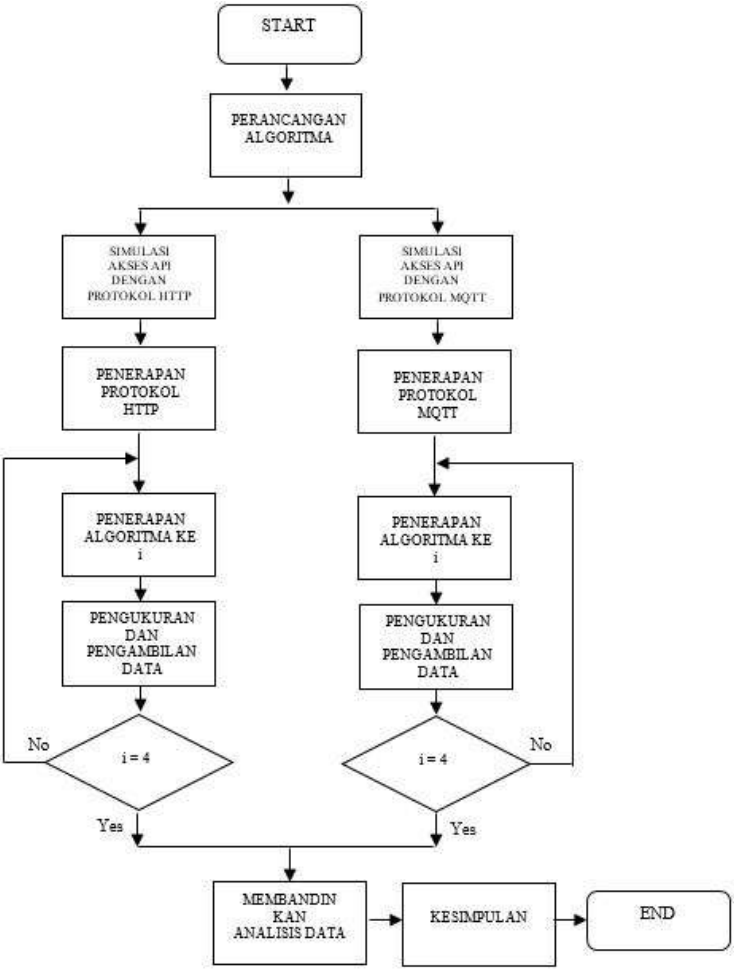
# BAB III
# METODE PENELITIAN

## 3.1. Prosedur Penelitian

Agar tercapainya tujuan dari penelitian ini maka diperlukan kerangka/tahapan dalam setiap langkah penelitian, tahap tersebut direpresentasikan menjadi kerangka tahapan menggunakan diagram alir seperti pada Gambar 3.1.




Gambar 3.1. *Flowchart* Prosedur Penelitian.

### 3.2. Perangkat Penunjang

Untuk merealisasikan skenario yang telah dirancang, setelah *hardware* siap digunakan adalah:

1. ALDEBARAN
2. FTDI *Serial USB module*
3. Multimeter Digital SANWA CD800a
4. Asus TP300LD *Intel Core i3-4030U* CPU *@1.90 GHz*

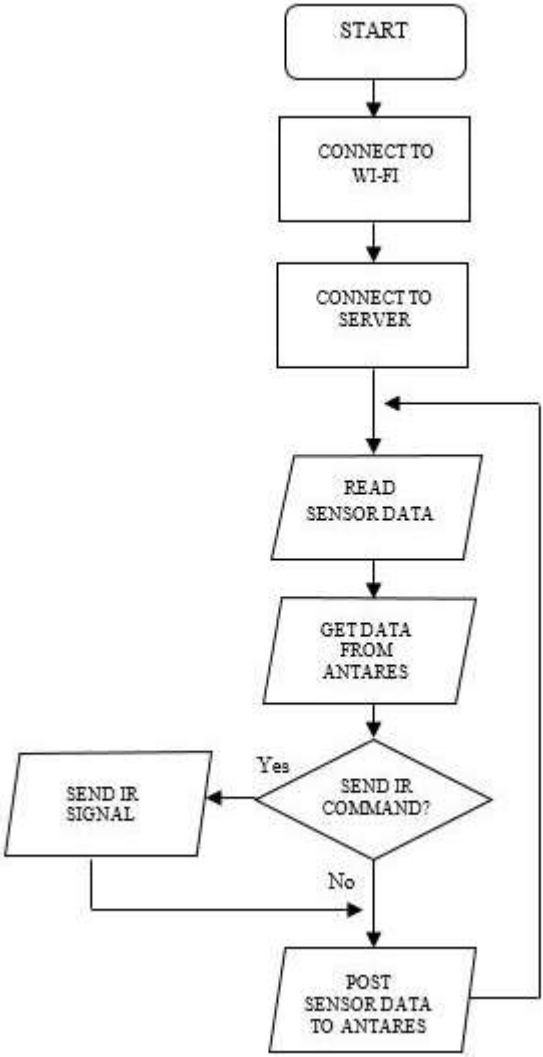
perangkat lunak yang digunakan sebagai berikut:
1. Windows 10 *Professional Edition 64 bit Operating System*
2. Arduino IDE
3. POSTMAN
4. MQTT Fx
5. Chrome Browser

