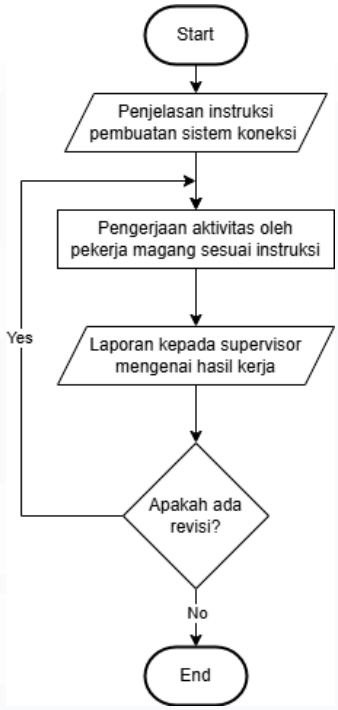
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan kerja magang dilakukan pada kantor PT Patara Teknik Solusindo. Peserta kerja magang memiliki tugas sebagai SCADA *engineer* untuk membantu para staf senior departemen *engineering* dalam mengerjakan proyek-proyek tertentu. Tugas-tugas yang dikerjakan oleh peserta kerja magang dalam membantu pengerjaan proyek berupa melakukan riset yang berkaitan dengan proyek dan pengujian peralatan-peralatan dalam proyek. Dalam melaksanakan tugas kerja magang, peserta didampingi oleh supervisor.

Alur kerja peserta kerja magang dimulai dengan penjelasan oleh supervisor mengenai aktivitas yang akan dilakukan oleh peserta kerja magang beserta dengan contoh pengerjaan aktivitas tersebut. Aktivitas yang dikerjakan oleh penulis berupa pembuatan sistem koneksi untuk sensor-sensor pada mesin *wood chipper* dengan HMI. Setelah penjelasan aktivitas, peserta kerja magang lanjut mengerjakan aktivitas sembari didampingi oleh supervisor, dan diperbolehkan menanyakan hal yang tidak dimengerti dari pengerjaan aktivitas tersebut. Hasil akhir yang dikerjakan oleh peserta kerja magang kemudian diperiksa terlebih dahulu oleh supervisor. Jika ada revisi yang diusulkan oleh supervisor, maka peserta kerja magang memperbaiki sesuai dengan revisi yang diusulkan.



Gambar 3.1 Alur Kerja Magang Penulis

3.2 Tugas dan Uraian Kerja Magang

Dalam kerja magang, berupa 2 kegiatan yang dikerjakan oleh penulis yaitu proyek utama dan tugas keseharian. Sebelum masuk ke dalam proyek utama, tugas keseharian diberikan sebagai gambaran besar tentang apa yang dilakukan dalam keseharian kerja. Tugas-tugas tersebut berupa:

- Pembelajaran mengenai sistem SCADA secara mendasar yang diimplementasikan melalui pembuatan sistem SCADA beserta HMI untuk sebuah gardu induk.
- Membantu pekerjaan staf senior dalam pelaksanaan *Softcomm* panel RTU, pembuatan kabel komunikasi, dan pengantaran barang ke lapangan kerja.

Proyek utama yang diberikan oleh supervisor kepada penulis adalah pembuatan sistem *database* dan koneksi yang terhubung dari sebuah mesin *wood chipper* dengan HMI. Mesin *wood chipper* yang digunakan dalam proyek utama adalah mesin *wood chipper* milik Bruks dengan tipe Microwoodchipper

WH500-1000. Bentuk Microwoodchipper WH500-1000 dapat dilihat pada Gambar 3.2.

Gambar 3.2 Mesin *Wood Chipper* Bermerek Bruks Tipe Microwoodchipper WH500-1000

Dalam pembuatan HMI untuk mesin *wood chipper*, perlu diketahui terlebih dahulu sensor-sensor yang bekerja dalam mesin *wood chipper* tersebut untuk diintegrasikan ke dalam pembuatan sistem *database* dan koneksinya. Berdasarkan spesifikasi dari klien, sensor-sensor yang penting untuk dipantau dalam kerja mesin *wood chipper* adalah sebagai berikut:

- Sensor ampere, yang memiliki fungsi untuk mendeteksi apakah mesin dalam keadaan menyala atau dalam keadaan mati.
- Sensor temperatur, yang memiliki fungsi untuk mendeteksi suhu mesin demi mencegah terjadinya kerusakan mesin oleh karena *overheat*.
- Sensor rotasi motor, yang memiliki fungsi untuk mendeteksi putaran baling-baling pemotong mesin *wood chipper*. Terdapat 2 sensor rotasi motor dikarenakan adanya 2 buah baling-baling pemotong yang bekerja.

