

1. NO. JOBSHEET : 1

**2. JUDUL : DASAR PEMROGRAMAN ESP32 UNTUK
PEMROSESAN DATA INPUT/OUTPUT ANALOG
DAN DIGITAL**

3. TUJUAN

- 1) Mahasiswa dapat memahami dan mengoperasikan GPIO pada ESP32.
- 2) Mahasiswa dapat memahami dan melakukan pengolahan data untuk input/output analog dan digital.
- 3) Mahasiswa dapat melakukan optimalisasi pembacaan sensor analog menggunakan metode regresi linear.

4. ALAT DAN BAHAN

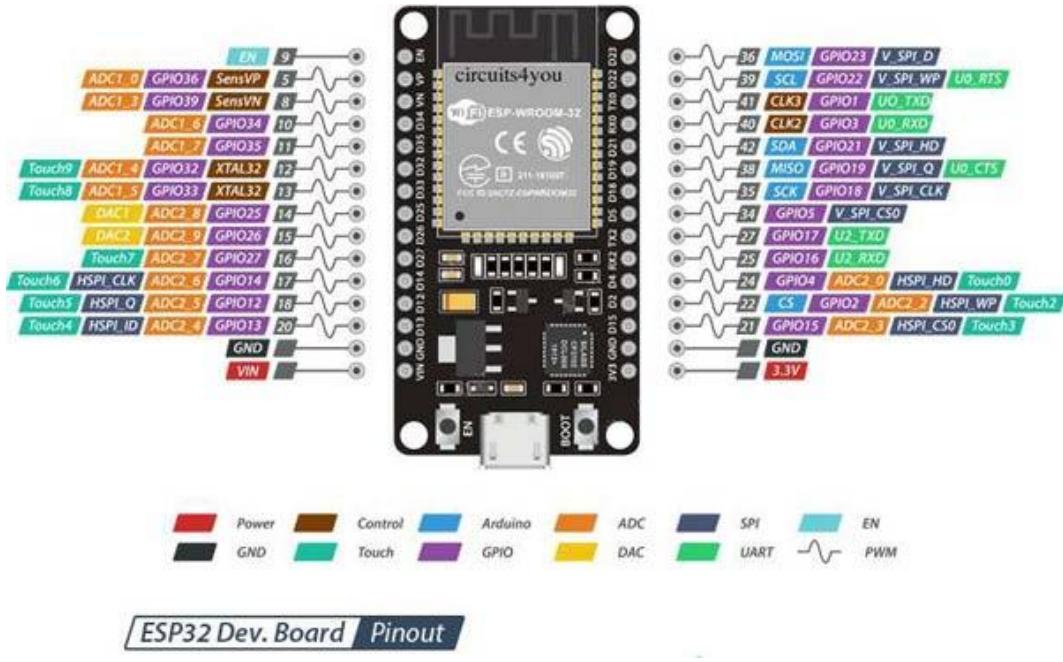
- | | |
|------------------------------|------------------------------------|
| 1) ESP32 | 5) Sensor Capacitive Soil Moisture |
| 2) Breadboard | 6) LED (5) dan Push Button (3) |
| 3) Kabel jumper | 7) Multimeter |
| 4) Potensiometer 10k Ohm (1) | 8) Resistor 330,1K, 10K Ohm (@ 3) |

5. TEORI SINGKAT

ESP-32 adalah mikrokontroler yang dikenalkan oleh Espressif System merupakan penerus dari mikrokontroler ESP8266. Pada mikrokontroler ini sudah tersedia modul WiFi dalam chip sehingga sangat mendukung untuk membuat sistem aplikasi Internet of Things. Perbedaan antara ESP32 dengan ESP8266 adalah pada bagian prosesornya. ESP32 sudah Dual-Core 32 bit, jelas lebih cepat ESP32 secara kinerja. Selain itu modul ini juga mempunyai bluetooth, satu fitur yang tidak ada di ESP8266.

Gambar 5.1 merupakan susunan pin pada modul ESP32. Pada pin out tersebut terdiri dari :

- 18 ADC (Analog Digital Converter)
- 2 DAC (Digital Analog Converter)
- 16 PWM (Pulse Width Modulation)
- 10 Sensor sentuh
- 2 jalur antarmuka UART
- pin antarmuka I2C, I2S, dan SPI



Gambar 5.1 Pinout ESP32

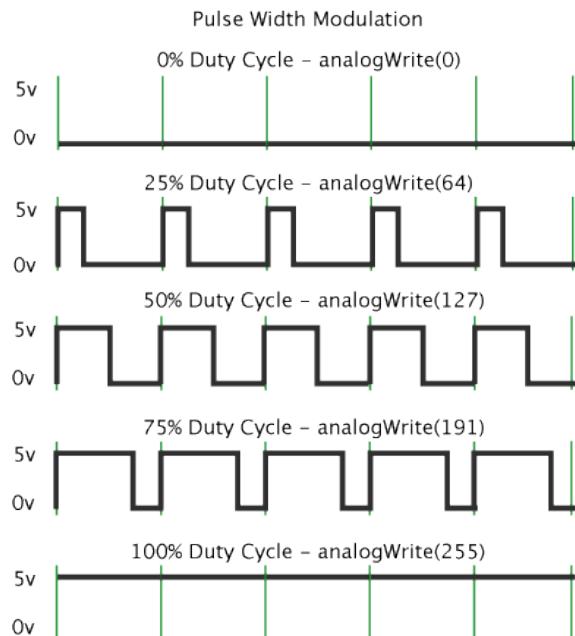
5.1 Pulse Wide Modulation (PWM)

Pin analog pada ESP32 (dan mikrokontroller lain pada umumnya) dapat digunakan sebagai *input* dan *output* digital. Hanya saja pin analog memiliki fitur untuk dapat mengubah sinyal analog yang masuk menjadi nilai digital yang mudah diukur. Pin digital hanya dapat mengenali sinyal 0 volt sebagai nilai *LOW* dan 3,3 volt sebagai nilai *HIGH*. Sedangkan pin analog dapat mengenali sinyal pada rentang nilai voltase tersebut.

Pin analog terhubung dengan *converter* pada mikrokontroller yang dikenal dengan istilah *analog-to-digital converter* (disingkat ADC atau A/D). *Converter* ini mengubah nilai analog berbentuk sinyal voltase ke dalam bentuk digital/angka supaya nilai analog ini dapat digunakan dengan lebih mudah dan aplikatif. ESP32 didukung perangkat keras yang mampu membaca input channel ADC hingga resolusi 12 bit. Hal tersebut berarti bahwa ESP32 mampu mendapatkan pembacaan analog mulai dari 0 hingga 4095, di mana 0 sesuai dengan 0V dan 4095 hingga 3.3V. Resolusi channel ADC tersebut dapat dikonversi juga menjadi lebih kecil menggunakan kode dan rentang ADC pada program.

Analog *output* pada *microkontroller* dihasilkan oleh teknik yang dikenal dengan istilah PWM atau *Pulse Width Modulation*. PWM memanipulasi keluaran

digital sedemikian rupa sehingga menghasilkan sinyal analog. Metode PWM menggunakan pendekatan perubahan lebar pulsa untuk menghasilkan nilai tegangan analog yang diinginkan. Pin yang difungsikan sebagai PWM analog *output* akan mengeluarkan sinyal pulsa digital dengan frekuensi 5000 Hz yang mana nilai tegangan analog diperoleh dengan mengubah *duty cycle* atau perbandingan lamanya pulsa HIGH terhadap periode (T) dari sinyal digital tersebut. Mikrokontroler melakukan pengaturan *output* digital ke *HIGH* dan *LOW* bergantian dengan porsi waktu tertentu untuk setiap nilai keluarannya. Durasi waktu untuk nilai *HIGH* disebut *pulse width* atau panjang pulsa. Hal tersebut dapat dilihat pada Gambar 5.2.



Gambar 5.2. Duty cycle pada PWM

Sumber : www.arduino.cc

Kondisi *HIGH* adalah kondisi ketika sinyal berada di atas grafik (3,3V) dan *LOW* adalah ketika sinyal berada di bawah (0V). *Duty cycle* adalah persentasi panjang pulsa *HIGH* dalam satu periode sinyal. Ketika *duty cycle* 0% atau sinyal *LOW* penuh, maka nilai analog yang dikeluarkan adalah 0V atau setara dengan GND. Jika pulsa *HIGH* muncul selama setengah dari periode sinyal, maka *duty cycle* yang dihasilkan adalah 50% yang berarti sinyal analog yang dihasilkan sebesar setengah dari tegangan analog maksimal yaitu 1/2 dari 3,3 V atau sama

dengan 1,65 V. Ketika *duty cycle* 100% atau sinyal penuh maka sinyal yang dikeluarkan adalah 3.3V.

5.2 Regresi Linear

Regresi analisis adalah teknik statistika untuk menginvestigasi dan memodelkan hubungan antara variabel dari data statistik sebelumnya. Pengaplikasian regresi cukup banyak dan terjadi hampir di banyak bidang seperti bidang keteknikan, ilmu fisika dan kimia, ekonomi, manajemen, ilmu biologi dan ilmu sosial. Bahkan analisis regresi mungkin lebih banyak digunakan dalam teknik statistik.

Regresi dapat dibedakan menjadi tiga jenis, yaitu regresi linier, regresi multi linier dan regresi tak linier. Di dalam model regresi linier terdapat dua jenis variabel yaitu variabel bebas atau input tegangan (X) dan variabel tak bebas atau output sensor (Y). Dalam bentuk yang paling sederhana, regresi linier direpresentasikan pada persamaan (5.1).

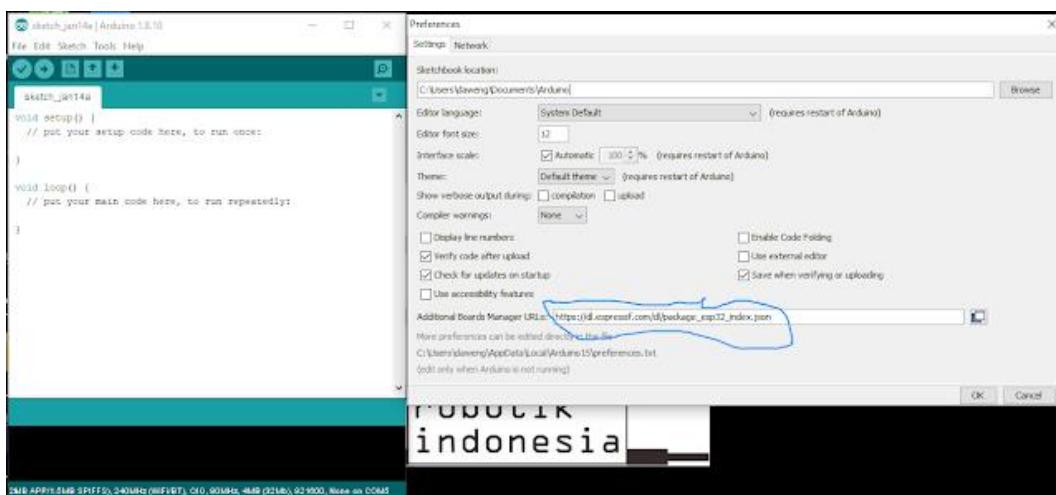
$$Y = A + Bx \quad (5.1)$$

Di mana A disebut sebagai sumbu awal dan B adalah koefisien arah atau koefisien beta.