### 3.3. Perancangan Algoritma

Pada penelitian ini dirancang beberapa algoritma pengiriman data yang akan diterapkan pada *running mode* ALDEBARAN. Algoritma ini terbagi menjadi beberapa tahapan, mulai dari pembacaan data dari sensor, terhubung dengan Wi-Fi, membangun koneksi dengan *server*, GET data dari *server*, pengambilan keputusan sistem kontrol IR, dan POST data ke *server*. Beberapa tahapan tersebut dibuat menjadi beberapa rancangan algoritma yang kemudian akan diterapkan dan dibandingkan. Tujuan dari perancangan algoritma ini adalah mendapatkan rancangan algoritma yang paling hemat dalam penggunaan daya jika diterapkan pada ALDEBARAN. Algoritma yang dirancang ini dapat diterapkan pada perangkat yang menggunakan konektivitas Wi-Fi seperti ESP8266 dan mikrokontroler lainnya yang dilengkapi *Wi-Fi Shield*.

#### 3.3.1. *Continuous Data Transmit Algorithm*

Algoritma ini menjelaskan tahapan pengiriman dan pengambilan data secara kontinu tanpa *interval delay* yang digunakan untuk komunikasi data *real time*. ALDEBARAN menerapkan algoritma ini untuk mendukung fungsi kontrol AC melalui komunikasi *infrared*. Algoritma pengiriman dan pengambilan data kontinu dapat dilihat pada Gambar 3.2.




Gambar 3.2. *Flowchart Continuous Data Transmit Algorithm*.

*Flowchart* pada Gambar 3.2 menunjukkan beberapa tahapan diantaranya terhubung dengan Wi-Fi, pembacaan data sensor, membangun koneksi dengan *server*, mengambil data perintah kendali AC dari *server*, pengambilan keputusan fungsi kontrol AC, dan mengirim data sensor ke *server*.

Pada tahap terhubung dengan Wi-Fi, ALDEBARAN mencoba memasuki SSID dari Wi-Fi yang di tuju. Setelah perangkat mendapatkan koneksi internet dari Wi-Fi, maka perangkat akan mencoba terhubung dengan *server*. Dalam penelitian ini *server* yang digunakan sebagai *host* adalah platform.antares.id dengan *port* 8080 untuk HTTP dan 1883 untuk MQTT. Perangkat membaca data sensor berupa nilai suhu dan kelembaban ruangan yang ditampung pada *variable* dengan tipe data float. Setelah terhubung dengan *server*, maka ALDEBARAN memperoleh data dari *server* dengan mengakses API pada IoT ANTARES *platform* menggunakan HTTP *request* dengan metode GET. Data tersebut berisi perintah kendali AC yang dikirim *oleh smartphone* dan data yang diterima oleh ALDEBARAN berupa data yang berisi *header* dan *body* dengan format JSON/XML. Data yang diterima kemudian diuraikan hingga didapatkan perintah kendali AC.

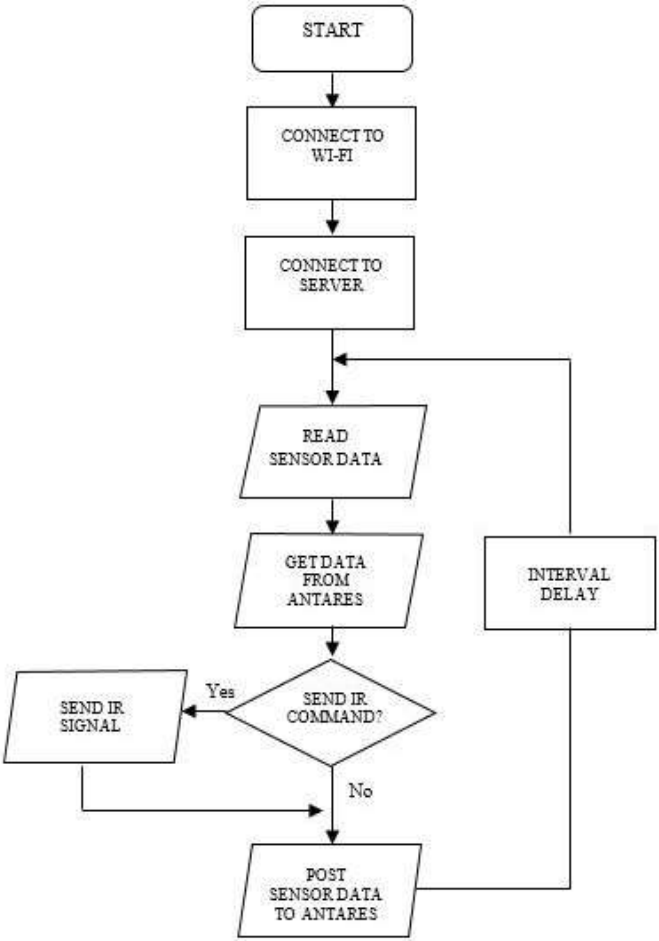
Pada tahap pengambilan keputusan fungsi kontrol, perintah kontrol diterima kemudian dijadikan dasar pengambilan keputusan untuk mengirim sinyal *infrared.* Tahap terakhir dari algoritma ini adalah mengirim data ke *server* dengan mengakses API IoT ANTARES *platform*. Data yang dikirim berisi data sensor yang diperoleh sebelumnya. Selanjutnya program akan melakukan kembali pembacaan data sensor dan mengulangi proses pengiriman dan pengambilan data secara terus menerus sampai perangkat dimatikan.