Setelah diketahui sensor-sensor yang digunakan dalam mesin *wood chipper*, proses dilanjutkan dengan pembuatan sistem *database* yang dilanjut

dengan pembuatan HMI. Dalam pembuatan hal tersebut, beberapa aplikasi digunakan. Aplikasi-aplikasi yang digunakan berupa:

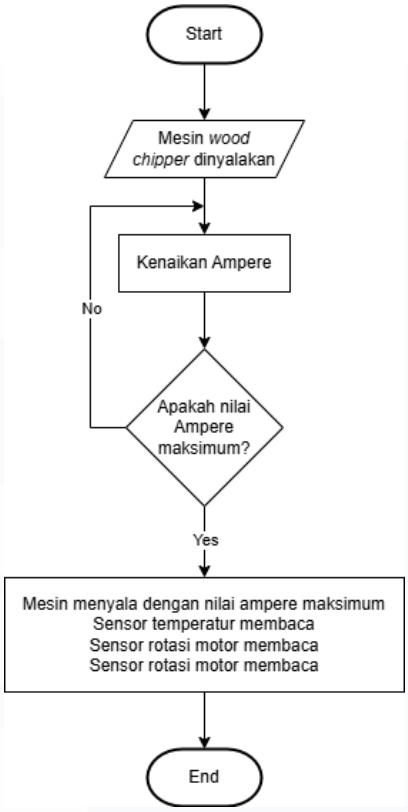
- Aplikasi Node-Red sebagai aplikasi pembuatan algoritma penghasil angka mengacak sebagai simulasi pengganti sensor-sensor pada mesin *wood chipper*.
- Aplikasi InfluxDB sebagai aplikasi pembuatan sistem *database* mesin *wood chipper*.
- Aplikasi Grafana sebagai aplikasi pembuatan HMI mesin *wood chipper*.

Cara kerja sistem dimulai dari aplikasi Node-Red. Di dalam aplikasi Node-Red terdapat algoritma simulasi untuk setiap sensor-sensor yang ada pada mesin *wood chipper*. Output yang dihasilkan dari masing-masing sensor kemudian dikumpulkan ke dalam satu *bucket* di dalam aplikasi InfluxDB. Data-data yang terkumpul dalam aplikasi InfluxDB kemudian ditampilkan oleh HMI pada aplikasi Grafana.

Berikut dijabarkan penjelasan fungsi dari setiap aplikasi yang dipakai:

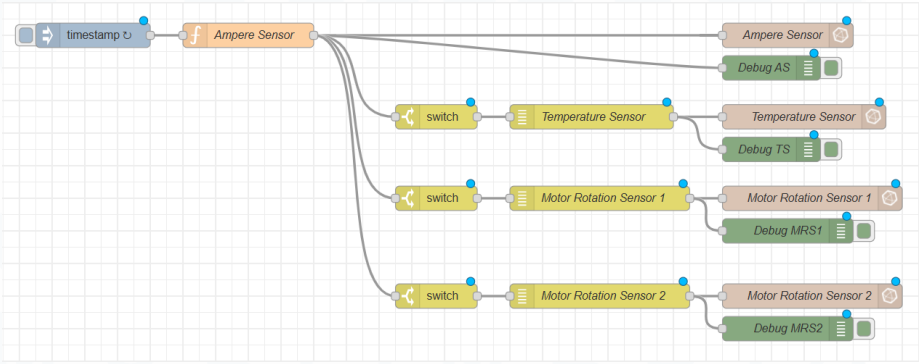
3.2.1. Aplikasi Node-Red

Dalam pembuatan algoritma untuk setiap sensor, perlu diketahui terlebih dahulu sistem kerja sensor dalam mesin *wood chipper*. Ketika mesin *wood chipper* dinyalakan, pembacaan nilai sensor ampere akan naik secara perlahan ketika mesin baru dinyalakan dan akan dalam kondisi tetap di nilai ampere maksimum selama mesin *wood chipper* bekerja. Sensor-sensor lainnya tidak akan mendeteksi nilai sebelum pembacaan sensor ampere sudah mencapai nilai ampere maksimum, dan akan terus mendeteksi nilai selama mesin *wood chipper* bekerja. Sistem kerja sensor dalam mesin *wood chipper* dapat dilihat pada Gambar 3.3 dalam bentuk *flowchart*.



Gambar 3.3 *Flowchart* Penjelasan Sistem Kerja Sensor

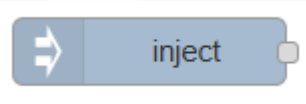
Dengan penjabaran sistem kerja sensor tersebut, algoritma simulasi setiap kerja sensor dapat dibuat dalam aplikasi Node-Red. Hasil pembuatan algoritma simulasi setiap kerja sensor dalam aplikasi Node-Red pada Gambar 3.4.



Gambar 3.4 Algoritma Simulasi Sistem Kerja Mesin dalam Aplikasi Node-Red

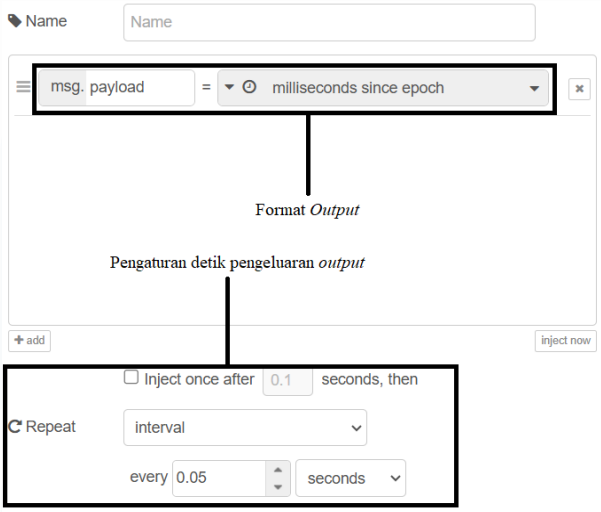
Penjelasan mengenai setiap simbol dalam algoritma tersebut sebagai berikut:

- *Inject Node*



Gambar 3.5 Bentuk *Inject Node*

Inject node memiliki fungsi untuk memberikan interval kepada perintah yang berada pada urutan *flow* berikut-berikutnya. *Flow* algoritma dimulai dengan sebuah *inject node*, yang berfungsi memberikan interval waktu 0.05 detik untuk setiap pengeluaran output pada *node-node* berikutnya. Format output yang dihasilkan berupa variabel "msg.payload", yang berupa format dari pengaturan awal aplikasi Node-Red. Tampilan pengaturan *inject node* dapat dilihat pada Gambar 3.6.



Gambar 3.6 Pengaturan *Inject Node*

- *Function Node*

Gambar 3.7 Bentuk *Function Node*

Function node memiliki fungsi untuk menjalankan sebuah perintah yang dapat dibuat sesuai dengan keinginan pengguna. Setelah *inject node* pada *flow* algoritma, dilanjut oleh sebuah *function node* yang berfungsi sebagai simulasi sensor ampere yang digunakan oleh mesin *wood chipper*. Kode yang bekerja di dalam *function node* tersebut dapat dilihat pada Gambar 3.8.

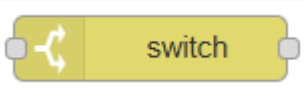
```
const maxCount = 500;

let count = context.get("count") || 0;
msg.power=0;
count++;

if (count >= maxCount) {
    count = maxCount;
    msg.power = 1;
}

context.set("count", count);

msg.payload = count;

return msg;
```

Gambar 3.8 Kode *Function Node*

Kode tersebut berfungsi untuk mensimulasikan mesin *wood chipper* yang menyala dengan terukurnya nilai ampere yang perlahan naik ke nilai maksimal 500 ampere. Ketika sudah mencapai nilai 500 ampere, nilai ampere yang terukur akan konstan di nilai tersebut sampai mesin dimatikan. Pada kode tersebut, dideklarasikan juga sebuah variabel bernama "msg.power". Alasan mengapa dideklarasikan variabel "msg.power" adalah untuk membedakan antara output yang dihasilkan dari pengaturan sistem bawaan. Variabel "msg.power" akan bernilai 0 di awal dan akan berubah

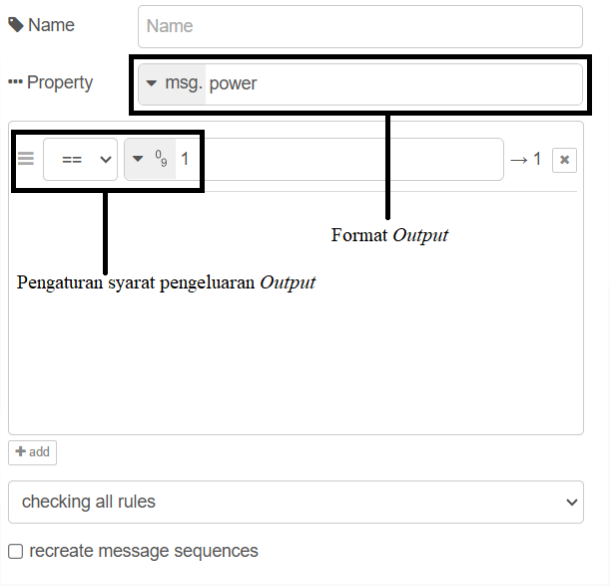
nilainya menjadi 1 ketika sensor ampere sudah mengukur nilai ampere maksimal. Kegunaan variabel tersebut dijelaskan pada ketiga *switch node* yang berada pada *flow* algoritma berikutnya.

- *Switch Node*



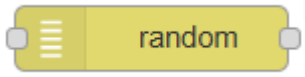
Gambar 3.9 Bentuk *Switch Node*

Switch node memiliki fungsi untuk melanjutkan perintah-perintah pada *node-node* berikutnya setelah memenuhi sebuah syarat yang ditentukan sesuai dengan keinginan pengguna. Kegunaan ketiga *switch node* pada *flow* algoritma adalah menghambat kerja *node-node* setelahnya sebelum syarat yang dideklarasikan terpenuhi. Dalam algoritma tersebut, syarat yang dideklarasikan berupa variabel "msg.power" yang harus bernilai 1 untuk dapat menjalani fungsi-fungsi *node* berikutnya. Hal tersebut dibuat untuk mensimulasikan mesin *wood chipper* yang harus mencapai nilai ampere maksimal terlebih dahulu supaya sensor-sensor lainnya dapat mulai mengukur nilai. Tampilan pengaturan *switch node* dapat dilihat pada Gambar 3.10.