6. LANGKAH PERCOBAAN

A. Instalasi Board ESP32 pada Arduino IDE

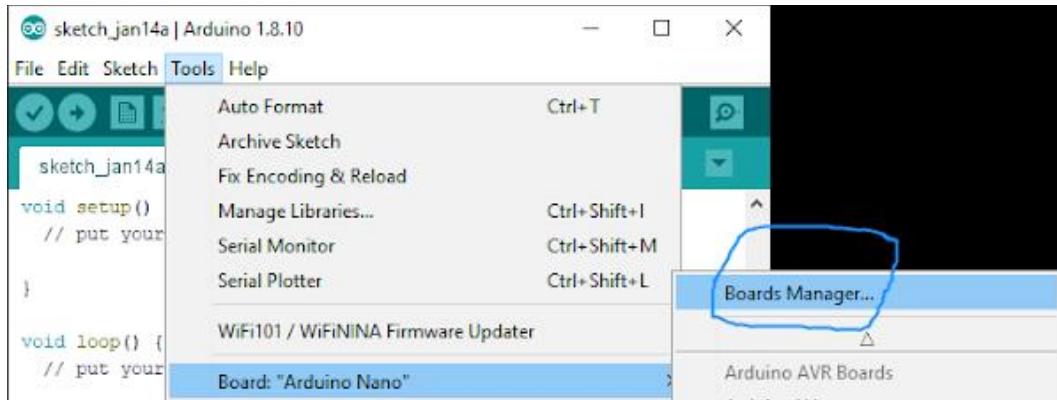
1. Buka Arduino IDE
2. Kemudian klik Menu File > Preferences



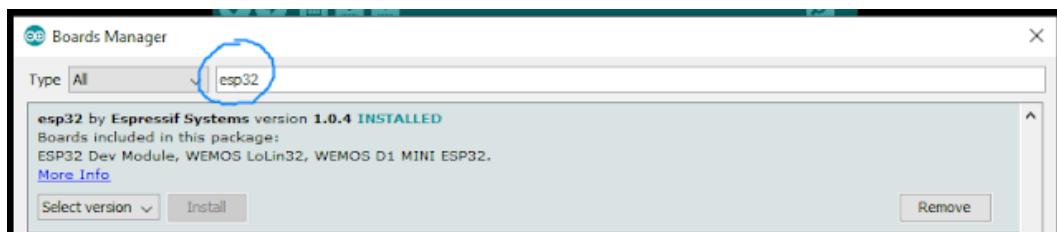
3. Pada kolom Additional ... yang ada dibawah, tambahkan link berikut

https://dl.espressif.com/dl/package_esp32_index.json

4. Klik menu Tools > Board: > Pilih Boards Manager ...



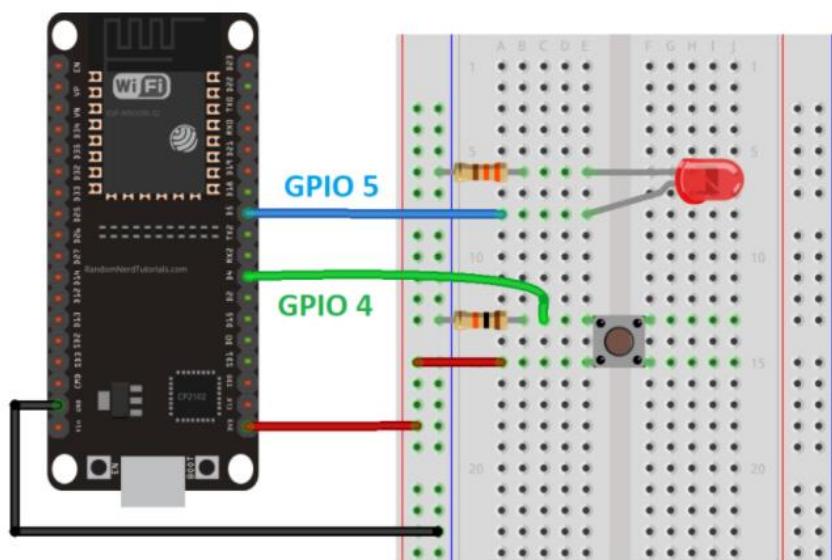
5. Pada kolom pencarian tulis ESP32 kemudian install dan tunggu sampai selesai.



B. Mengakses GPIO dan PWM ESP32

a) GPIO

1. Buatlah rangkaian seperti pada Gambar di bawah ini.



2. Buka program example blink, kemudian modifikasi dan buat agar LED dapat melakukan blink dengan interval 100ms, 1 detik, 2 detik dan 3 detik sekali. Setelah itu, buatlah program agar LED dapat blink 1 detik sekali menggunakan timer millis(). Dokumentasikan hasilnya.
3. Buatlah program seperti pada script di bawah ini untuk mengendalikan led menggunakan push button. Kemudian upload program tersebut pada ESP32 dan dokumentasikan hasilnya.

```
// set pin numbers
const int buttonPin = 4; // the number of the pushbutton pin
const int ledPin = 5; // the number of the LED pin

// variable for storing the pushbutton status
int buttonState = 0;

void setup() {
  Serial.begin(115200);
  // initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
  // initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);
}

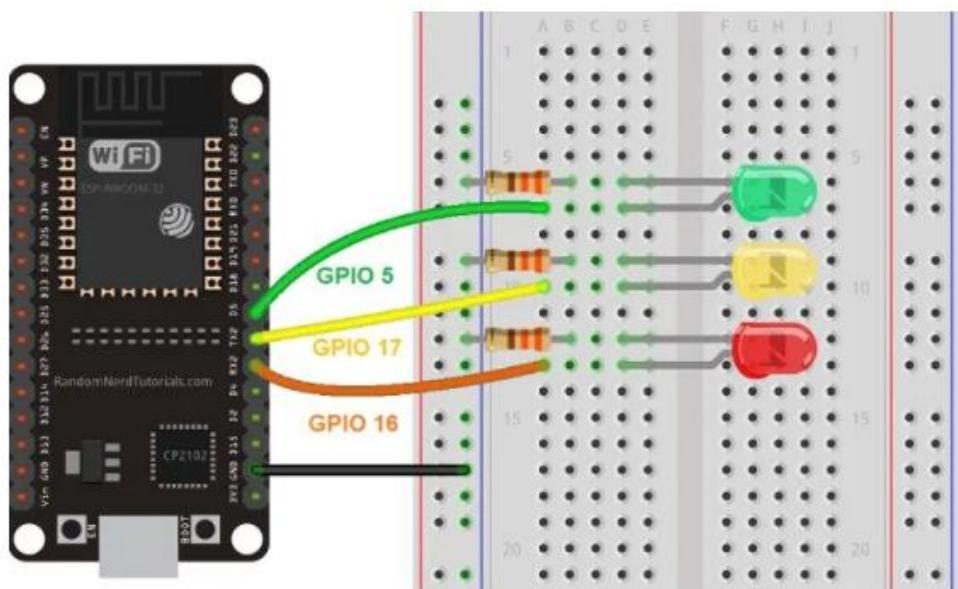
void loop() {
  // read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH
  if (buttonState == HIGH) {
    // turn LED on
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off
    digitalWrite(ledPin, LOW);
  }
}
```

4. Tambahkan 1 LED dan 1 push button pada rangkaian, kemudian kembangkan program agar ketika push button ke-2 ditekan, LED akan melakukan blink setiap 500 ms sekali. Kemudian dokumentasikan hasilnya.

5. Tambahkan 3 LED dan 1 push button pada rangkaian, kemudian kembangkan program agar ketika push button ke-3 ditekan, LED akan menyala menjadi running led (menyala bergantian dari kiri ke kanan). Setelah itu dokumentasikan hasilnya.

2) PWM

1. Buatlah rangkaian seperti pada gambar di bawah ini.



2. Buatlah script program seperti berikut.

```
// the number of the LED pin
const int ledPin = 16; // 16 corresponds to GPIO16

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0; //PWM Channel
const int resolution = 8; //resolution bit

void setup(){
    // configure LED PWM functionalitites
    ledcSetup(ledChannel, freq, resolution);

    // attach the channel to the GPIO to be controlled
    ledcAttachPin(ledPin, ledChannel);
}

void loop(){
    // increase the LED brightness
}
```

```

for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
}

// decrease the LED brightness
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
}

```

3. Upload program tersebut, kemudian amati dan analisis apa yang terjadi serta dokumentasikan hasilnya.
4. Buatlah program lanjutan seperti pada script berikut ini.

```

// the number of the LED pin
const int ledPin = 16; // 16 corresponds to GPIO16
const int ledPin2 = 17; // 17 corresponds to GPIO17
const int ledPin3 = 5; // 5 corresponds to GPIO5

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup(){
    // configure LED PWM functionalitites
    ledcSetup(ledChannel, freq, resolution);

    // attach the channel to the GPIO to be controlled
    ledcAttachPin(ledPin, ledChannel);
    ledcAttachPin(ledPin2, ledChannel);
    ledcAttachPin(ledPin3, ledChannel);
}

void loop(){
    // increase the LED brightness
    for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
        // changing the LED brightness with PWM
    }
}

```

```

        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }

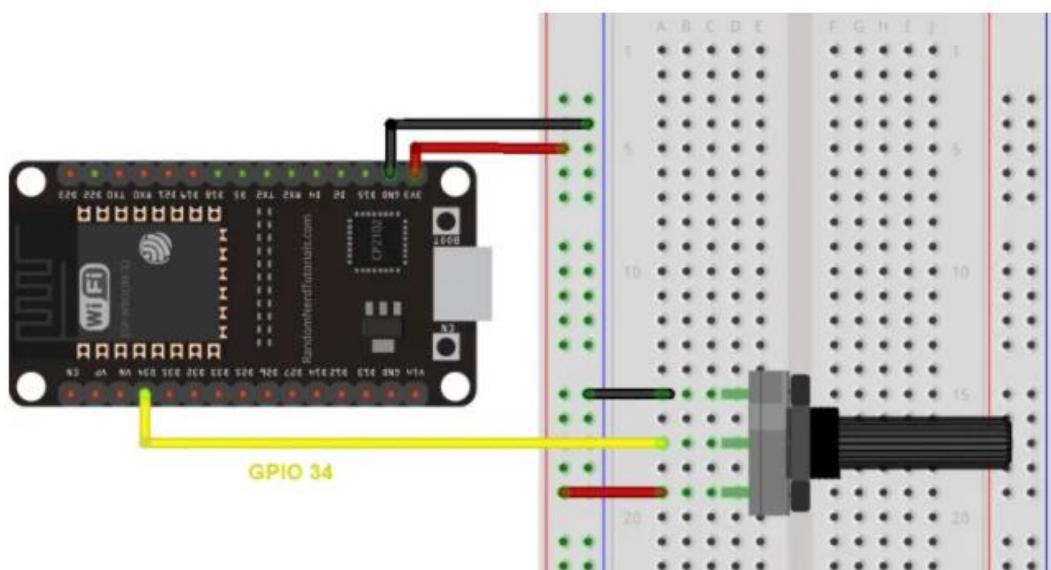
    // decrease the LED brightness
    for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
        // changing the LED brightness with PWM
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
}

```

- Upload program tersebut, kemudian amati dan analisis apa yang terjadi serta dokumentasikan hasilnya.

C. ADC dan DAC

- Buatlah rangkaian seperti pada gambar di bawah ini.



- Buatlah program seperti pada script berikut ini.

```

// Potentiometer is connected to GPIO 34 (Analog ADC1_CH6)
const int potPin = 34;

// variable for storing the potentiometer value
int potValue = 0;

void setup() {
    Serial.begin(115200);
}

```

```

    delay(1000);
}

void loop() {
    // Reading potentiometer value
    potValue = analogRead(potPin);
    Serial.println(potValue);
    delay(500);
}

```

3. Putar potensiometer secara perlahan agar mendapatkan nilai 0 hingga 4095 pada tampilan serial monitor. Analisis apa yang terjadi dan dokumentasikan hasilnya.
4. Buatlah program seperti pada script berikut ini.Tambahkan LED pada GPIO 5.

```

// These constants won't change. They're used to give names to the pins used:
const int analogInPin = 34; // Analog input pin that the potentiometer is attached to
const int analogOutPin = 5; // Analog output pin that the LED is attached to

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

int sensorValue = 0;      // value read from the pot
int outputValue = 0;      // value output to the PWM (analog out)

void setup() {
    Serial.begin(115200); // initialize serial communications at 115200 bps:

    // configure LED PWM functionalitites
    ledcSetup(ledChannel, freq, resolution);

    // attach the channel to the GPIO to be controlled
    ledcAttachPin(analogOutPin, ledChannel);
}

void loop() {
    sensorValue = analogRead(analogInPin); // read the analog in value:
    outputValue = map(sensorValue, 0, 4095, 0, 255); // map it to the range of the analog out:
    analogWrite(analogOutPin, outputValue); // change the analog out value:
}

```

```

// print the results to the Serial Monitor:
Serial.print("sensor = ");
Serial.print(sensorValue);
Serial.print("\t output = ");
Serial.println(outputValue);

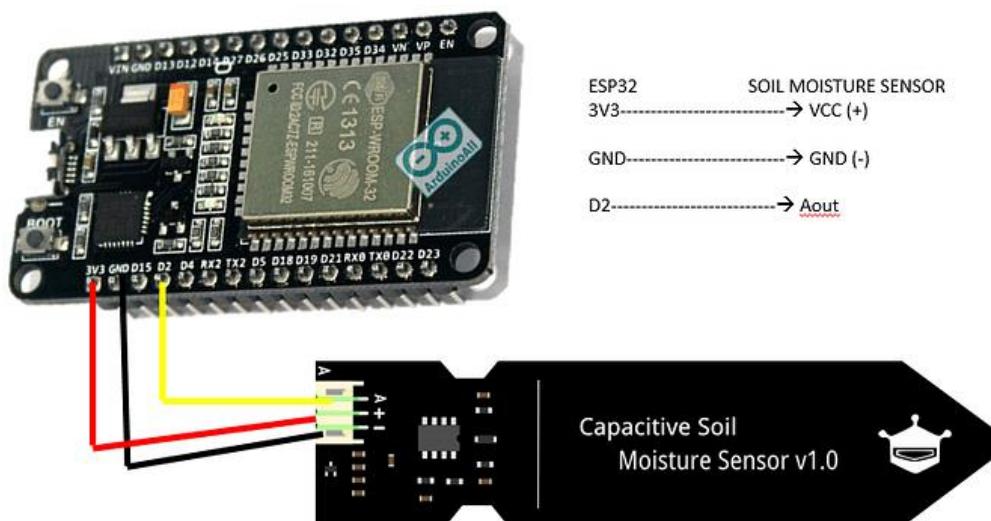
// wait 2 milliseconds before the next loop for the analog-to-digital
// converter to settle after the last reading:
delay(2);
}

```

- Upload program, kemudian putar potensiometer dari nilai terendah hingga nilai tertinggi. Amati yang terjadi, analisis dan dokumentasikan hasilnya.