Waktu untuk melakukan satu siklus perulangan tergantung dari waktu terhubung ke Wi-Fi, membangun koneksi dengan *server*, pembacaan data sensor, pengambilan data dari *server*, dan pengiriman data ke *server*. Jumlah keseluruhan waktu tersebut menjadi waktu siklus sistem dari algoritma ini.

### 3.3.2. *Periodic Data Transmit Algorithm*

Algoritma ini menjelaskan tahapan pengiriman dan pengambilan data secara periodik dengan *interval delay* yang digunakan untuk komunikasi data yang tidak *real time*. ALDEBARAN menerapkan

algoritma ini untuk mendukung fungsi kontrol AC melalui komunikasi *infrared.* Algoritma pengiriman dan pengambilan data periodik dapat dilihat pada Gambar 3.3.




Gambar 3.3. *Flowchart Periodic Data Transmit Algorithm*.

*Flowchart* pada Gambar 3.3. menunjukkan beberapa tahapan yang sama dengan algoritma pada Gambar 3.2. Perbedaannya, pada algoritma ini terdapat *interval delay* yang dapat di atur ketika ALDEBARAN berada pada *config mode*. Pada saat *delay* terjadi, maka status Wi-Fi dalam keadaan *idle*. Selanjutnya program akan melakukan kembali pembacaan data sensor secara periodik sampai perangkat dimatikan.

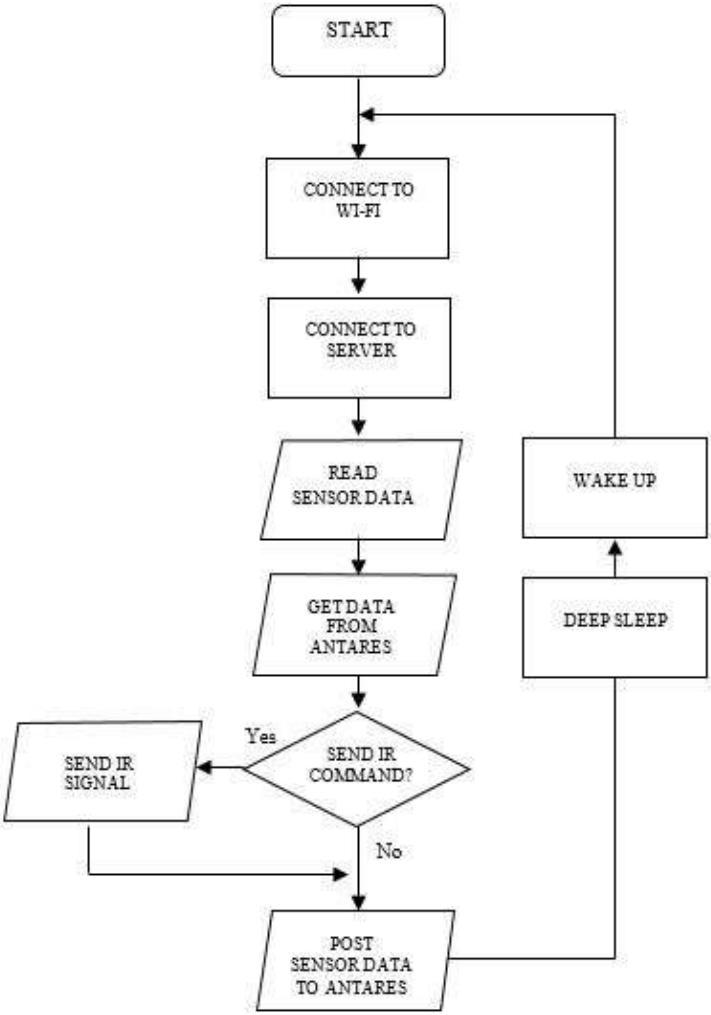
Waktu untuk melakukan satu siklus perulangan tergantung dari waktu terhubung ke Wi-Fi, membangun koneksi dengan *server*, pembacaan data sensor, pengambilan data dari *server*, pengiriman data ke *server*, dan *interval delay*. Jumlah keseluruhan waktu tersebut menjadi waktu siklus sistem dari algoritma ini.

### 3.3.3. *Periodic Data Transmit Algorithm with Deep sleep Mode*

Algoritma ini menjelaskan tahapan pengiriman dan pengambilan data secara periodik dengan *interval delay* menggunakan RTC pada mode *deep sleep* dan digunakan untuk komunikasi data yang tidak *real time*. ALDEBARAN menerapkan algoritma ini untuk mendukung fungsi kontrol AC melalui komunikasi *infrared.* Algoritma pengiriman dan pengambilan data periodik dengan mode *deep sleep* dapat dilihat pada Gambar 3.4.