Gambar 3.10 Pengaturan *Switch Node*

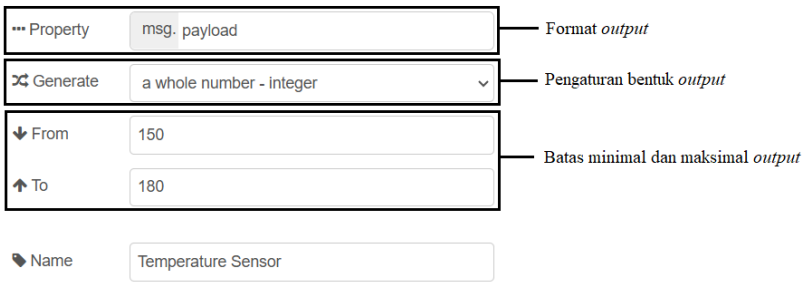
- *Random Node*



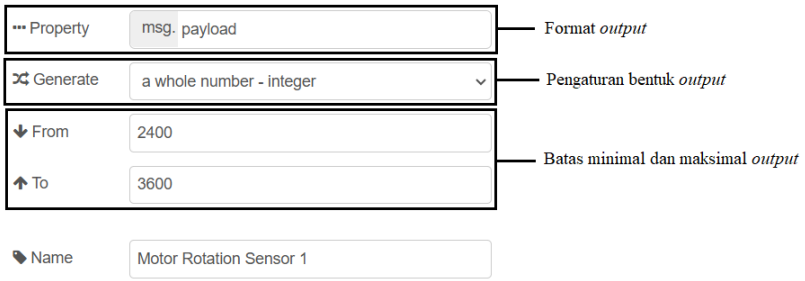
Gambar 3.11 Bentuk *Random Node*

Random node memiliki fungsi untuk menghasilkan sebuah angka mengacak sebagai output, antara dalam bentuk bilangan *integer* (bilangan bulat) atau dalam bentuk bilangan *float* (bilangan desimal). *Flow* algoritma yang tersambung setelah *switch node* dilanjutkan oleh sebuah *random node*. *Random node* berfungsi sebagai simulasi sensor temperatur dan kedua sensor rotasi motor baling-baling pemotong. Untuk sensor temperatur, *random node* diatur untuk menghasilkan bilangan *integer* yang mengacak dari 150 sampai 180 sebagai simulasi pengukuran pada suhu 150 °C sampai 180 °C. Untuk kedua sensor rotasi motor, *random node* diatur untuk menghasilkan bilangan *integer* yang mengacak dari 2400 sampai 3600 sebagai simulasi pengukuran pada kecepatan 2400 rpm sampai

3600 rpm. Tampilan pengaturan setiap *random node* dapat dilihat pada Gambar 3.12 dan Gambar 3.13.

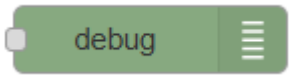


Gambar 3.12 Pengaturan *Random Node* untuk Sensor Temperatur



Gambar 3.13 Pengaturan *Random Node* untuk Sensor Rotasi Motor

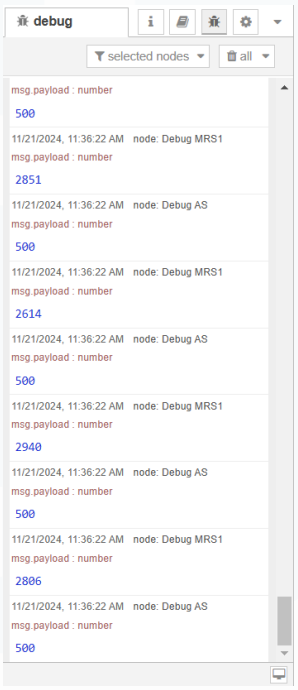
- *Debug Node*



Gambar 3.14 Bentuk *Debug Node*

Node tersebut memiliki fungsi untuk menampilkan output yang dihasilkan dari *node* sebelumnya pada aplikasi Node-Red. *Node* ini biasa dipakai untuk pengembangan algoritma. Setelah setiap *random node* beserta dengan *function node*, dilanjut oleh sebuah *debug node* dan sebuah influxdb *out node*. *Debug node* digunakan dalam

perkembangan algoritma dan perbandingan antara data yang muncul pada aplikasi Node-Red dengan aplikasi berikut-berikutnya. Tampilan *debug screen* yang muncul pada aplikasi Node-Red dapat dilihat pada Gambar 3.15.



Gambar 3.15 Tampilan *Debug Screen* pada Aplikasi Node-Red

- Influxdb *out Node*

Gambar 3.16 Bentuk Influxdb *out Node*

Node tersebut memiliki fungsi untuk menampung output yang dihasilkan dari *node* sebelumnya ke dalam sebuah *bucket* di dalam aplikasi InfluxDB. Untuk influxdb *out node* digunakan untuk menampung *input* yang dihasilkan dari *node* sebelumnya ke dalam sebuah *bucket* pada aplikasi InfluxDB. Cara kerja influxdb *out node*

dalam mengirimkan data ke dalam aplikasi InfluxDB dimulai dengan penulisan server InfluxDB, penulisan *organization*, penulisan *bucket*, dan penulisan nama identifikasi yang dipakai. Sebagai contoh tampilan pengaturan, tampilan pengaturan sensor ampere dalam influxdb *out node* dapat dilihat pada Gambar 3.17.