D. Simulasi Pemrosesan Data Menggunakan Regresi Linier

- Buatlah rangkaian seperti pada Gambar di bawah ini.



- Buatlah dan upload script program berikut ini.

```

const int numReadings = 10;

int readings[numReadings]; // the readings from the analog input
int readIndex = 0; // the index of the current reading
int total = 0; // the running total
int average = 0; // the average

int inputPin = 2;

void setup() {

```

```

// initialize serial communication with computer:
Serial.begin(9600);

// initialize all the readings to 0:
for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
}

}

void loop() {
    // subtract the last reading:
    total = total - readings[readIndex];
    // read from the sensor:
    readings[readIndex] = analogRead(inputPin);
    // add the reading to the total:
    total = total + readings[readIndex];
    // advance to the next position in the array:
    readIndex = readIndex + 1;

    // if we're at the end of the array...
    if (readIndex >= numReadings) {
        // ...wrap around to the beginning:
        readIndex = 0;
    }

    // calculate the average:
    average = total / numReadings;
    // send it to the computer as ASCII digits
    Serial.println(average);
    delay(1);      // delay in between reads for stability
}

```

3. Masukkan sensor pada sampel tanah 1-3.
4. Catat hasil pembacaan sensor dan 3 Way Meter pada masing-masing sampel tanah.
5. Ukur tegangan output sensor pada pin Aout (A) menggunakan multimeter. Kemudian catat hasilnya.
6. Upload program berikut pada ESP32.

```

const int numReadings = 10;

int readings[numReadings];    // the readings from the analog input
int readIndex = 0;            // the index of the current reading
int total = 0;                // the running total

```

```

int average = 0;           // the average
float vs = 0; //sensor voltage

int inputPin = 2;

void setup() {
    // initialize serial communication with computer:
    Serial.begin(9600);
    // initialize all the readings to 0:
    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
        readings[thisReading] = 0;
    }
}

void loop() {
    // subtract the last reading:
    total = total - readings[readIndex];
    // read from the sensor:
    readings[readIndex] = analogRead(inputPin);
    // add the reading to the total:
    total = total + readings[readIndex];
    // advance to the next position in the array:
    readIndex = readIndex + 1;

    // if we're at the end of the array...
    if (readIndex >= numReadings) {
        // ...wrap around to the beginning:
        readIndex = 0;
    }

    // calculate the average:
    average = total / numReadings;
    // send it to the computer as ASCII digits
    Serial.println(average);
    delay(1);      // delay in between reads for stability

    vs = (3.3/4095)* average;
    Serial.println(vs);
}

```

7. Catat keluaran dari vs.
8. Masukkan semua data pada tabel Excel menggunakan format berikut ini.

3 Way Meter	ADC	V out	Vs

9. Buatlah scatter chart dengan sumbu X = Vs dan sumbu Y = 3 Way Meter.
10. Klik kanan line pada scatter chart, kemudian pilih add trendline. Kemudian pada format trendline, atur trendline option = linear, checklist pada Display Equaiton on Chart dan Display R-squared value on Chart.
11. Masukkan persamaan yang dihasilkan grafik ke dalam program Arduino. Ganti variabel a dan b dengan nilai pada persamaan yang dihasilkan.
12. Dokumentasikan hasilnya, serta buatlah analisis dan kesimpulannya.

```

int average = 0;           // the average
float vs = 0; //sensor voltage
float moisture=0;

int inputPin = 2;

void setup() {
    // initialize serial communication with computer:
    Serial.begin(9600);
    // initialize all the readings to 0:
    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
        readings[thisReading] = 0;
    }
}

void loop() {
    // subtract the last reading:
    total = total - readings[readIndex];
    // read from the sensor:
    readings[readIndex] = analogRead(inputPin);
    // add the reading to the total:
    total = total + readings[readIndex];
    // advance to the next position in the array:
    readIndex = readIndex + 1;

    // if we're at the end of the array...
}

```

```
if (readIndex >= numReadings) {  
    // ...wrap around to the beginning:  
    readIndex = 0;  
}  
  
// calculate the average:  
average = total / numReadings;  
// send it to the computer as ASCII digits  
Serial.println(average);  
delay(1);      // delay in between reads for stability  
  
vs = (3.3/4095)* average;  
moisture = ax + b;  
Serial.println(vs);  
Serial.println(moisture);  
}
```

7. PERTANYAAN DAN TUGAS

- 1. NO. JOBSHEET : 2**
- 2. JUDUL : PROTOKOL KOMUNIKASI DAN SENSOR**
- 3. TUJUAN**
 - 1) Mahasiswa dapat memahami cara kerja protokol komunikasi yang terdapat pada ESP32, seperti UART, I2C, OneWire, SPI.
 - 2) Mahasiswa dapat menggunakan protokol komunikasi data seperti UART, I2C, OneWire, dan SPI untuk mengakses sensor.
 - 3) Mahasiswa dapat memanfaatkan transducer sensor dan actuator untuk membuat sebuah perangkat IoT.
- 4. ALAT DAN BAHAN**
 - 1) ESP32
 - 2) Breadboard
 - 3) Kabel jumper
 - 4) Sensor DHT 11, RFID
 - 5) LED (5) dan Push Button (3)
 - 6) Servo
 - 7) Resistor 330,1K, 10K Ohm (@ 3)
- 5. TEORI SINGKAT**

ESP32 adalah nama dari mikrokontroler yang dirancang oleh perusahaan yang berbasis di Shanghai, China yakni Espressif Systems. ESP32 menawarkan solusi jaringan WiFi dan BLE. ESP32 menggunakan prosesor dual core yang berjalan di instruksi Xtensa LX16. Selain itu, ESP32 telah mendukung protokol komunikasi seperti I2C, UART dan SPI.

5.1. *Universal Asynchronous Receiver Transmitter (UART)*

UART adalah protokol komunikasi yang umum digunakan dalam pengiriman data serial antara device satu dengan yang lainnya. Sebagai contoh komunikasi antara sesama mikrokontroler atau mikrokontroler ke PC. Dalam pengiriman data, clock antara pengirim dan penerima harus sama karena paket data dikirim tiap bit mengandalkan clock tersebut. Inilah salah satu keuntungan model asynchronous dalam pengiriman data karena dengan hanya satu kabel transmisi maka data dapat dikirimkan. Berbeda dengan model synchronous yang terdapat pada protokol SPI dan I2C, karena protokol membutuhkan minimal dua kabel dalam transmisi data, yaitu transmisi clock dan data. Namun kelemahan model asynchronous adalah dalam hal kecepatannya dan jarak transmisi. Karena semakin cepat dan jauhnya

jarak transmisi membuat paket-paket bit data menjadi terdistorsi sehingga data yang dikirim atau diterima bisa mengalami error.

Asynchronous memungkinkan transmisi mengirim data tanpa sang pengirim harus mengirimkan sinyal detak ke penerima. Sebaliknya, pengirim dan penerima harus mengatur parameter waktu di awal dan bit khusus ditambahkan untuk setiap data yang digunakan untuk mensinkronkan unit pengiriman dan penerimaan. Saat sebuah data diberikan kepada UART untuk transmisi Asynchronous, "Bit Start" ditambahkan pada setiap awal data yang akan ditransmisikan. Bit Start digunakan untuk memperingatkan penerima yang kata data akan segera dikirim, dan memaksa bit-bit sinyal di receiver agar sinkron dengan bit-bit sinyal di pemancar. Kedua bit ini harus akurat agar tidak memiliki penyimpangan frekuensi dengan lebih dari 10% selama transmisi bit-bit yang tersisa dalam data. (Kondisi ini ditetapkan pada zaman teleprinter mekanik dan telah dipenuhi oleh peralatan elektronik modern. Setelah Bit Start, bit individu dari data yang dikirim, dengan sinyal bit terkecil yang pertama dikirim. Setiap bit dalam transmisi ditransmisikan serupa dengan jumlah bit lainnya, dan penerima mendeteksi jalur di sekitar pertengahan periode setiap bit untuk menentukan apakah bit adalah 1 atau 0. Misalnya, jika dibutuhkan dua detik untuk mengirim setiap bit, penerima akan memeriksa sinyal untuk menentukan apakah itu adalah 1 atau 0 setelah satu detik telah berlalu, maka akan menunggu dua detik dan kemudian memeriksa nilai bit berikutnya, dan seterusnya.

5.2. Serial Peripheral Interface (SPI)

SPI adalah protokol data serial sinkron digunakan oleh mikrokontroler untuk berkomunikasi dengan satu atau lebih perangkat periferal cepat jarak pendek. Hal ini juga dapat digunakan untuk komunikasi antara dua mikrokontroler. Dengan koneksi SPI selalu ada perangkat satu master (biasanya mikrokontroler) yang mengontrol perangkat periferal.

SPI merupakan salah satu mode komunikasi serial synchrounous kecepatan tinggi yang dimiliki oleh Atmega 328. Komunikasi SPI membutuhkan 3 jalur yaitu MOSI, MISO, dan SCK. Melalui komunikasi ini data dapat saling dikirimkan baik antara mikrokontroller maupun antara mikrokontroller dengan

peripheral lain di luar mikrokontroller. Penjelasan 3 jalur utama dari SPI adalah sebagai berikut.

1. **MOSI** : Master Output Slave Input Artinya jika dikonfigurasi sebagai master maka pin MOSI sebagai output tetapi jika dikonfigurasi sebagai slave maka pin MOSI sebagai input.
2. **MISO** : Master Input Slave Output Artinya jika dikonfigurasi sebagai master maka pin MISO sebagai input tetapi jika dikonfigurasi sebagai slave maka pin MISO sebagai output.
3. **CLK** : Clock Jika dikonfigurasi sebagai master maka pin CLK berlaku sebagai output tetapi jika dikonfigurasi sebagai slave maka pin CLK berlaku sebagai input.

5.3. *Inter Integrated Circuit (I2C)*,

I2C adalah sebuah protokol untuk komunikasi serial antar IC, dan sering disebut juga Two Wire Interface (TWI). Bus yang digunakan untuk komunikasi antara mikrokontroler dan device lainnya seperti sensor, dll.

Komunikasi dilakukan melalui dua jalur: SDA (serial data) dan SCL (serial clock). Setiap device I2C memiliki 7-bit alamat yang unik. MSB adalah fix dan ditujukan untuk kategori device. Sebagai contoh, 1010 biner ditujukan untuk serial EEPROM. Tiga bit berikutnya memungkinkan 8 kombinasi alamat I2C, yang berarti, dimungkinkan 8 device dengan tipe yang sama, beroperasi pada bus I2C yang sama. Pengalaman 7-bit memungkinkan 128 device pada bus yang sama. Alamat I2C dikirim dalam byte pertama. LSB dari byte ini digunakan untuk menunjukkan bila master akan melakukan penulisan (0) atau pembacaan (0) terhadap slave.

Device yang mengirim data sepanjang bus disebut master. Sementara device yang menerima data disebut slave. Master memulai transmisi dengan sebuah sinyal start, dan menghentikan transmisi dengan sebuah sinyal stop pada jalur SDA. Selama sinyal start dan stop, jalur SCL harus dalam keadaan high. Setelah master memulai pengiriman data dengan sebuah sinyal start, master menulis satu byte alamat device kepada slave. Setiap byte data harus memiliki panjang 8-bit. Slave harus memberikan konfirmasi dari byte data yang diterimanya dengan sebuah bit acknowledge (ACK).

6. LANGKAH PERCOBAAN

A. ESP32 Capacitive Touch Sensor

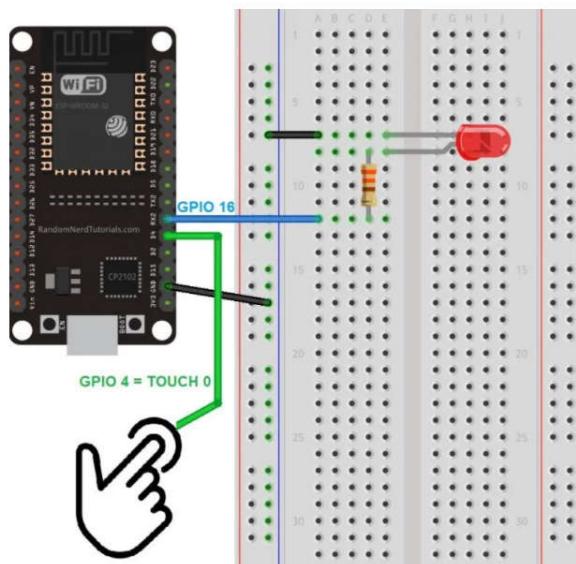
1. Hubungkan kabel jumper Male-to-Female pada Pin D4 ESP32.
2. Buka Arduino IDE dan upload script program berikut ke ESP32.

```
// ESP32 Touch Test
// Just test touch pin - Touch0 is T0 which is on GPIO 4.

void setup() {
    Serial.begin(115200);
    delay(1000); // give me time to bring up serial monitor
    Serial.println("ESP32 Touch Test");
}

void loop() {
    Serial.println(touchRead(4)); // get value of Touch 0 pin = GPIO 4
    delay(1000);
}
```

3. Buka serial monitor untuk melihat raw data. Ubah tampilan serial monitor menjadi Serial Plotter pada menu **Tools > Serial Plotter**.
4. Sentuh ujung kabel jumper dan amati yang terjadi, kemudian dokumentasikan hasilnya.
5. Buatlah rangkaian seperti gambar berikut ini.