*Flowchart* pada Gambar 3.4 menunjukkan beberapa tahapan yang sama dengan algoritma pada Gambar 3.3. Perbedaannya, *interval delay* disini adalah waktu untuk berada pada mode *deep sleep*. Pada saat *deep sleep* terjadi, maka status Wi-Fi dan CPU dalam keadaan mati. Setelah waktu hitung mundur dari *interval delay* yang ditentukan selesai, maka selanjutnya perangkat akan bangun dari *sleep mode* dalam keadaan Wi-Fi mati. Perangkat perlu mengulangi langkah terhubung dengan Wi-Fi untuk dapat terhubung kembali ke internet. Kemudian program akan melakukan kembali pembacaan data sensor secara periodik dan mengulang proses *sleep* sampai perangkat dimatikan.

Waktu untuk melakukan satu siklus perulangan tergantung dari waktu terhubung ke Wi-Fi, membangun koneksi dengan *server*, pembacaan data sensor, pengambilan data dari *server*, pengiriman data ke *server*, dan *sleep mode*. Jumlah keseluruhan waktu tersebut menjadi waktu siklus sistem dari algoritma ini.



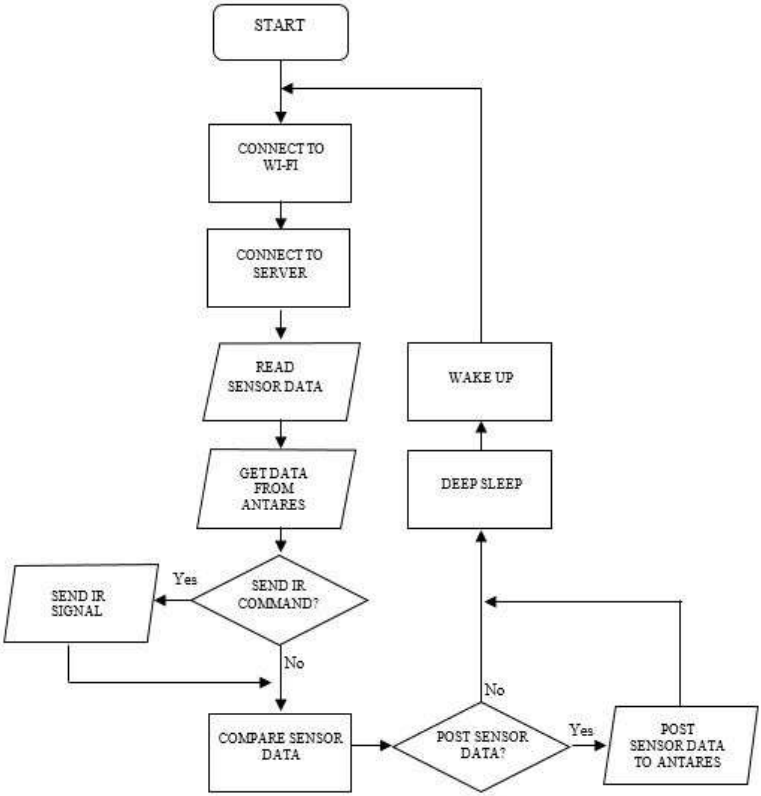

Gambar 3.4. *Flowchart Periodic Data Transmit Algorithm with Deep Sleep Mode*.

### 3.3.4. *Adaptive Data Transmit Algorithm*

Algoritma ini menjelaskan tahapan pengambilan data secara periodik dengan *interval delay* menggunakan RTC pada mode *deep sleep* dan tahapan pengiriman data secara adaptif yang digunakan untuk komunikasi data yang tidak *real time*. ALDEBARAN menerapkan algoritma ini untuk mendukung fungsi kontrol AC melalui komunikasi *infrared*. Algoritma pengiriman dan pengambilan data adaptif dapat dilihat pada Gambar 3.5.

*Flowchart* pada Gambar 3.5 menunjukkan beberapa tahapan diantaranya terhubung dengan Wi-Fi, pembacaan data sensor, membangun koneksi dengan *server*, mengambil data perintah kendali AC dari *server*, pengambilan keputusan fungsi kontrol AC, membandingkan data sensor sekarang dengan data sebelumnya, pengambilan keputusan pengiriman data sensor ke *server*, *interval delay* menggunakan *deep sleep*, dan *wake up* dari mode *sleep*.