Gambar 3.17 Pengaturan influxdb *out node*

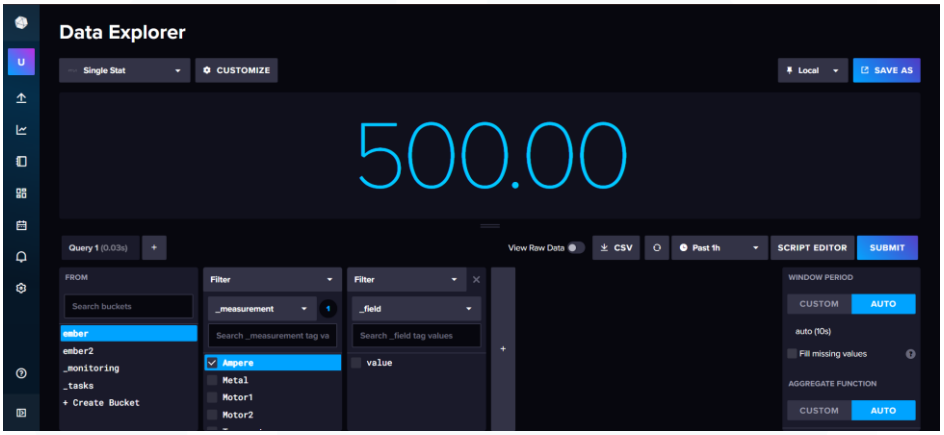
Dalam pengaturan tersebut, penulisan setiap kategori sangat penting untuk mengatur alamat pengiriman datanya. *Organization* ditulis untuk memberikan informasi tentang *account* yang ingin dikirimkan datanya. *Bucket* ditulis untuk memberikan informasi tentang tampungan *database* di dalam *account* yang ingin dikirimkan datanya. Nama identifikasi ditulis untuk memberikan identifikasi tentang *input* yang dikirim kepada *database* InfluxDB.

Setelah pengaturan influxdb *out node* sudah disesuaikan, Node-Red dapat mengirimkan data-datanya ke dalam aplikasi InfluxDB.

3.2.2. Aplikasi InfluxDB

Untuk mengirimkan data-datanya, dilakukan pemrosesan data pada influxdb *out node* untuk dijadikan format data yang sesuai dengan InfluxDB. Data yang sudah diproses kemudian dikirim ke dalam pengaturan alamat yang sudah dijelaskan sebelumnya. Data yang didapatkan kemudian dapat dimunculkan ke dalam aplikasi InfluxDB.

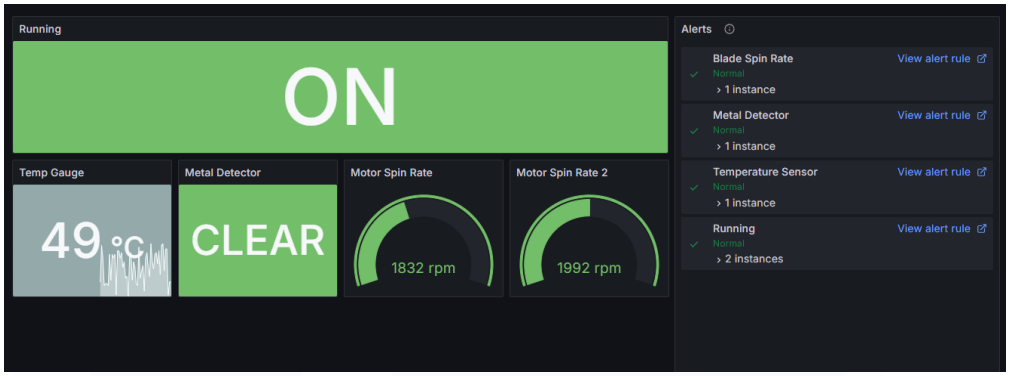
Contoh penampilan data dari sensor ampere yang sudah mencapai nilai ampere maksimum pada aplikasi InfluxDB dapat dilihat pada Gambar 3.18.