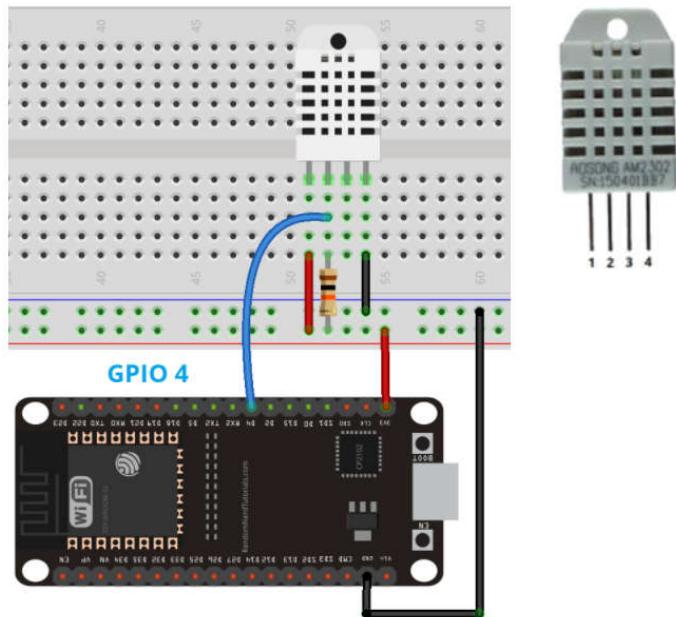


6. Buatlah program agar LED menyala ketika sensor disentuh, dan LED akan mati ketika sensor tidak disentuh.
7. Buatlah program agar ketika sensor disentuh, LED menyala Blink.

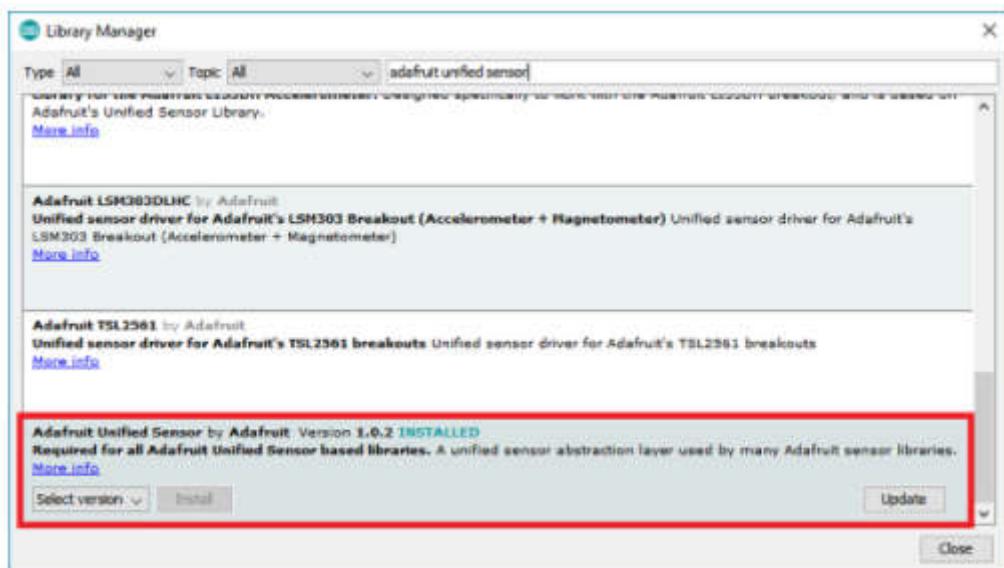
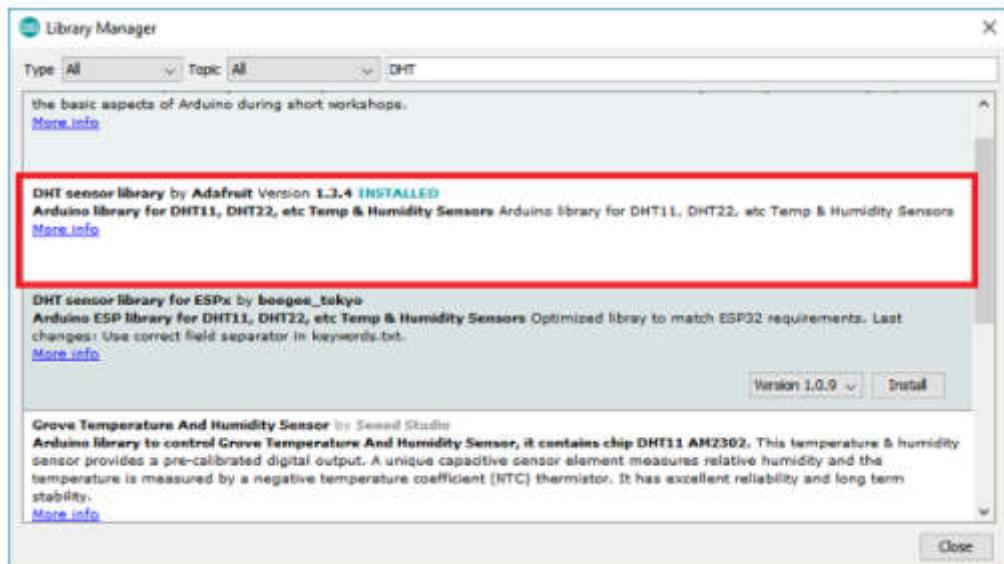
8. Buatlah program agar ketika LED menyala, maka pada Serial Monitor akan menampilkan angka yang akan bertambah setiap kali sensor disentuh.
9. Tambahkan 2 LED sehingga pada rangkaian terdapat 3 LED. Buatlah program agar ketika sensor disentuh, LED menyala menjadi running LED. Nyala running LED tersebut adalah bergerak dari kiri ke kanan, kemudian kanan ke kiri secara kontinyu.

B. Mengakses Sensor DHT 11 (Single Wire / BUS)

1. Buatlah rangkaian seperti pada Gambar di bawah ini.



2. Install library sensor DHT 11 melalui **Sketch > Include Library > Manage Libraries**. Ketikkan **DHT** pada kolom pencarian, pilih library yang akan diinstall seperti pada Gambar berikut ini. Kemudian install juga **Adafruit Unified Sensor** menggunakan cara yang sama.



- Buatlah program seperti pada script di bawah ini untuk mengakses sensor DHT11. Kemudian upload program tersebut pada ESP32 dan dokumentasikan hasilnya.

```
// REQUIRES the following Arduino libraries:  
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library  
// - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit\_Sensor  
  
#include "DHT.h"  
  
#define DHTPIN 4 // Digital pin connected to the DHT sensor  
  
// Uncomment whatever type you're using!  
#define DHTTYPE DHT11 // DHT 11  
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321  
// #define DHTTYPE DHT21 // DHT 21 (AM2301)
```

```

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 Embedded System Test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahrenheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

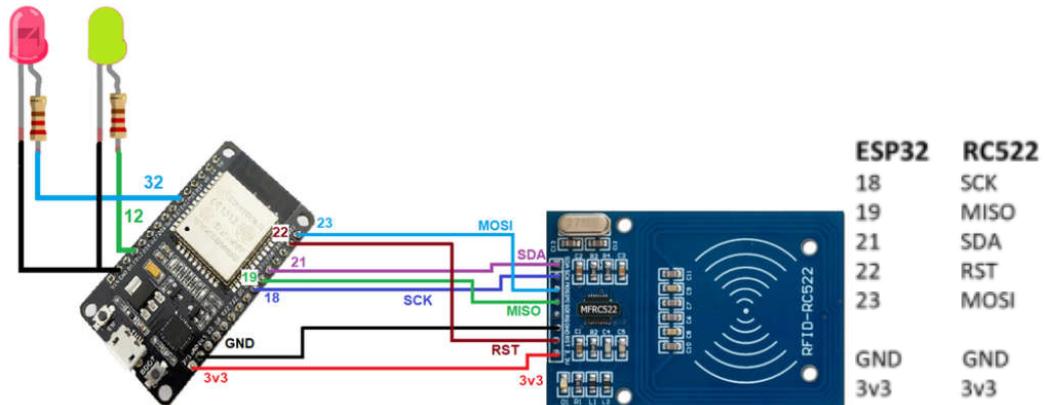
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("% Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}

```

4. Buatlah program agar ketika suhu rungan mencapai 30 derajat celcius, maka ESP32 akan menyalakan LED Merah dan buzzer secara beep (blink). Apabila suhu dibawah 30 derajat, ESP32 akan mematikan buzzer dan menyalakan LED berbentuk running LED seperti pada **Percobaan A**. Kemudian dokumentasikan hasilnya.

C. Mengakses Sensor RFID (SPI Communication)

1. Buatlah rangkaian seperti pada gambar di bawah ini.



2. Install Library MFRC522 dari Library Manager.
3. Kemudian buatlah program seperti pada script berikut ini.

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 21 // ESP32 pin GPIO21
#define RST_PIN 22 // ESP32 pin GPIO22

MFRC522 rfid(SS_PIN, RST_PIN);

byte keyTagUID[4] = {0xFF, 0xFF, 0xFF, 0xFF};

void setup() {
  Serial.begin(9600);
  SPI.begin(); // init SPI bus
  rfid.PCD_Init(); // init MFRC522

  Serial.println("Tap RFID/NFC Tag on reader");
}

void loop() {
  if (rfid.PICC_IsNewCardPresent()) { // new tag is available
    if (rfid.PICC_ReadCardSerial()) { // NUID has been readed
      MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);

      if (rfid.uid.uidByte[0] == keyTagUID[0] &&
          rfid.uid.uidByte[1] == keyTagUID[1] &&
          rfid.uid.uidByte[2] == keyTagUID[2] &&
          rfid.uid.uidByte[3] == keyTagUID[3] ) {
        Serial.println("Access is granted");
      }
      else
      {
        Serial.print("Access denied for user with UID:");
        for (int i = 0; i < rfid.uid.size; i++) {
```

```

        Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(rfid.uid.uidByte[i], HEX);
    }
    Serial.println();
}