Sampai dengan tahap pengambilan keputusan fungsi kontrol, algoritma ini masih sama dengan algoritma pada Gambar 3.2. Tahap selanjutnya adalah membandingkan data sensor yang dikirim ke *server* pada siklus *deep sleep* sebelumnya dengan data sensor yang baru terbaca. Data sensor yang dikirim ke *server* pada siklus sebelumnya dijadikan *threshold* pengambilan keputusan. *Threshold* terdiri dari *threshold* atas dan bawah yang ditentukan dengan menjumlahkan ± deviasi *threshold* pada nilai yang terbaca sebelumnya. Deviasi *threshold* dapat diatur pada menu *system config*. Apabila data yang baru masih dalam rentang *threshold*, maka data tidak akan dikirim ke *server* dan perangkat langsung masuk dalam mode *deep sleep*. Jika data yang baru diluar rentang *threshold*, maka *threshold* diperbarui dan data dikirim ke *server* kemudian perangkat masuk dalam mode *deep sleep*. Pada saat *deep sleep* terjadi, maka status Wi-Fi dan CPU dalam keadaan mati. Setelah waktu hitung mundur dari *interval delay* yang ditentukan selesai, maka selanjutnya perangkat akan bangun dari *sleep mode* dalam keadaan Wi-Fi mati. Perangkat perlu mengulangi langkah terhubung dengan Wi-Fi untuk dapat terhubung kembali ke internet. Kemudian program akan melakukan kembali pembacaan data sensor secara periodik dan mengulang proses *sleep* sampai perangkat dimatikan.




Gambar 3.5. *Flowchart Adaptive Data Transmit Algorithm*.

Waktu untuk melakukan satu siklus perulangan tergantung dari waktu terhubung ke Wi-Fi, membangun koneksi dengan *server*, pembacaan data sensor, pengambilan data dari *server*, pengiriman data ke *server*, dan *sleep mode*. Jumlah keseluruhan waktu tersebut menjadi waktu siklus sistem dari algoritma ini.

### 3.4. Penerapan Algoritma dan Protokol

Algoritma pengiriman dan pengambilan data yang telah dirancang diterapkan pada algoritma ALDEBARAN yang sebelumnya sudah ada. Menerapkan algoritma tersebut cukup mengubah bagian *running mode*. Algoritma ditulis dalam bahasa pemrograman C++ dengan menggunakan *compiler* Arduino IDE *(Integrated Development Environment)*.

*Compiler* Arduino IDE mendukung banyak tipe mikrokontroler, tidak hanya Arduino *board* saja. ALDEBARAN berbasiskan Mikrokontroler ESP8266 seri ESP12-E juga dapat menggunakan Arduino IDE sebagai *compilernya*. Tampilan UI *(User Interface)* pada *Workspace* Arduino IDE dapat dilihat pada Gambar 3.6.


Gambar 3.6. Tampilan UI *Workspace* Arduino IDE.

Pemrograman dengan bahasa C++ menggunakan Arduino IDE strukturnya sama seperti *compiler* pada umumnya. Pada Arduino IDE terdapat fungsi utama yaitu *void setup* dan fungsi perulangan tak berhingga yaitu *void loop*. Pada *void setup* program dieksekusi sekali sedangkan *void loop* mengeksekusi program berulang kali. Di atas *void setup* adalah bagian pemasukan pustaka/ *library*, bagian deklarasi *variable global* dengan tipe datanya, dan bagian deklarasi *class variable*.

Arduino IDE adalah *compiler open source* yang digunakan oleh komunitas *makers*, sehingga mendukung banyak *library* dan *board MCU*. *Class variable* biasanya dibuat di dalam library berbasiskan OOP *(Object Oriented Programming). Class variable* yang telah ditambahkan melalui *library* harus dideklarasikan baik menjadi *global variable* ataupun *local variable*.

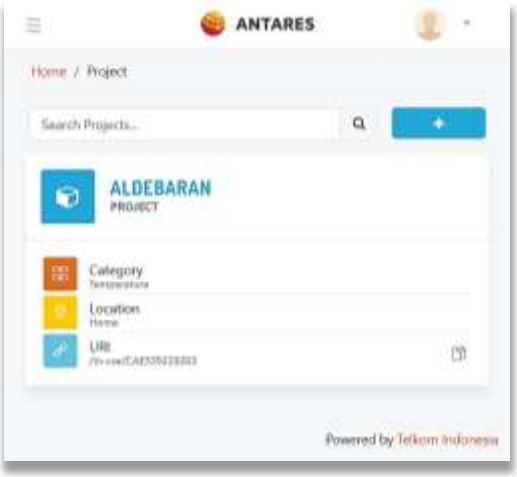
Penerapan protokol HTTP dan MQTT juga ditulis menggunakan *compiler* Arduino IDE. Penerapan protokol ini dapat dilakukan dengan menambahkan *library* pada Arduino IDE. Menerapkan protokol HTTP dapat menggunakan *library* <WiFiClient.h> sedangkan untuk menerapkan protokol MQTT dapat menggunakan library <PubSubClient.h>. *Library* yang digunakan untuk mendukung segala keperluan protokol dapat diakses pada laman https://github.com.

Untuk melakukan *request* ke *server* baik menggunakan protokol HTTP dan MQTT keduanya memiliki format *request* untuk mengakses API *server*. Pada penelitian ini menggunakan IoT ANTARES *platform* sebagai *server* untuk menyimpan data. Format untuk melakukan *request* ke *server* yang ditentukan oleh IoT ANTARES *platform* terdiri dari dua jenis format yaitu XML *(Extensible Markup Languange)* dan JSON *(Java Script Object Notation)*.