Gambar 3.18 Penampilan Data Sensor Ampere Nilai Maksimum pada Aplikasi InfluxDB

3.2.3. Aplikasi Grafana

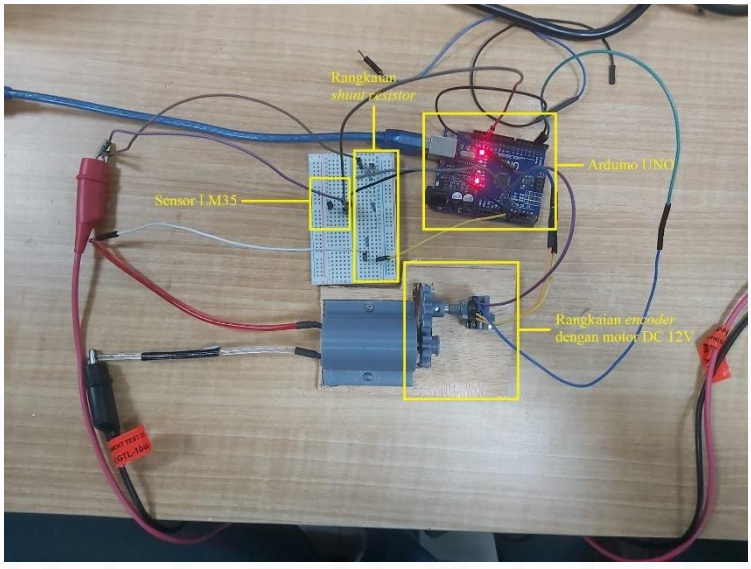
Melalui *database* yang disediakan dari aplikasi InfluxDB tersebut, data-data yang sudah ditampung dapat ditampilkan ke dalam sebuah HMI secara *real-time*. Untuk menyambungkan database dari aplikasi InfluxDB dengan HMI yang dibuat dari aplikasi Grafana, pertama perlu masuk ke dalam opsi *Configuration* > *Data Sources*. *Add data source* ditekan, lalu pilih InfluxDB. Setelah langkah tersebut, dilanjutkan ke dalam pengaturan spesifikasi *Uniform Resource Locator* (URL) yang dipakai, *organization* yang dipakai, dan nama *bucket* yang dipakai. Setelah semua pengaturan sudah diatur, data-data dapat mulai ditampilkan pada HMI dari aplikasi Grafana.



Gambar 3.19 Penampilan HMI Grafana

3.2.4. Simulasi Sistem Menggunakan Miniatur

Selain simulasi menggunakan algoritma penghasil angka mengacak, dibuat juga simulasi pembacaan data melalui perangkat langsung. Perangkat langsung yang digunakan merupakan miniatur dari mesin *wood chipper*, dan dibuat berbasis Arduino. Tampak dari miniatur tersebut dapat dilihat pada Gambar 3.20.



Gambar 3.20 Miniatur Simulasi Kerja Mesin *Wood Chipper*

Miniatur tersebut dilengkapi dengan sebuah rangkaian *shunt resistor* yang berfungsi sebagai simulasi pembacaan nilai ampere, sebuah sensor

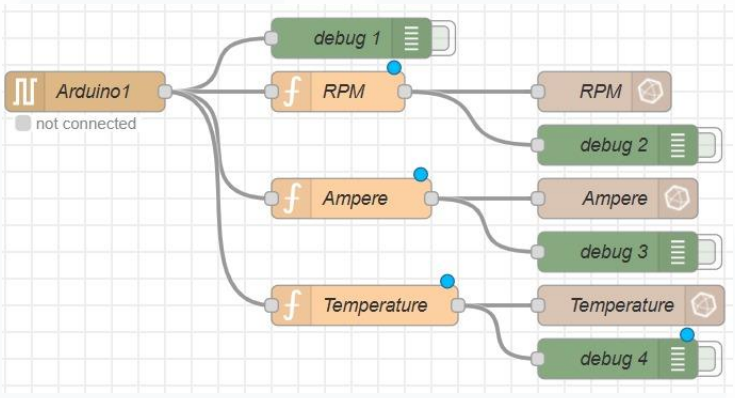
LM35 yang berfungsi sebagai simulasi pembacaan nilai temperatur, dan sebuah rangkaian *encoder* dengan motor DC 12V yang berfungsi sebagai simulasi pembacaan kecepatan rotasi baling-baling pemotong.

Untuk menjalankan setiap fungsi dari miniatur tersebut, digunakan algoritma di dalam aplikasi Arduino yang bertujuan untuk mensimulasikan kerja setiap sensor yang digunakan dalam mesin *wood chipper*. Algoritma yang dibuat untuk mensimulasikan kerja tersebut dapat dilihat pada Gambar 3.21.

```
const int encoderPinA = 2;
const int encoderPinB = 3;

volatile long pulseCount = 0;
unsigned long lastTime = 0;
const int interval = 100;
const int pulsesPerRevolution = 20;

const int currentSensorPin = A0;
const float conversionFactor = 0.05;
const float referenceVoltage = 5.0;
const int adcResolution = 1024;

const int tempSensorPin = A1;

void setup() {
  pinMode(encoderPinA, INPUT_PULLUP);
  pinMode(encoderPinB, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(encoderPinA), countPulses, RISING);

  Serial.begin(9600);
}

void loop() {
  unsigned long currentTime = millis();

  if (currentTime - lastTime >= interval) {
    //RPM
    detachInterrupt(digitalPinToInterrupt(encoderPinA));

    long pulses = pulseCount;
    pulseCount = 0; // Reset pulse count
    attachInterrupt(digitalPinToInterrupt(encoderPinA), countPulses, RISING);

    int rpm = (pulses / (float)pulsesPerRevolution) * (60000.0 / interval);

    Serial.print("RPM: ");
    Serial.println(rpm);

    //Current
    int adcValue = analogRead(currentSensorPin);

    float sensorVoltage = (adcValue / float(adcResolution)) * referenceVoltage;


    float current = sensorVoltage * conversionFactor;

    Serial.print("Current: ");
    Serial.println(current, 3);

    //Temperature
    float temperature;
    int sensorValue = analogRead(tempSensorPin);

    float voltage = sensorValue * (5.0 / 1023.0);

    temperature = voltage * 100.0;

    Serial.print("Temperature: ");
    Serial.println(temperature);



    lastTime = currentTime;
  }
}

void countPulses() {
  if (digitalRead(encoderPinB) == HIGH) {
    pulseCount++; // Clockwise rotation
  } else {
    pulseCount++; // Clockwise rotation
  }
}
```

Gambar 3.21 Kode Algoritma Sensor dalam Aplikasi Arduino

Setelah didapatkan algoritma tersebut, lanjut ke penyaluran output dari Arduino ke dalam aplikasi Node-Red. Hasil pembuatan algoritma pembacaan nilai dari Arduino dapat dilihat pada Gambar 3.22.