rfid.PICC_HaltA(); // halt PICC
rfid.PCD_StopCrypto1(); // stop encryption on PCD
}
}
}

```

4. Dekatkan kartu atau Tag RFID ke RFID Reader. Amati dan analisis cara kerja programnya.
5. Buatlah program agar Tag RFID yang terbaca sebelumnya dapat digunakan untuk hak akses. Apabila Tag RFID didekatkan pada Reader, maka LED Hijau akan menyala, servo akan bergerak ke kanan (lalu kembali ke posisi semula setelah 3 detik) dan di Serial Monitor akan tertampil pesan “Akses Diterima, Silahkan Masuk”. Apabila Tag RFID tidak dikenali, maka LED Merah akan menyala, servo tidak bergerak dan di Serial Monitor akan tertampil pesan “Akses Ditolak”. Gunakan Tag RFID lain untuk mencoba.
6. Amati yang terjadi, analisis dan dokumentasikan hasilnya.

7. PERTANYAAN DAN TUGAS

1. NO. JOBSHEET : 2.1**2. JUDUL : JARINGAN SENSOR NIRKABEL MENGGUNAKAN ESP-NOW****3. TUJUAN**

- 1) Mahasiswa dapat memahami konsep topologi jaringan sensor nirkabel berbasis ESP-NOW.
- 2) Mahasiswa dapat melakukan konfigurasi berbagai topologi ESP-NOW.
- 3) Mahasiswa dapat menganalisis dan menentukan topologi ESP-NOW, sesuai dengan studi kasus proyek.

4. ALAT DAN BAHAN

- | | |
|---------------|---------------------|
| 1) ESP32 | 3) Kabel jumper |
| 2) Breadboard | 4) Resistor 10K Ohm |

5. TEORI SINGKAT

ESP-NOW adalah protokol yang dikembangkan oleh Espressif, yang memungkinkan banyak perangkat untuk berkomunikasi satu sama lain tanpa menggunakan Wi-Fi. Protokol ini mirip dengan koneksi nirkabel 2.4GHz berdaya rendah. Pendefinisian alamat (MAC Address) pada masing-masing ESP32 diperlukan pada awal konfigurasi. Setelah konfigurasi alamat selesai dilakukan, jaringan *peer-to-peer* akan terbentuk dan perangkat tidak perlu melakukan *handshaking* kembali ketika akan berkomunikasi. Hal ini memunjukkan bahwa setelah perangkat ESP32 saling terpasang satu sama lain, koneksi akan tetap ada. Dengan kata lain, jika tiba-tiba salah satu ESP32 kehilangan daya atau diatur ulang, ketika *restart*, secara otomatis akan terhubung ke pasangan ESP32 yang telah terdefinisi alamatnya untuk melanjutkan komunikasi.

ESP-NOW mempunyai fitur sebagai berikut.

- a. Komunikasi unicast yang terenkripsi maupun tidak terenkripsi.
- b. Perpaduan komunikasi data yang terenkripsi maupun yang tidak terenkripsi pada perangkat yang berada pada topologi peer-to-peer.
- c. Payload (ukuran) data yang dapat dikirim mencapai 250 byte.
- d. Terdapat fungsi *callback* yang dapat menginformasikan data berhasil terkirim maupun gagal dikirim.

Selain itu, ESP-NOW mempunyai batasan sebagai berikut.

- a. Jumlah maksimal perangkat yang dapat berkomunikasi dalam mode station dengan data terenkripsi adalah 10 unit (6 dalam mode SoftAP atau SoftAP+Station).
- b. Untuk komunikasi tidak terenkripsi, jumlah maksimal perangkat adalah 20 unit, termasuk dengan yang terenkripsi.

ESP-NOW mempunyai 2 tipe jaringan, yaitu One-Way Communication dan Two-Way Communication. One-Way Communication terbagi menjadi Point-to-Point, One-to-Many Communication dan Many-to-One Communication. Sementara Two-Way Communication terbagi menjadi Point-to-Point dan Mesh Communication.

5.1. One-Way Communication



Gambar 1. Point-to-Point pada One Way Communication



Gambar 2. One-to-Many Communication



Gambar 3. Many-to-One Communication

Konfigurasi point-to-point seperti Gambar 1 sangat mudah diterapkan dan sangat baik untuk mengirim data dari satu ESP32 ke ESP32 lainnya seperti pembacaan sensor atau perintah ON/OFF untuk mengontrol GPIO. Sementara pada Gambar 2, satu board ESP32 (master) mengirimkan perintah yang sama atau berbeda ke papan ESP32 yang berbeda (slave). Konfigurasi ini sangat ideal untuk membangun suatu sistem seperti remote control. Pada Gambar 3, konfigurasi tersebut sangat ideal untuk sistem pengumpulan data dari beberapa node sensor ke dalam satu board ESP32. Topologi ini mirip seperti topologi star, yang mana ESP32 yang berperan sebagai Slave dapat dikonfigurasi sebagai Web Server.

5.2. Two-Way Communication



Gambar 4. Point-to-Point pada Two-Way Communication



Gambar 5. Topologi Mesh

Konfigurasi point-to-point pada mode Two-Way Communication seperti yang ditunjukkan pada Gambar 4, masing-masing board ESP32 dapat saling berkomunikasi secara simultan untuk bertukar informasi. Sementara itu, Gambar 5 merupakan konfigurasi ESP-NOW dengan topologi Mesh. Dengan topologi tersebut, setiap board ESP32 akan dapat saling berkomunikasi dengan semua anggota jaringan. Hal tersebut dapat mencegah terjadinya kegagalan transmisi data, karena terdapat jalur cadangan. Namun, konsumsi energi pada masing-masing ESP32 akan menjadi semakin besar.

6. LANGKAH PERCOBAAN

A. Memperoleh MAC Address ESP32 Receiver

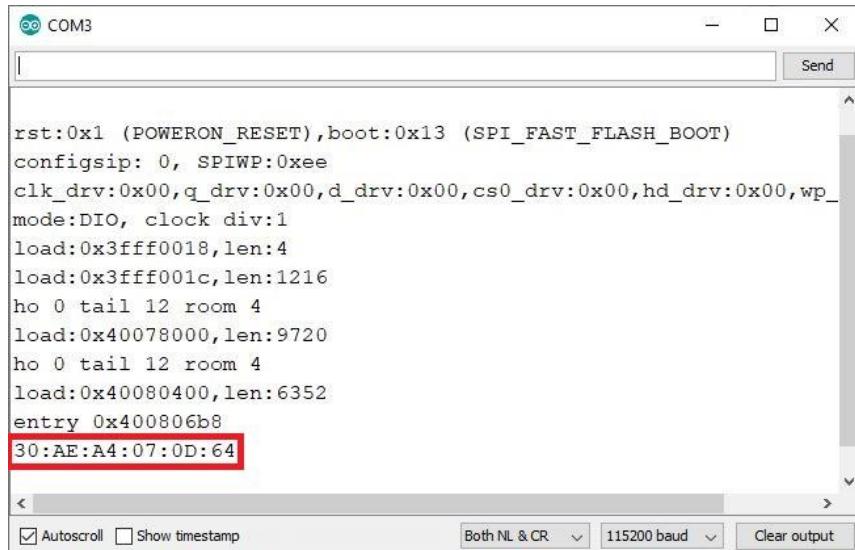
1. Buka Arduino IDE
2. Kemudian ketikkan script program berikut di Arduino IDE.

```
#include "WiFi.h"

void setup(){
    Serial.begin(115200);
    WiFi.mode(WIFI_MODE_STA);
    Serial.println(WiFi.macAddress());
}

void loop(){}
```

3. Upload program tersebut ke ESP32.
4. Setelah program berhasil diupload, buka serial monitor.
5. Catat Mac Address ESP32.



B. ESP-NOW One-Way Point-to-Point Communication

1. Siapkan board ESP32 sejumlah 2 unit yang akan diprogram sebagai Sender dan Receiver.
2. Kemudian ketikkan script program berikut untuk mengkonfigurasi ESP32 sebagai sender.
3. Isi array broadcastAddress [] dengan MAC Address ESP32 Receiver yang didapatkan pada praktikum poin A.
4. Upload program pada ESP32 Sender.

```

#include <esp_now.h>
#include <WiFi.h>

// Ganti dengan Mac Address ESP32 Receiver
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// Struktur pesan sender dan receiver harus sama
typedef struct struct_message {
    char a[32];
    int b;
    float c;
    bool d;
} struct_message;

// Driver untuk struktur pesan
struct_message myData;

esp_now_peer_info_t peerInfo;

// fungsi callback
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nStatus Paket Terakhir :\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Sukses Terkirim" : "Gagal Terkirim");
}

void setup() {
    // Init Serial Monitor
    Serial.begin(115200);

    // Set ESP32 sebagai station
    WiFi.mode(WIFI_STA);

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Gagal menginisialisasi ESP-NOW");
        return;
    }

    // Fungsi akses register cb untuk perintah mengirim data
    esp_now_register_send_cb(OnDataSent);

    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    // Add peer
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Gagal menambahkan peer");
        return;
    }
}

void loop() {
    // Set values to send
    strcpy(myData.a, "INI ADALAH CHAR");
    myData.b = random(1,20);
    myData.c = 1.2;
}

```

```

myData.d = false;

// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));

if (result == ESP_OK) {
    Serial.println("Data berhasil terkirim");
} else {
    Serial.println("Gagal mengirim data");
}
delay(2000);
}

```

5. Setelah ESP32 sender dikonfigurasi, kemudian konfigurasikan ESP32 yang lain sebagai Receiver. Ketikkan script program berikut untuk mengkonfigurasi ESP32 sebagai receiver.
6. Upload program, kemudian dokumentasikan hasilnya.
7. Buatlah analisis atau flow chart program untuk menjelaskan fungsi per-bagian pada program.

```

#include <esp_now.h>
#include <WiFi.h>

// Struktur pesan sender dan receiver harus sama
typedef struct struct_message {
    char a[32];
    int b;
    float c;
    bool d;
} struct_message;

// Driver struktur pesan
struct_message myData;

// fungsi callback yang akan dieksekusi ketika ada pesan diterima
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    memcpy(&myData, incomingData, sizeof(myData));
    Serial.print("Bytes received: ");
    Serial.println(len);
    Serial.print("Char: ");
    Serial.println(myData.a);
    Serial.print("Int: ");
    Serial.println(myData.b);
    Serial.print("Float: ");
    Serial.println(myData.c);
    Serial.print("Bool: ");
    Serial.println(myData.d);
    Serial.println();
}

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200);
}

```

```

// Set ESP32 sebagai station
WiFi.mode(WIFI_STA);

// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
}

// Fungsi akses register cb untuk proses penerimaan data
esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
}

```

8. Buatlah data dummy dengan ukuran yang terbaca oleh receiver \pm 250 byte.
9. Aturlah jarak awal komunikasi antara sender dan receiver yaitu 1 meter. Kemudian ubah jarak komunikasi dengan penambahan 1 meter. Data yang dikirim pada tiap iterasi pengujian adalah 10 data.
10. Aturlah tinggi antena atau peletakan ESP32 pada ground level (menempel tanah), 30 cm di atas tanah, dan 1 meter di atas tanah.
11. Masukkan hasil pengukuran pada kolom berikut dan buatlah analisisnya dengan pendekatan teori freshnel zone.

Tinggi Antena (m)	Jarak Transmisi (m)	Jumlah Pesan yang Dikirim [Dt]	Jumlah Pesan yang Diterima [Dr]	Packet Loss (%) $\frac{Dt - Dr}{Dt} \times 100$
Ground	1			
	2			
	Dst ...			
0.3	1			
	2			
	Dst ...			
1	1			
	2			
	Dst ...			

C. One-Way, One-to-Many Communication

a) Mengirim Pesan yang Sama Ke Beberapa Board ESP32

1. Siapkan 4 unit board ESP32.
2. Unggah program untuk mendapatkan Mac Address, kemudian catat masing-masing Mac Address pada setiap board ESP32 yang akan diatur sebagai Receiver.
3. Siapkan 1 unit ESP-32 yang akan dikonfigurasi sebagai Master/Sender.
4. Ketik program berikut ini dan tambahkan semua Mac Address ESP32 Receiver pada bagian broadcastAddress[].

```

#include <esp_now.h>
#include <WiFi.h>

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS
uint8_t broadcastAddress1[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
uint8_t broadcastAddress2[] = {0xFF, , , , , };
uint8_t broadcastAddress3[] = {0xFF, , , , , };

typedef struct test_struct {
    int x;
    int y;
} test_struct;

test_struct test;

esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    char macStr[18];
    Serial.print("Packet to: ");
    // Copies the sender mac address to a string
    sprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x",
            mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
            mac_addr[5]);
    Serial.print(macStr);
    Serial.print(" send status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
Fail");
}

void setup() {
    Serial.begin(115200);

    WiFi.mode(WIFI_STA);

    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    esp_now_register_send_cb(OnDataSent);

    // register peer
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    // register first peer
    memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Failed to add peer");
        return;
    }
    // register second peer
    memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Failed to add peer");
        return;
    }
    // register third peer
}

```

```

    memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Failed to add peer");
        return;
    }

void loop() {
    test.x = random(0,20);
    test.y = random(0,20);

    esp_err_t result = esp_now_send(0, (uint8_t *) &test, sizeof(test_struct));

    if (result == ESP_OK) {
        Serial.println("Sent with success");
    }
    else {
        Serial.println("Error sending the data");
    }
    delay(2000);
}

```

5. Upload program tersebut pada board ESP32 Sender.
6. Siapkan board Receiver, kemudian ketik script berikut ini pada Arduino IDE.

```

#include <esp_now.h>
#include <WiFi.h>

//Structure example to receive data
//Must match the sender structure
typedef struct test_struct {
    int x;
    int y;
} test_struct;

//Create a struct_message called myData
test_struct myData;

//callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    memcpy(&myData, incomingData, sizeof(myData));
    Serial.print("Bytes received: ");
    Serial.println(len);
    Serial.print("x: ");
    Serial.println(myData.x);
    Serial.print("y: ");
    Serial.println(myData.y);
    Serial.println();
}

void setup() {
    //Initialize Serial Monitor
    Serial.begin(115200);

    //Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);

    //Init ESP-NOW
    if (esp_now_init() != ESP_OK) {

```

```

    Serial.println("Error initializing ESP-NOW");
    return;
}

// Once ESPNow is successfully Init, we will register for recv CB to
// get recv packer info
esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
}

```

7. Upload program tersebut pada Receiver.
8. Dokumentasikan output dari program tersebut secara lengkap pada masing-masing board.
9. Matikan salah satu board Receiver, dokumentasikan hasilnya, dan buatlah analisisnya.
10. Buatlah koneksi menggunakan semua board ESP32 yang ada dikelas, dengan menambahkan Receiver ke dalam jaringan secara bertahap,
11. Dokumentasikan hasilnya secara lengkap. Catat dan analisis jumlah board maksimal yang dapat membentuk jaringan.

b) Mengirim Pesan yang Berbeda Ke Beberapa Board ESP32

1. Siapkan 4 board ESP32, 1 board sebagai Sender dan 3 board sebagai Receiver.
2. Buatlah program pada Sender agar dapat mengirim pesan yang berbeda pada 3 Receiver.
3. Tips : Buat 3 buat struktur data.

Buat 3 random data dummy generator pada masing-masing variabel x dan y.

Gunakan fungsi esp_now_send() pada masing-masing broadcastAddress dengan script yang terpisah

D. One-Way, Many-to-One Communication

Di dalam mode ini, Receiver harus dapat mengidentifikasi setiap MAC Address unik dari Sender. Namun, untuk dapat membaca MAC Address yang berbeda cukup rumit dan butuh sedikit trik. Sehingga, untuk membuatnya lebih mudah, masing-masing Sender akan diberikan ID unik, agar Receiver dapat lebih mudah mengidentifikasi Sender.

1. Siapkan 4 board ESP32. 3 board diatur sebagai Sender dan 1 board diatur sebagai Receiver.
2. Unggah program untuk menemukan MAC Address pada board Receiver, kemudian catat MAC Address-nya.
3. Ketikkan program berikut pada Arduino IDE untuk mengkonfigurasi board Sender.