IoT ANTARES *platform* digunakan untuk menyimpan data sensor dan perintah kontrol perangkat, serta merupakan *platform* yang cocok untuk *developer* lokal, karena pembuatan akun tidak berbayar dan mengakses API gratis hingga 10.000 akses/hari. IoT ANTARES *platform* dapat diakses dengan memasuki laman *web* http://antares.id. *Dashboard* dari *platform* IoT ANTARES yang diakses melalui *web browser* dapat dilihat pada Gambar 3.7.

Pada *dashboard* ANTARES, pengguna *platform* dapat membuat *project* dan setiap *project* yang dibuat mempunyai URI yang unik. Di dalam sebuah *project*, pengguna dapat membuat beberapa *data container* yang dapat menyimpan berbagai tipe data. *Data container* mempunyai URI yang unik untuk menyimpan data ke *server*, dan format *data access* yang unik untuk mengambil data dari *server*. Setiap *data container* dapat menampilkan sebuah grafik dan sebuah tabel. Apabila data yang dikirim ke *data container* berupa data numerik maka *platform* akan melakukan *parsing* data dan menampilkannya pada grafik. Namun

jika data yang dikirim ke *data container* berupa data berisi rangkaian kata, maka *platform* tetap menerima data tersebut dan akan muncul pada tabel di bawah grafik. Tampilan *data container* pada IoT ANTARES *platform* dapat dilihat pada Gambar 3.8.




Gambar 3.7. *Dashboard* IoT ANTARES *Platform*.


Gambar 3.8. Tampilan *Data Container* pada IoT ANTARES *Platform*.

### 3.4.1. Format Akses API ANTARES Menggunakan Protokol HTTP

Pada HTTP *request* perintah untuk menyimpan data ke IoT ANTARES *platform* menggunakan metode POST mengikuti format yang dapat dilihat pada Tabel 3.1.

Tabel 3.1. Format Menyimpan Data dengan Metode POST.

| Field | Value |
|---|---|
| URL | http://platform.antares.id:8080/~/in-cse/in-name/project_name/device_name<br>or :<br>http://platform.antares.id:8080/~/in-cse/device_uri<br>8443 for Secured Port |
| Method | POST |

| | | |
|---|---|---|
| Header | **Key** | **Value** |
| | X-M2M-Origin | <Access ID>:<Access Password> |
| | Content-Type | application/xml;ty=4 or application/json;ty=4 |
| Body (JSON version) | <pre>{<br>  "m2m:cin": {<br>  "cnf": "message",<br>  "con": "<br>  <obj><br>  <int name=\"data\" val=\"29\"/><br>  <int name=\"unit\" val=\"celsius\"/><br>  </obj><br>  "<br>  }<br>}</pre> | |
| Body (XML version) | <pre><m2m:cin<br>xmlns:m2m="http://www.onem2m.org/xml/protocols"><br><cnf>message</cnf><br><con><br><obj><br><int name="data" val="29"/><br><int name="unit" val="celsius"/><br></obj><br></con><br></m2m:cin></pre> | |

Pada HTTP *request* perintah untuk mengambil data dari IoT ANTARES *platform* menggunakan metode GET mengikuti format yang dapat dilihat pada Tabel 3.2.

Tabel 3.2. Format Mengambil Data Terakhir dengan Metode GET.

| **Field** | **Value** | |
|---|---|---|
| URL | http://platform.antares.id:8080/~/in-cse/in-name/project_name/device_name/la<br>or :<br>http://platform.antares.id:8080/~/in-cse/device_uri/la<br>8443 for Secured Port | |
| Method | GET | |
| Header | **Key** | **Value** |
| | X-M2M-Origin | <Access ID>:<Access Password> |

| | | |
|---|---|---|
| | Content-Type | application/xml or application/json |
| Body | - | |

### 3.4.2. Format Akses API ANTARES Menggunakan Protokol MQTT

Pada MQTT *publish/subscribe*, perintah untuk menyimpan data ke *broker* IoT ANTARES *platform* mengikuti format yang dapat dilihat pada Tabel 3.3 dan 3.4.

Tabel 3.4. Format Menyimpan Data dengan MQTT *Subscribe*.

| Type | Description |
|---|---|
| Response | The aim of the response is to subscribe. The Response topic is: /oneM2M/resp/in-cse/your-access-key/json |
| Response | <pre>{<br>"m2m:rsp" : {<br>"rsc" : 2001,<br>"rqi" : "123456",<br>"pc" : {<br>"m2m:cin" : [ {<br>"rn" : "cin_876977242",<br>"ty" : 4,<br>"ri" : "/in-cse/cin-876977242",<br>"pi" : "/in-cse/cnt-516560931",<br>"ct" : "20171110T051959",<br>"lt" : "20171110T051959",<br>"st" : 0,<br>"cnf" : "message",<br>"cs" : 72,<br>"con" :<br>"<obj><br><int name=\"data\" val=\"30.5\"/><br><int name=\"unit\" val=\"celsius\"/><br></obj>"<br>} ]<br>},<br>"to" : "e7e349fc2216941a:9d0cf82c25277bdd",<br>"fr" : "/in-cse"<br>} }</pre> |

Tabel 3.3. Format Menyimpan Data dengan MQTT *Publish*.