Gambar 3.22 Algoritma Pembacaan Sensor pada Miniatur dalam Aplikasi Node-Red

Dalam flow algoritma tersebut, digunakan sebuah *node* baru, yaitu *serial in node*.

Gambar 3.23 Bentuk *Serial In Node*

Serial in node memiliki fungsi untuk membaca hasil *output* yang dihasilkan oleh Arduino, lalu dijadikan *input* ke dalam *flow* algoritma yang digunakan. Untuk dapat mendeteksi Arduino yang digunakan, perlu dispesifikasikan *port* yang tersambung dengan Arduino. Dalam simulasi ini, Arduino tersambung pada *port* COM5 yang terdapat pada laptop yang digunakan. Tampilan pengaturan *serial in node* dapat dilihat pada Gambar 3.24.

Gambar 3.24 Pengaturan *Serial In Node*

Output yang dihasilkan oleh *serial in node* masih dalam bentuk variabel string, sedangkan aplikasi InfluxDB hanya dapat membaca *input* dengan bentuk variabel *number*. Untuk mengubah bentuk variabel *output* tersebut sekaligus membedakan *output* dari setiap sensor, digunakan beberapa *function node* dengan kegunaan yang sama untuk setiap *output* dari setiap sensor. Kode yang bekerja di dalam *function node* tersebut dapat dilihat pada Gambar 3.25, Gambar 3.26, dan Gambar 3.27.

```
const input = msg.payload.trim();

if (input.startsWith("RPM:")) {
    const rpmValue = parseInt(input.split(":")[1]);

    if (!isNaN(rpmValue)) {
        msg.payload = rpmValue
        return msg;
    }
}

return null;
```

Gambar 3.25 Kode *Function Node* untuk Sensor Putaran Motor

```
const input = msg.payload.trim();

if (input.startsWith("Current:")) {
    const ampValue = parseFloat(input.split(":")[1]);

    if (!isNaN(ampValue)) {
        msg.payload = ampValue
        return msg;
    }
}

return null;
```

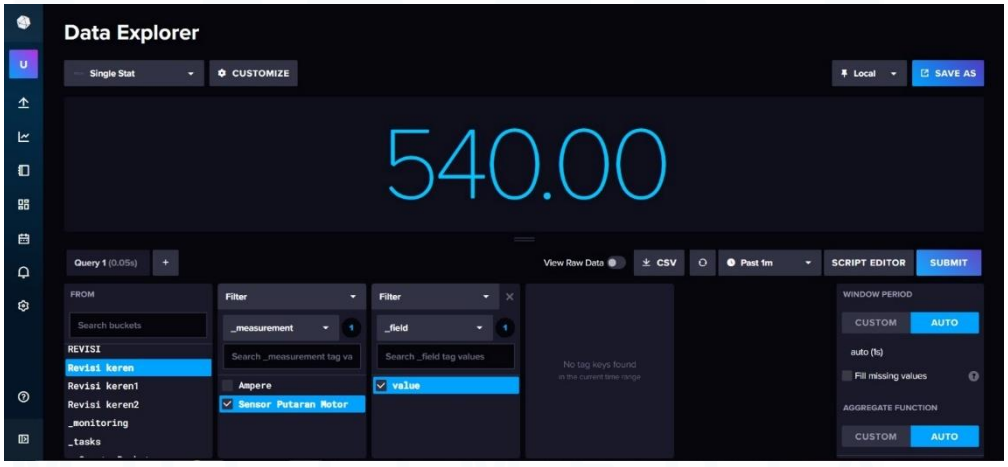
Gambar 3.26 Kode *Function Node* untuk Sensor Ampere

```
const input = msg.payload.trim();

if (input.startsWith("Temperature:")) {
    const tempValue = parseFloat(input.split(":")[1]);

    if (!isNaN(tempValue)) {
        msg.payload = tempValue
        return msg;
    }
}

return null;
```