```

#include <esp_now.h>
#include <WiFi.h>

// REPLACE WITH THE RECEIVER'S MAC Address
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
    int id; // must be unique for each sender board
    int x;
    int y;
} struct_message;

// Create a struct_message called myData
struct_message myData;

// Create peer interface
esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
    // Init Serial Monitor
    Serial.begin(115200);

    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Once ESPNow is successfully Init, we will register for Send CB to
    // get the status of Trasnmitted packet
    esp_now_register_send_cb(OnDataSent);

    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    // Add peer
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Failed to add peer");
        return;
    }
}

void loop() {
    // Set values to send
    myData.id = 1;
}

```

```

myData.x = random(0,50);
myData.y = random(0,50);

// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));

if (result == ESP_OK) {
    Serial.println("Sent with success");
}
else {
    Serial.println("Error sending the data");
}
delay(10000);
}

```

4. Upload program tersebut pada board Sender.
5. Siapkan board Receiver, ketikkan script berikut di Arduino IDE, kemudian upload program tersebut.

```

#include <esp_now.h>
#include <WiFi.h>

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    int id;
    int x;
    int y;
}struct_message;

// Create a struct_message called myData
struct_message myData;

// Create a structure to hold the readings from each board
struct_message board1;
struct_message board2;
struct_message board3;

// Create an array with all the structures
struct_message boardsStruct[3] = {board1, board2, board3};

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
    char macStr[18];
    Serial.print("Packet received from: ");
    sprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
            mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
mac_addr[5]);
    Serial.println(macStr);
    memcpy(&myData, incomingData, sizeof(myData));
    Serial.printf("Board ID %u: %u bytes\n", myData.id, len);
    // Update the structures with the new incoming data
    boardsStruct[myData.id-1].x = myData.x;
    boardsStruct[myData.id-1].y = myData.y;
    Serial.printf("x value: %d \n", boardsStruct[myData.id-1].x);
    Serial.printf("y value: %d \n", boardsStruct[myData.id-1].y);
    Serial.println();
}

```

```

}

void setup() {
    //Initialize Serial Monitor
    Serial.begin(115200);

    //Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);

    //Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Once ESPNow is successfully Init, we will register for recv CB to
    // get recv packer info
    esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
    // Acess the variables for each board
    /*int board1X = boardsStruct[0].x;
    int board1Y = boardsStruct[0].y;
    int board2X = boardsStruct[1].x;
    int board2Y = boardsStruct[1].y;
    int board3X = boardsStruct[2].x;
    int board3Y = boardsStruct[2].y;*/

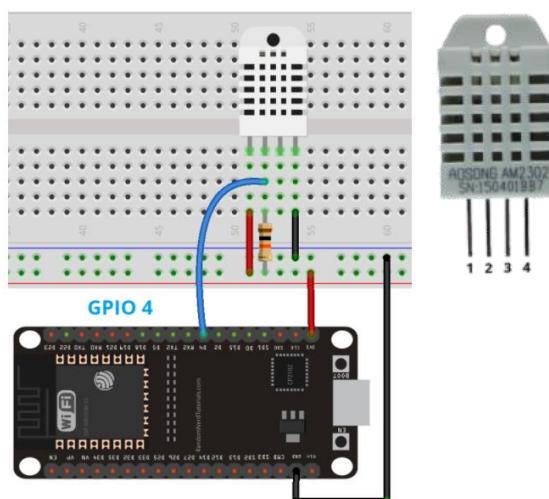
    delay(10000);
}

```

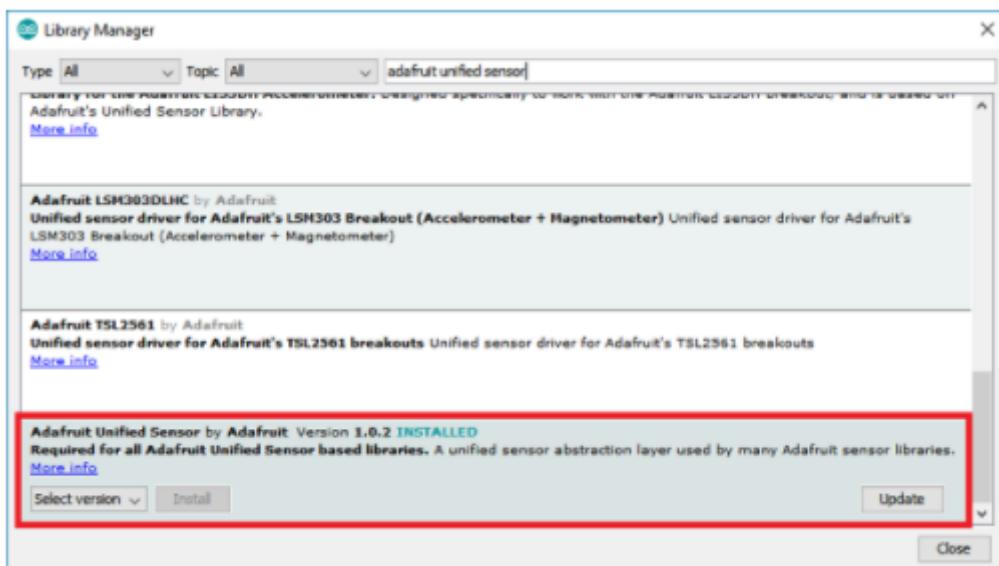
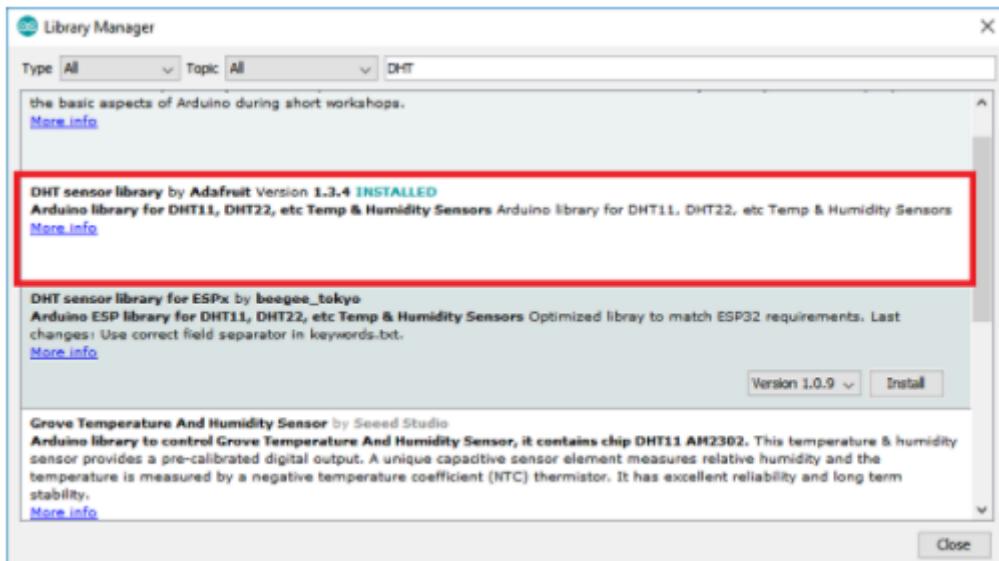
6. Buka serial monitor dan dokumentasikan output program.

E. Two-Way Communication

1. Siapkan 2 unit ESP32 dan 2 unit sensor DHT11.
2. Buatlah rangkaian seperti pada Gambar di bawah ini.



3. Install library sensor DHT 11 melalui **Sketch > Include Library > Manage Libraries**. Ketikkan **DHT** pada kolom pencarian, pilih library yang akan diinstall seperti pada Gambar berikut ini. Kemudian install juga **Adafruit Unified Sensor** menggunakan cara yang sama.



4. Upload program berikut ini untuk melakukan pengecekan sensor DHT11.

```
#include "DHT.h"

#define DHTPIN 4 // Digital pin connected to the DHT sensor

// Uncomment whatever type you're using!
#define DHTTYPE DHT11 // DHT 11
//#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);
```

```

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 Embedded System Test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahrenheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("% Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}

```

5. Dokumentasikan output program yang ditampilkan pada serial monitor Arduino IDE.
6. Unggah program menemukan MAC Address pada masing-masing board, kemudian catat MAC Address-nya.
7. Ketikkan script berikut, kemudian masukkan MAC Address receiver pada masing-masing board.
8. Upload program pada masing-masing board.

```

#include <esp_now.h>
#include <WiFi.h>

#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// REPLACE WITH THE MAC Address of your receiver
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// Define variables
float temperature;
float humidity;

// Define variables to store incoming readings
float incomingTemp;
float incomingHum;

// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the receiver structure
typedef struct struct_message {
    float temp;
    float hum;
} struct_message;

// Create a struct_message sensors reading
struct_message DHTReadings;

// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;

esp_now_peer_info_t peerInfo;

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
    if (status ==0){
        success = "Delivery Success :)";
    }
    else{
        success = "Delivery Fail :(";
    }
}

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
    memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
    Serial.print("Bytes received: ");
    Serial.println(len);
    incomingTemp = incomingReadings.temp;
    incomingHum = incomingReadings.hum;
}

```

```
}

void setup() {
    // Init Serial Monitor
    Serial.begin(115200);

    // Init DHT
    dht.begin();

    // Set device as a Wi-Fi Station
    WiFi.mode(WIFI_STA);

    // Init ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Once ESPNow is successfully Init, we will register for Send CB to
    // get the status of Trasnmitted packet
    esp_now_register_send_cb(OnDataSent);

    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    // Add peer
    if (esp_now_add_peer(&peerInfo) != ESP_OK){
        Serial.println("Failed to add peer");
        return;
    }
    // Register for a callback function that will be called when data is received
    esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
    delay(2000);
    getReadings();

    // Set values to send
    DHTReadings.temp = temperature;
    DHTReadings.hum = humidity;

    // Send message via ESP-NOW
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &DHTReadings,
sizeof(DHTReadings));

    if (result == ESP_OK) {
        Serial.println("Sent with success");
    }
    else {
        Serial.println("Error sending the data");
    }

    delay(1000);
}

void getReadings(){
```

```
temperature=dht.readTemperature();
humidity=dht.readHumidity();