| Type | Description |
|---|---|
| Broker | platform.antares.id:1883 or platform.antares.id:8883 with TLS. |
| Request | The aim of the request is to store data on container. The Request topic is: /oneM2M/req/your-access-key/in-cse/json |

| | |
|---|---|
| | Payload:<br>{<br>"m2m:rqp": {<br>"fr": "access-key",<br>"to": "/in-cse/in-name/project-name/device-name",<br>"op": 1,<br>"rqi": 123456,<br>"pc": {<br>"m2m:cin": {<br>"cnf": "message",<br>"con":<br>"<obj><br><int name=\"data\" val=\"30.5\"/><br><int name=\"unit\" val=\"celsius\"/><br></obj>"<br>}<br>},<br>"ty": 4<br>}<br>} |

Pada MQTT *publish/subscribe*, perintah untuk mendapatkan data terakhir dari *broker* IoT ANTARES *platform* mengikuti format yang dapat dilihat pada Tabel 3.5 dan 3.6.

Tabel 3.5. Format Mengambil Data dengan MQTT *Publish*.

| Type | Description |
|---|---|
| Broker | platform.antares.id:1883 or platform.antares.id:8883 with TLS. |
| Request | The aim of the request is to retrieve data from container. The Request topic is: /oneM2M/req/your-access-key/in-cse/json<br>Payload:<br>{<br>"m2m:rqp" : {<br>"fr": "your-access-key",<br>"to": "/in-cse/in-name/project-name/device-name/la",<br>"op": 1,<br>"rqi": 123456,<br>"ty": 23<br>}<br>} |

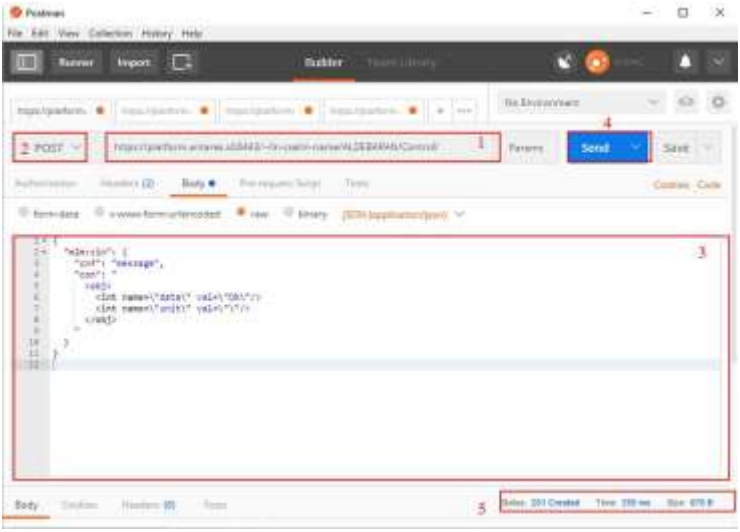
Tabel 3.6. Format Mengambil Data dengan MQTT *Subscribe*.

| Type | Description |
|---|---|
| Response | The aim of the response is to subscribe. The Response topic |

| | |
|---|---|
| | is: /oneM2M/resp/in-cse/your-access-key/json |
| Response | {<br>"m2m:rsp" : {<br>"rsc" : 2000,<br>"rqi" : "123456",<br>"pc" : {<br>"m2m:cin" : [ {<br>"rn" : "cin_526387358",<br>"ty" : 4,<br>"ri" : "/in-cse/cin-526387358",<br>"pi" : "/in-cse/cnt-516560931",<br>"ct" : "20171110T053021",<br>"lt" : "20171110T053021",<br>"st" : 0,<br>"cnf" : "message",<br>"cs" : 72,<br>"con" :<br>"<obj><br><int name=\"data\" val=\"30.5\"/><br><int name=\"unit\" val=\"celsius\"/><br></obj>"<br>} ]<br>},<br>"to" : "e7e349fc2216941a:9d0cf82c25277bdd",<br>"fr" : "/in-cse"<br>}<br>} |

### 3.5. Simulasi Akses API dengan Protokol HTTP

Simulasi dilakukan menggunakan *software* POSTMAN. POSTMAN adalah *software* yang berfungsi sebagai REST (*Representational State Transfer) client*. REST *client* sendiri dapat menggunakan beberapa metode seperti GET, POST, PUT, dan DELETE. REST dilengkapi URIs *(Uniform Resource Identifiers)* yang unik untuk tiap *resources*. REST *client* berjalan menggunakan protokol HTTP yang sering digunakan untuk komunikasi antara *client* dan *server* dan memuat file berformat XML *(Extensible Markup Languange)* atau JSON *(Java Script Object Notation)*. Tampilan UI *software* POSTMAN dapat dilihat pada Gambar 3.9.



Gambar 3.9. Tampilan UI *software* POSTMAN.