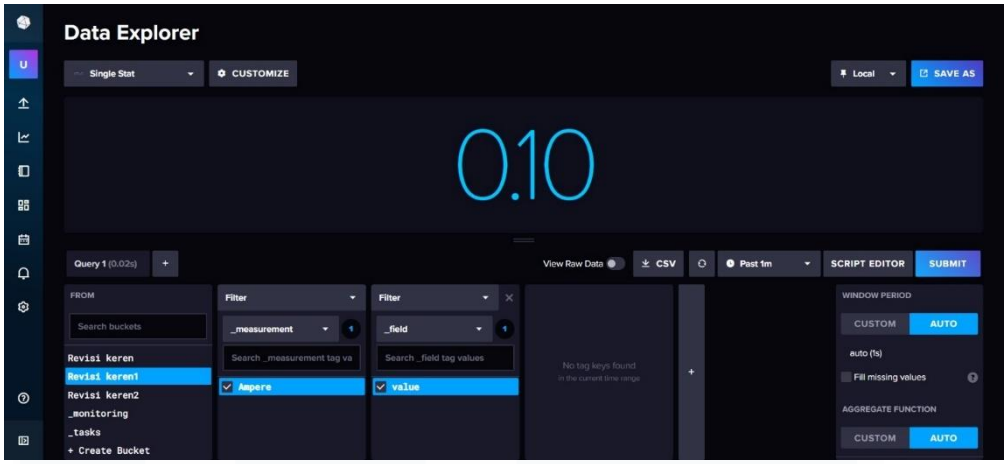
Gambar 3.27 Kode *Function Node* untuk Sensor Temperatur

Kode yang digunakan berfungsi untuk membedakan *output* sensor berdasarkan keterangan *output* yang terbaca dari Arduino, yaitu “RPM: “, “Current: “, atau “Temperature: “. Setelah didapatkan *output* nya, kode tersebut merubah *format output* yang terbaca sambil menghilangkan kata keterangannya sehingga dapat dibaca oleh aplikasi InfluxDB. Untuk pengiriman data yang dikumpulkan ke dalam aplikasi InfluxDB, tetap digunakan influxdb out *node*.

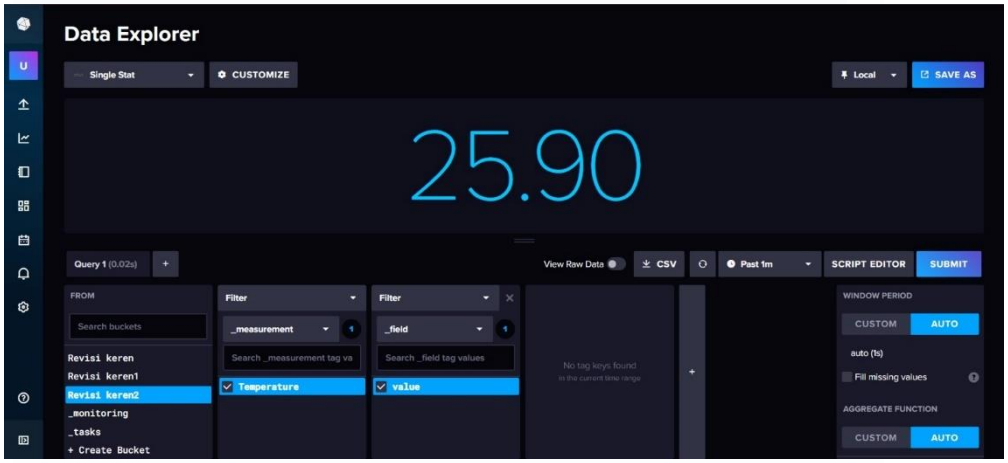
Penampilan data untuk sensor rotasi motor dapat dilihat pada Gambar 3.28, penampilan data untuk sensor ampere dapat dilihat pada Gambar 3.29, dan penampilan data untuk sensor temperatur dapat dilihat pada Gambar 3.30.



Gambar 3.28 Penampilan Data Sensor Putaran Motor pada Aplikasi InfluxDB



Gambar 3.29 Penampilan Data Ampere Motor pada Aplikasi InfluxDB



Gambar 3.30 Penampilan Data Sensor Temperatur pada Aplikasi InfluxDB

3.3 Kendala yang Ditemukan

Dalam pengerjaan proyek utama, beberapa kendala ditemukan dalam proses pengerjaannya. Kendala-kendala tersebut berupa:

- Tidak adanya perangkat sensor langsung yang dapat dipakai untuk pengujian sistem koneksi. Hal tersebut diakibatkan oleh keberadaan perangkat tersebut di lokasi lapangan yang berada di luar kota, dan tidak adanya izin untuk mendatangi lokasi lapangan terus menerus.
- Kurangnya penguasaan penulis dalam penggunaan aplikasi-aplikasi yang dipakai dalam pengerjaan proyek utama. Hal tersebut diakibatkan jarang

digunakannya aplikasi-aplikasi tersebut dalam pekerjaan magang sehari-hari.

- Kurangnya referensi dalam pengerjaan proyek utama dikarenakan sumber referensi dipegang oleh supervisor yang kehadirannya tidak menentu. Hal tersebut diakibatkan supervisor yang sering bepergian keluar kota atau pulau untuk pekerjaannya.

3.4 Solusi atas Kendala yang Ditemukan

Solusi yang ditemukan dari kendala-kendala tersebut berupa:

- Untuk menggantikan penggunaan perangkat sensor langsung, dibuat algoritma penghasil angka pada aplikasi Node-Red. Batasan minimal dan maksimal angka-angka yang dibuat berasal dari diskusi dengan klien proyek utama tersebut.
- Kurangnya penguasaan dalam penggunaan aplikasi-aplikasi dibantu ajarkan oleh *supervisor* dan teman-teman angkatan yang lebih menguasai aplikasi-aplikasi tersebut.