if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
}
Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F("% Temperature: "));
Serial.print(t);
Serial.print(F("°C "));
```

9. Buka serial monitor pada Arduino IDE, kemudian dokumentasikan outputnya.
10. Buatlah jaringan sensor nirkabel ESPNow topologi MESH berlandaskan Two-Way Communication. Diagram blok jaringan dapat dilihat pada Gambar 5.

7. PERTANYAAN DAN TUGAS

- 1. NO. JOBSHEET : 3**
- 2. JUDUL : TOPOLOGI JARINGAN LOKAL DAN WIFI**
- 3. TUJUAN**
 - 1) Mahasiswa dapat memahami cara kerja protokol topologi jaringan lokal yang memanfaatkan Wi-Fi untuk berkomunikasi.
 - 2) Mahasiswa dapat merancang topologi jaringan yang memanfaatkan Wi-Fi untuk penerapan Wireless Sensor Network (WSN) dan Internet of Things (IoT).
 - 3) Mahasiswa dapat memilih dan menggunakan topologi jaringan yang tepat sesuai dengan kondisi lapangan untuk penerapan WSN dan IoT.
- 4. ALAT DAN BAHAN**

1) ESP32	5) LED (5) dan Push Button (3)
2) Breadboard	6) Servo
3) Kabel jumper	7) Resistor 330,1K, 10K Ohm (@ 3)
4) Sensor DHT 11, RFID	
- 5. TEORI SINGKAT**

Wireless Fidelity atau yang lebih awam kita sebut wifi adalah suatu teknologi yang menggunakan gelombang radio dalam rentang 2,4GHz sampai dengan 5GHz untuk menghubungkan perangkat seperti PC/laptop, smartphone, dan perangkat microcontroller seperti ESP32 dan ESP8266 ke jaringan lokal wireless untuk bisa mengakses internet. Untuk dapat melakukan akses internet tersebut,maka perangkat elektronik diatas perlu berada dalam satu titik akses atau hotspot jaringan nirkabel sehingga terhubung dengan wifi. Pada umumnya jaringan wifi dapat menjangkau hingga 20 meter didalam ruangan dan lebih dari 20 meter untuk di luar ruangan. Pada awal kemunculannya, wifi hanya digunakan sebagai perangkat nirkabel pada jaringan LAN (Local Area Network) akan tetapi karena pesatnya teknologi di zaman sekarang wifi menjadi kebutuhan sehari-hari untuk akses jaringan internet dan IoT.

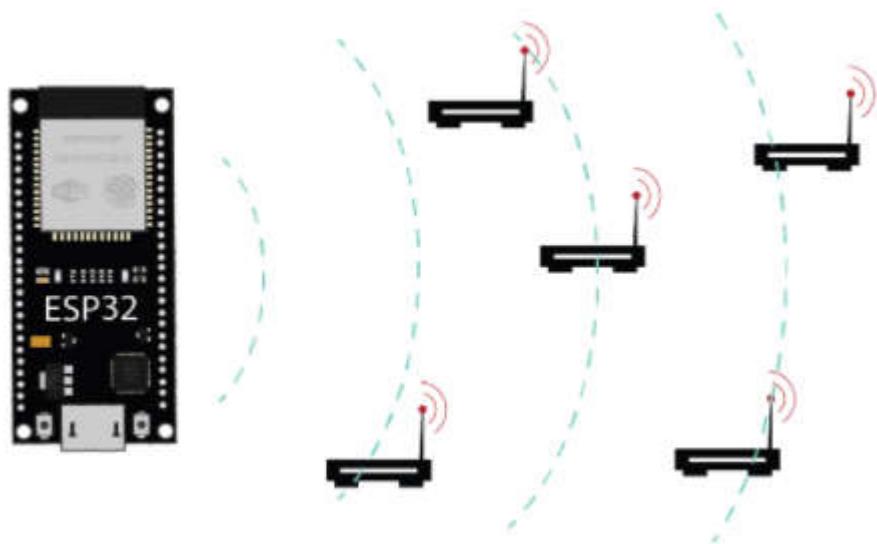
Berbagai data yang kita minta atau kirimkan melalui wifi didistribusikan melalui gelombang radio di udara. Supaya data tersebut bisa terbaca maka harus ada yang namanya wireless adaptor yang menghubungkan ke wifi. Gelombang radio yang berwujud sinyal ini lalu dikirim menuju router yang fungsinya untuk

memecahkan kode. Setelah terbaca maka data dikirim ke jaringan internet yang memanfaatkan koneksi ethernet. Karena jaringan wifi ini bekerja dua arah maka tiap data yang diterima dalam waktu yang sama menjadi kode pada tiap paket data lalu dikirim kembali dalam bentuk sinyal radio yang diterima adaptor komputer nirkabel.

6. LANGKAH PERCOBAAN

A. ESP32 Wi-Fi Modes and Wifi-Scan

1. Pada ESP32, terdapat 3 mode akses untuk Wifi, yaitu WIFI_STA (station mode : ESP32 sebagai client yang terkoneksi ke access point), WIFI_AP (access point mode : ESP32 berperan sebagai access point), WIFI_STA_AP (access point and station : ESP32 dapat terkoneksi dengan access point yang lain).



2. Buka Arduino IDE dan upload script program berikut ke ESP32 untuk melakukan scan jaringan Wi-Fi.

```
/ #include "WiFi.h"

void setup() {
  Serial.begin(115200);

  // Set WiFi to station mode and disconnect from an AP if it was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");
}
```

```

void loop() {
    Serial.println("scan start");

    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(": ");
            Serial.print(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.print(")");
            Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)? " ":"*");
            delay(10);
        }
    }
    Serial.println("");
}

// Wait a bit before scanning again
delay(5000);
}

```

3. Buka serial monitor dan dokumentasikan outputnya.
4. Buatlah flow chart program diatas.

B. Menghubungkan ESP32 dengan Jaringan Wi-Fi

1. Buatlah program seperti script dibawah ini, kemudian upload program tersebut ke ESP32.

```

#include <WiFi.h>

// Replace with your network credentials (STATION)
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi ..");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print('.');
        delay(1000);
    }
}

```

```

    }

    Serial.println(WiFi.localIP());
}

void setup() {
    Serial.begin(115200);
    initWiFi();
    Serial.print("RSSI: ");
    Serial.println(WiFi.RSSI());
}

void loop() {
    // put your main code here, to run repeatedly:
}

```

2. Buka serial monitor, kemudian dokumentasikan outputnya.
3. Buatlah flow chart program diatas.

C. Menghubungkan Kembali (Re-connect) ESP32 dengan Jaringan Wi-Fi

1. Buatlah program seperti script dibawah ini, kemudian upload program tersebut ke ESP32.

```

#include <WiFi.h>

// Replace with your network credentials (STATION)
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

unsigned long previousMillis = 0;
unsigned long interval = 30000;

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi .. ");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print('.');
        delay(1000);
    }
    Serial.println(WiFi.localIP());
}

```

```

void setup() {
    Serial.begin(115200);
    initWiFi();
    Serial.print("RSSI: ");
    Serial.println(WiFi.RSSI());
}

void loop() {
    unsigned long currentMillis = millis();
    // if WiFi is down, try reconnecting every CHECK_WIFI_TIME seconds
    if ((WiFi.status() != WL_CONNECTED) && (currentMillis - previousMillis >= interval)) {
        Serial.print(millis());
        Serial.println("Reconnecting to WiFi... ");
        WiFi.disconnect();
        WiFi.reconnect();
        previousMillis = currentMillis;
    }
}

```

2. Buka serial monitor, kemudian matikan paket data sebentar hingga koneksi ESP32 dengan jaringan Wi-Fi terputus. Setelah itu, nyalakan lagi paket data. Dokumentasikan proses yang terjadi.
3. Buatlah flow chart program diatas.

D. Mengganti Hostname ESP32

1. Buatlah program seperti script dibawah ini, kemudian upload program tersebut ke ESP32.

```

#include <WiFi.h>

// Replace with your network credentials (STATION)
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

String hostname = "ESP32 Node Temperature";

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);
}

```

```

WiFi.setHostname(hostname.c_str()); //define hostname
//wifi_station_set_hostname( hostname.c_str() );
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi .. ");
while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
}
Serial.println(WiFi.localIP());
}

void setup() {
Serial.begin(115200);
initWiFi();
Serial.print("RSSI: ");
Serial.println(WiFi.RSSI());
}

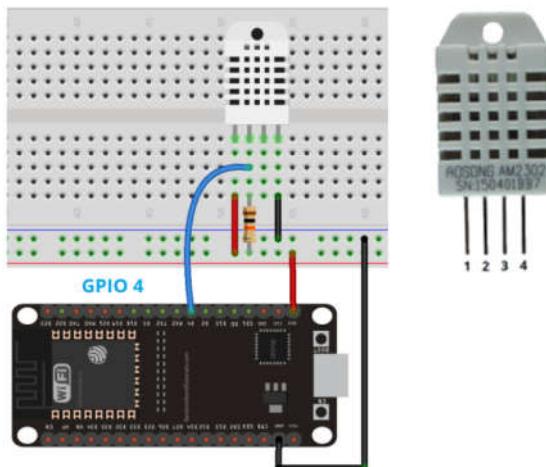
void loop() {
// put your main code here, to run repeatedly:
}

```

2. Buka serial monitor, kemudian aktifkan mode koneksi Wi-Fi pada Smart-Phone atau Laptop. Lakukan scan dan lihat daftar jaringan Wi-Fi yang tersedia. Dokumentasikan hasil keluarannya.
3. Buatlah flow chart program diatas.

E. Mengirim Data Sensor ke Database

1. Buatlah rangkaian seperti Gambar di bawah ini.



1. Install library Asynch Web Server dan Asyncn TCP untuk ESP 32 dengan cara download dari link berikut ini.

- a. <https://github.com/me-no-dev/ESPAsyncWebServer>
- b. <https://github.com/me-no-dev/AsyncTCP/archive/master.zip>

Install library tersebut secara manual dengan cara menyalin folder hasil ekstraksi file.zip ke direktori libarary Arduino di folder Document.

2. Buatlah script program seperti berikut ini.

```
// Import required libraries
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define DHTPIN 4 // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String readDHTTemperature() {
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    //float t = dht.readTemperature(true);
    // Check if any reads failed and exit early (to try again).
    if (isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
        return "--";
    }
    else {
        Serial.println(t);
        return String(t);
    }
}

String readDTHumidity() {
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    if (isnan(h)) {
        Serial.println("Failed to read from DHT sensor!");
```

```

        return "--";
    }
    else {
        Serial.println(h);
        return String(h);
    }
}

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-fnmOCqbTlWlj8LyTjo7mOUSTjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
<style>
html {
font-family: Arial;
display: inline-block;
margin: 0px auto;
text-align: center;
}
h2 {font-size: 3.0rem; }
p {font-size: 3.0rem; }
.units {font-size: 1.2rem; }
.dht-labels{
font-size: 1.5rem;
vertical-align:middle;
padding-bottom: 15px;
}
</style>
</head>
<body>
<h2>ESP32 DHT Server</h2>
<p>
<i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
<span class="dht-labels">Temperature</span>
<span id="temperature">%TEMPERATURE%</span>
<sup class="units">&deg;C</sup>
</p>
<p>
<i class="fas fa-tint" style="color:#00add6;"></i>
<span class="dht-labels">Humidity</span>
<span id="humidity">%HUMIDITY%</span>
<sup class="units">&percnt;</sup>
</p>
</body>
<script>
setInterval(function () {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("temperature").innerHTML = this.responseText;
}
}

```

```

};

xhttp.open("GET", "/temperature", true);
xhttp.send();
}, 10000 );

setInterval(function () {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("humidity").innerHTML = this.responseText;
}
};
xhttp.open("GET", "/humidity", true);
xhttp.send();
}, 10000 );
</script>
</html>)rawliteral";

// Replaces placeholder with DHT values
String processor(const String& var){
//Serial.println(var);
if(var == "TEMPERATURE"){
return readDHTTemperature();
}
else if(var == "HUMIDITY"){
return readDHTHumidity();
}
return String();
}

void setup(){
// Serial port for debugging purposes
Serial.begin(115200);

dht.begin();

// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.println("Connecting to WiFi.. ");
}
}

// Print ESP32 Local IP Address
Serial.println(WiFi.localIP());

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
request->send_P(200, "text/html", index_html, processor);
});
server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
request->send_P(200, "text/plain", readDHTTemperature().c_str());
});
server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
request->send_P(200, "text/plain", readDHTHumidity().c_str());
});
}

```

```
};