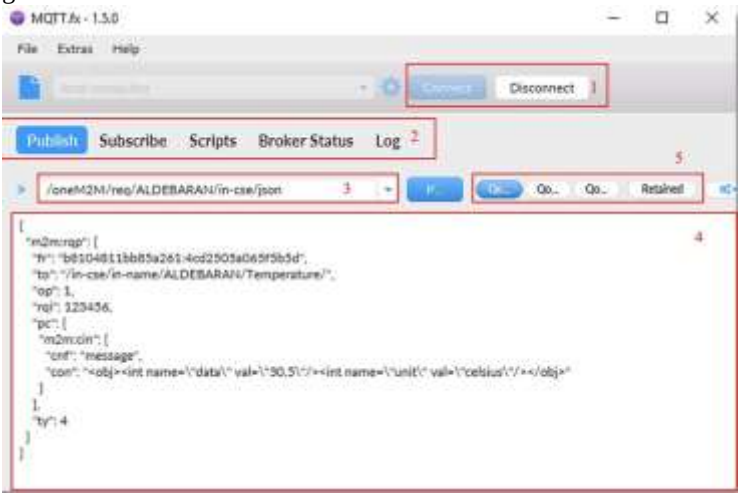
Pada tampilan *software* POSTMAN terlihat bagian-bagian yang ditandai dengan kotak berwarna merah. Kotak bernomor 1 berisi URL *(Uniform Resource Locator) server* yang dituju untuk melakukan *request*. Kotak bernomor 2 adalah pilihan metode HTTP *request* diantaranya GET, POST, PUT dan DELETE yang merupakan fungsi REST API. Kotak bernomor 3 merupakan kolom yang dapat diisi *payload* yang akan dikirim pada bagian *body*, sedangkan pada bagian *header* kolom tersebut berisi *header* yang diperlukan. Kotak bernomor 4 adalah tombol kirim apabila semua data yang dibutuhkan telah terpenuhi. Kotak bernomor 6 adalah *status* pengiriman data yang terdiri dari HTTP *status*, *latency*, dan *payload size*.

### 3.6. Simulasi Akses API dengan Protokol MQTT

Simulasi dilakukan menggunakan *software* MQTT Fx yang berfungsi sebagai *subscriber* dan *publisher* sehingga dapat melakukan *subscribe*/*publish* dengan *topic* yang ditentukan. MQTT Fx dapat terhubung ke *broker* dengan memasukan *broker address*. MQTT Fx

berjalan menggunakan protokol MQTT yang sering digunakan untuk komunikasi antara *subscriber* dengan *broker* maupun *publisher* dengan *broker* yang memuat file berformat XML *(Extensible Markup Languange)* atau JSON *(Java Script Object Notation)*. Tampilan UI *software* MQTT Fx dapat dilihat pada Gambar 3.10.

Pada tampilan UI *software* MQTT Fx terlihat bagian-bagian yang ditandai dengan kotak berwarna merah. Pada kotak bernomor 1 terdapat 3 *icon* yaitu *setting*, *connect* dan *disconnect*. Menu *setting* digunakan untuk menentukan nama profil, alamat *broker*, port *broker*, kemudian setingan koneksi lainnya seperti waktu *time out, credential, SSL/TLS, Proxy, Will.* Setelah menu *setting* terkonfigurasi maka tombol *connect* dapat ditekan untuk terhubung ke *broker* dan tombol *disconnect* untuk memutus hubungan dengan *broker*. Kotak bernomor 2 merupakan fungsi yang dapat dilakukan MQTT Fx diantaranya *publish, subscribe, script*, melihat status *broker*, dan log komunikasi. Kotak bernomor 3 merupakan *topic* yang dapat diisi ketika melakukan *publish* atau *subscribe*. Kotak bernomor 4 merupakan kolom yang dapat diisi *payload* yang akan dikirim. Kotak bernomor 5 adalah pilihan QoS yang digunakan.




Gambar 3.10. Tampilan UI *software* MQTT Fx.

### 3.7. Pengambilan Data

Penelitian ini bertujuan untuk mengoptimalkan komunikasi data, khususnya untuk membuat komunikasi dengan *server* menjadi lebih efisien dan membuat komunikasi data yang hemat daya pada perangkat ALDEBARAN. Agar dapat mencapai tujuan tersebut perlu membandingkan hasil analisis kombinasi algoritma dengan protokol HTTP dan MQTT yang diterapkan pada ALDEBARAN. Implementasi ALDEBARAN dan pengambilan data dilakukan di PT.Telekomunikasi Indonesia, Tbk pada Lab IoT *Research and Development* Divisi *Digital Service* yang beralamat di Jl. Gegerkalong Hilir No 47, Sukarasa, Sukasari, Kota Bandung, Jawa Barat 40152.

### 3.8. Analisis Data

Data yang diperoleh dari penerapan algoritma dan protokol adalah, waktu siklus sistem, jumlah data yang dikirim, dan konsumsi arus, sehingga dapat dianalisis:

1. Perbandingan *payload*, *overhead* dan *throughput* pada protokol HTTP dan MQTT.
2. Perbandingan *latency* pengiriman data pada protokol HTTP dan MQTT.
3. Perbandingan daya transmisi data pada protokol HTTP dan MQTT.
4. Perbandingan kinerja algoritma yang diterapkan dengan protokol HTTP dan MQTT.

Penjelasan dari setiap poin analisis dijelaskan pada BAB 4.