/. [Diakses: 10-Jun-2017].

[20] “Read Me · Redux.” [Daring]. Tersedia pada: http://redux.js.org/. [Diakses: 10-Jun-2017].

[21] “Ant Design - A UI Design Language.” [Daring]. Tersedia pada: https://ant.design/. [Diakses: 10-Mei-2017].

[22] “General Python FAQ — Python 3.6.1 documentation.” [Daring]. Tersedia pada: https://docs.python.org/3/faq/general.html. [Diakses: 10-Mei-2017].

[23] K. Tang, Y. Wang, H. Liu, Y. Sheng, X. Wang, dan Z. Wei, “Design and implementation of push notification system based on the MQTT protocol,” in *International Conference on Information Science and Computer Applications (ISCA 2013)*, 2013, hal. 116–119.

[24] “back end - definition of back end in English | Oxford Dictionaries.” [Daring]. Tersedia pada: https://en.oxforddictionaries.com/definition/back_end. [Diakses: 6-Mei-2017].

[25] “The Clean Architecture | 8th Light.” [Daring]. Tersedia pada: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html. [Diakses: 6-Mei-2017].

[26] Henrik Sjöstrand, “Low Latency Mobile Messaging using MQTT,” May 2013. [Daring]. Tersedia pada: https://www.slideshare.net/henriksjostrand/devmobile-2013-low-latencymessagingusingmqtt. [Diakses: 9-Jun-2017].

LAMPIRAN A

UJI COBA

1.1 Aksi Push Notification

Berikut kode untuk mengimplementasikan *push notification*.

```
import (
  "time"
  "encoding/json"

  MQTT "github.com/eclipse/paho.mqtt.golang"
  "colsys-backend/pkg/domain"
)

var client MQTT.Client
func init() {
  connOpts := &MQTT.ClientOptions{
    MaxReconnectInterval: 1 * time.Second,
    KeepAlive: 0,
  }

  connOpts.AddBroker(brokerAddr)
  client = MQTT.NewClient(connOpts)
  if token := client.Connect(); token.Wait() && token.Error() !=
       nil {
    panic(token.Error())
  }
}

func pushNotif(topic string, payload []byte) {
  client.Publish(topic, byte(0), false, payload)
}

func PushNotif(invokedRule *domain.InvokedRule) {
  notifData := map[string]interface{}{"rule": invokedRule.Rule.Name
       , "data": invokedRule.Data}
  payload, _ := json.Marshal(notifData)
  pushNotif("notification/1", payload)
}
```

Kode Sumber 1.1: Implementasi *push notification*

1.2 AutoAI menggunakan protokol HTTP

Berikut kode untuk mengimplementasikan AutoAI menggunakan protokol HTTP.

```
1  from flask import Flask, request, jsonify, g
2  from models import db
3  from models import SensorData, InvokedRule
4  import json
5  import time
6  app = Flask(__name__)
7
8  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres@10
       .151.32.111/colsys'
9  db.init_app(app)
10
11 @app.route("/post", methods = ['POST'])
12 def post():
13     reqjson = request.json
14     data = jsonify(reqjson)
15     print reqjson
16     sd = SensorData(reqjson['sensorid'], reqjson['val'], reqjson['
           time'])
17     db.session.add(sd)
18     db.session.commit()
19     jsondata = json.dumps({"s1": reqjson['val'], "s2": "false", "s3
           ": "false", "s4": "false"})
20     ir = InvokedRule(1, jsondata)
21     db.session.add(ir)
22     db.session.commit()
23     return data
24
25 if __name__ == "__main__":
26     app.run(host='0.0.0.0')
```

Kode Sumber 1.2: Implementasi HTTP AutoAI

BIODATA PENULIS

Muhamad Luthfie La Roeha, akrab dipanggil Lutfi lahir di Bogor pada tanggal 19 Maret 1995. Penulis adalah anak kedua dari tiga bersaudara. Penulis menempuh pendidikan formal di SDN Pabuaran 7 Cibinong, SMPN 1 Cibinong, SMKN 1 Cibinong dan S1 Teknik Informatika FTIF ITS. Selama menempuh pendidikan di kampus, penulis berpartisipasi dalam organisasi Himpunan Mahasiswa Teknik Computer-Informatika. Penulis juga pernah menjadi asisten praktikum pada mata kuliah Sistem Operasi dan Jaringan Komputer. Penulis memiliki ketertarikan yang sangat tinggi terhadap perkembangan teknologi dan dampaknya pada masyarakat luas. Beberapa topik yang sedang di dalami adalah topik terkait aplikasi berbasis Web dan *Internet of Things*. Penulis dapat dihubungi melalui surat elektronik `luthfielaroeha@gmail.com`.