// Start server
server.begin();
}

void loop(){
```

3. Upload program di atas. Kemudian buka serial monitor untuk mengetahui IP Address ESP32.
4. Akses IP Address ESP32 pada browser laptop. Dokumentasikan hasilnya dan buatlah flow chart dari program tersebut.

7. PERTANYAAN DAN TUGAS

1. Buatlah program pada ESP32 dengan urutan proses seperti berikut ini, agar ESP32 bisa melakukan setting SSID dan Passsword lebih fleksibel.
 - a. Ketika pertama kali booting, mode ESP32 adalah Station Mode untuk membaca kombinasi SSID dan Password pada jaringan sebelumnya.
 - b. Jika gagal, ESP32 akan berubah mode menjadi Access Point Mode dan membuat jaringan Wi-Fi tanpa proteksi/tanpa password, agar user dapat terhubung dengan ESP32.
 - c. Setelah itu, hubungkan laptop dengan ESP32 dan akses IP Address ESP32 (ESP32 Web Server) pada browser laptop untuk membuat konfigurasi SSID dan Password dan menyimpannya pada EEPROM.
 - d. Kemudian matikan ESP32. ESP32 akan berusaha terhubung dengan jaringan yang telah dikonfigurasi sebelumnya.
 - e. Jika berhasil terhubung, pada serial monitor akan terdapat pesan **Connected to “SSID” Successfully.**
 - f. Jika gagal terhubung, ESP32 akan masuk pada mode Access Point kembali untuk melakukan konfigurasi SSID dan Password.

- 1. NO. JOBSHEET : 4**
- 2. JUDUL** : TRANSMISI DATA MENGGUNAKAN PROTOKOL HTTP DAN MQTT

3. TUJUAN

- 1) Mahasiswa dapat memahami cara kerja protokol HTTP dan MQTT untuk transmisi data (akuisisi data dan kendali) pada Platform IoT Node-Red.
- 2) Mahasiswa dapat merancang dan melakukan konfigurasi pada perangkat Internet of Things (IoT) menggunakan protokol HTTP dan MQTT untuk monitoring dan kendali melalui Platform IoT Node-Red.

4. ALAT DAN BAHAN

- | | |
|------------------------------|------------------------------|
| 1) ESP32 dan Server Node-Red | 4) Sensor DHT 11 |
| 2) Breadboard | 5) LED (5) |
| 3) Kabel jumper | 6) Resistor 330,1K Ohm (@ 3) |

5. TEORI SINGKAT

Penggunaan internet saat ini telah berkembang dan masuk dalam semua aspek kehidupan manusia. Hampir setiap peralatan yang digunakan oleh manusia terhubung ke internet, seperti mobil, jam tangan, peralatan rumah tangga, bahkan binatang peliharaan pun dapat dipakaikan kalung yang dapat mengirimkan data keberadaannya. Hal tersebut kemudian dikenal sebagai Internet of Things (IoT). IoT memerlukan elemen dashboard/plaform yang dapat mengintrepretasikan dan memvisualisasikan data-data yang dikirim ke server, agar mudah di baca dan dipahami oleh pengguna.

Platform IoT adalah suatu ekosistem yang digabungkan untuk menjadi wadah pembuatan produk dan solusi IoT agar efisien dan tidak memakan banyak waktu. Platform IoT disini sebagai lingkungan IoT yang siap dipakai untuk suatu produk atau bisnis. Platform IoT dapat digunakan untuk mengumpulkan data dari berbagai sumber, menyimpan data, menampilkan data, mengontrol perangkat, mengelola inventory perangkat dan lain-lain. Sebelum membangun sebuah platform, terdapat beberapa hal yang perlu dipertimbangkan, antara lain protokol komunikasi yang digunakan, jenis koneksi, jenis jaringan dan format data.

Hypertext Transfer Protocol (HTTP) adalah sebuah protokol jaringan lapisan aplikasi yang digunakan untuk sistem informasi terdistribusi, kolaboratif, dan

menggunakan hypermedia. Protokol tersebut sering diimplementasikan untuk melayani permintaan data dari user dan untuk manajemen website. Sementara itu, *Message Queuing Telemetry Transport* (MQTT) adalah protokol komunikasi yang berjalan di atas *stack* TCP/IP, didesain untuk komunikasi *Machine-to-Machine* (M2M), bersifat *open sources* dan *lightweight*, mempunyai *protocol overhead* yang rendah (minimum 2 bytes) sehingga berefek pada konsumsi daya yang kecil dan mampu bekerja pada *latency* yang tinggi serta *bandwidth* yang kecil, sehingga protokol ini sering dimanfaatkan untuk transmisi data dari Node Sensor menuju Server.

6. LANGKAH PERCOBAAN

A. Setting SSID dan Password Wi-Fi ESP32 melalui Web Server

1. Buka Arduino IDE, kemudian upload script berikut pada ESP32.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <WebServer.h>
#include <EEPROM.h>

//Variables
int i = 0;
int statusCode;
const char* ssid = "Default SSID";
const char* passphrase = "Default passord";
String st;
String content;
String esid;
String epass = "";

//Function Decalration
bool testWifi(void);
void launchWeb(void);
void setupAP(void);

//Establishing Local server at port 80
WebServer server(80);

void setup()
{
    Serial.begin(115200); //Initialising if(DEBUG)Serial Monitor
    Serial.println();
    Serial.println("Disconnecting current wifi connection");
    WiFi.disconnect();
    EEPROM.begin(512); //Initialasing EEPROM
    delay(10);
}
```

```

pinMode(15, INPUT);
Serial.println();
Serial.println();
Serial.println("Startup");

//----- Read eeprom for ssid and pass
Serial.println("Reading EEPROM ssid");

for (int i = 0; i < 32; ++i)
{
    esid += char(EEPROM.read(i));
}
Serial.println();
Serial.print("SSID: ");
Serial.println(esid);
Serial.println("Reading EEPROM pass");

for (int i = 32; i < 96; ++i)
{
    epass += char(EEPROM.read(i));
}
Serial.print("PASS: ");
Serial.println(epass);

WiFi.begin(esid.c_str(), epass.c_str());
}

void loop() {

if ((WiFi.status() == WL_CONNECTED))
{

for (int i = 0; i < 10; i++)
{
    Serial.print("Connected to ");
    Serial.print(esid);
    Serial.println(" Successfully");
    delay(100);
}

}
else
{

if (testWifi() && (digitalRead(2) != 1))
{
    Serial.println(" connection status positive");
    return;
}
else
{
    Serial.println("Connection Status Negative / D15 HIGH");
    Serial.println("Turning the HotSpot On");
}
}
}

```

```

    launchWeb();
    setupAP();// Setup HotSpot
}

Serial.println();
Serial.println("Waiting.");

while ((WiFi.status() != WL_CONNECTED))
{
    Serial.print(".");
    delay(100);
    server.handleClient();
}
delay(1000);
}

//----- Fuctions used for WiFi credentials saving and connecting to
it which you do not need to change
bool testWifi(void)
{
int c = 0;
//Serial.println("Waiting for Wifi to connect");
while ( c < 20 ) {
    if (WiFi.status() == WL_CONNECTED)
    {
        return true;
    }
    delay(500);
    Serial.print("*");
    c++;
}
Serial.println("");
Serial.println("Connect timed out, opening AP");
return false;
}

void launchWeb()
{
Serial.println("");
if (WiFi.status() == WL_CONNECTED)
    Serial.println("WiFi connected");
Serial.print("Local IP: ");
Serial.println(WiFi.localIP());
Serial.print("SoftAP IP: ");
Serial.println(WiFi.softAPIP());
createWebServer();
// Start the server
server.begin();
Serial.println("Server started");
}

void setupAP(void)
{
    WiFi.mode(WIFI_STA);
}

```

```

WiFi.disconnect();
delay(100);
int n = WiFi.scanNetworks();
Serial.println("scan done");
if (n == 0)
    Serial.println("no networks found");
else
{
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i)
    {
        // Print SSID and RSSI for each network found
        Serial.print(i + 1);
        Serial.print(": ");
        Serial.print(WiFi.SSID(i));
        Serial.print("(");
        Serial.print(WiFi.RSSI(i));
        Serial.print(")");
        //Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE) ? " " : "*");
        delay(10);
    }
}
Serial.println("");
st = "<ol>";
for (int i = 0; i < n; ++i)
{
    // Print SSID and RSSI for each network found
    st += "<li>";
    st += WiFi.SSID(i);
    st += "(";
    st += WiFi.RSSI(i);

    st += ")";
    //st += (WiFi.encryptionType(i) == ENC_TYPE_NONE) ? " " : "*";
    st += "</li>";
}
st += "</ol>";
delay(100);
WiFi.softAP("MiSREd IoT", "");
Serial.println("Initializing_softap_for_wifi_credentials_modification");
launchWeb();
Serial.println("over");
}

void createWebServer()
{
{
    server.on("/", []() {
        IPAddress ip = WiFi.softAPIP();
        String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);
        content = "<!DOCTYPE HTML>\r\n<html>Welcome to Wifi Credentials Update page";
        content += "<form action=\"/scan\" method=\"POST\"><input type=\"submit\" value=\"scan\"></form>";

```

```

content += ipStr;
content += "<p>";
content += st;
content += "</p><form method='get' action='setting'><label>SSID: </label><input name='ssid' length=32><input name='pass' length=64><input type='submit'></form>";
content += "</html>";
server.send(200, "text/html", content);
});

server.on("/scan", []() {
//setupAP();
IPAddress ip = WiFi.softAPIP();
String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);

content = "<!DOCTYPE HTML>\r\n<html>go back";
server.send(200, "text/html", content);
});

server.on("/setting", []() {
String qsid = server.arg("ssid");
String qpass = server.arg("pass");
if (qsid.length() > 0 && qpass.length() > 0) {
Serial.println("clearing eeprom");
for (int i = 0; i < 96; ++i) {
EEPROM.write(i, 0);
}
Serial.println(qsid);
Serial.println("");
Serial.println(qpass);
Serial.println("");

Serial.println("writing eeprom ssid:");
for (int i = 0; i < qsid.length(); ++i)
{
EEPROM.write(i, qsid[i]);
Serial.print("Wrote: ");
Serial.println(qsid[i]);
}
Serial.println("writing eeprom pass:");
for (int i = 0; i < qpass.length(); ++i)
{
EEPROM.write(32 + i, qpass[i]);
Serial.print("Wrote: ");
Serial.println(qpass[i]);
}
EEPROM.commit();

content = "{\"Success\":\"saved to eeprom... reset to boot into new wifi\"}";
statusCode = 200;
ESP.restart();
} else {
content = "{\"Error\":\"404 not found\"}";
statusCode = 404;
Serial.println("Sending 404");
}
server.sendHeader("Access-Control-Allow-Origin", "*");
}

```

```

    server.send(statusCode, "application/json", content);

}
}
}

```

2. Setelah itu, buka serial monitor Arduino IDE. Jika ESP32 belum tersambung dengan jaringan, maka EPS32 akan menampilkan daftar SSID Wi-Fi yang tersedia. Kemudian akses IP Address yang ditampilkan pada serial monitor pada browser dan isikan SSID dan Password yang sesuai pada form yang disediakan.
3. Dokumentasikan hasilnya dan buatlah flow chart program tersebut.

B. Transmisi Data Menggunakan Protokol HTTP

1. Buka Virtual Machine pada komputer. Konfigurasi pengaturan adapter jaringan menggunakan mode **bridge**, kemudian jalankan VM Ubuntu.
2. Buatlah database dengan nama **banjir_db**. Kemudian buatlah tabel dengan nama **smartwater**. Setelah itu, buatlah kolom dengan nama sebagai berikut.
 - id (INT [11], not null, auto increment, primary key),
 - time_stamp (not null, current timestamp),
 - dev_id (varchar[64]),
 - level (decimal[16,2]),
 - rainfall (decimal[16,2]),
 - flow (decimal[16,2])
3. Import flow program berikut untuk membuat konfigurasi Multi-Protocol IoT Server.

```

[
  {
    "id": "5b890ccf.cfd994",
    "type": "tab",
    "label": "Multi-Protocol Server",
    "disabled": false,
    "info": ""
  },
  {
    "id": "714c1277091756b2",
    "type": "http in",
    "z": "5b890ccf.cfd994",
    "name": "Post Data",
    "url": "/flood/node1",
    "method": "POST"
  }
]

```

```
        "method": "post",
        "upload": false,
        "swaggerDoc": "",
        "x": 100,
        "y": 120,
        "wires": [
            [
                "cfb8d82279080505",
                "d2b73f6ee32154e0",
                "2c3ad4b412708bcd"
            ]
        ]
    },
    {
        "id": "5bddfc457244127",
        "type": "mysql",
        "z": "5b890ccf.cfd994",
        "mydb": "fe8cc6b6.d28218",
        "name": "DB Banjir",
        "x": 620,
        "y": 180,
        "wires": [
            []
        ]
    },
    {
        "id": "cfb8d82279080505",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "Validation",
        "func": "var status = 0\\nif (msg.payload.dev_id &&\\nmsg.payload.level &&\\n msg.payload.rainfall &&\\nmsg.payload.flow)\\n{\\n status = 1\\n}\\nmsg.payload.status =\nstatus\\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 260,
        "y": 120,
        "wires": [
            [
                "6f7e721672b767bf"
            ]
        ]
    },
    {
        "id": "6f7e721672b767bf",
        "type": "switch",
        "z": "5b890ccf.cfd994",
        "name": "Router",
        "property": "payload.status",
        "t0": "0",
```

```
"propertyType": "msg",
"rules": [
    {
        "t": "eq",
        "v": "1",
        "vt": "str"
    },
    {
        "t": "eq",
        "v": "0",
        "vt": "str"
    }
],
"checkall": "true",
"repair": false,
"outputs": 2,
"x": 390,
"y": 120,
"wires": [
    [
        "700571713ca9a07c",
        "405337a110cc8bb3",
        "d52308cc1f9c1820"
    ],
    [
        "c66b1d1429bf1627",
        "cde681d0057aa22f"
    ]
]
},
{
    "id": "700571713ca9a07c",
    "type": "http response",
    "z": "5b890ccf.cfd994",
    "name": "Resp OK",
    "statusCode": "200",
    "headers": {
        "content-type": "application/json"
    },
    "x": 500,
    "y": 60,
    "wires": []
},
{
    "id": "c66b1d1429bf1627",
    "type": "http response",
    "z": "5b890ccf.cfd994",
    "name": "Resp Bad",
    "statusCode": "400",
    "headers": {
        "content-type": "application/json"
    },
    "x": 860,
```

```

        "y": 60,
        "wires": []
    },
    {
        "id": "49f136c7d34e0ca5",
        "type": "debug",
        "z": "5b890ccf.cfd994",
        "name": "HTTP Log",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 630,
        "y": 140,
        "wires": []
    },
    {
        "id": "405337a110cc8bb3",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "",
        "func": "msg.topic = \"INSERT INTO smartwater\n(dev_id, level, rainfall, flow)\" + \n\" VALUES(\" + \nmsg.payload.dev_id + \",\", \" +\nmsg.payload.level + \",\", \" +\nmsg.payload.rainfall + \",\", \" +\nmsg.payload.flow +\n\")\" +\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 360,
        "y": 180,
        "wires": [
            [
                "5bddcf457244127",
                "49f136c7d34e0ca5"
            ]
        ]
    },
    {
        "id": "ab1f81c3367a8f8e",
        "type": "mqtt in",
        "z": "5b890ccf.cfd994",
        "name": "",
        "topic": "flood/node1",
        "qos": "2",
        "datatype": "auto",
        "broker": "f9a971b7c57d8b6c",

```

```
        "nl": false,
        "rap": false,
        "x": 110,
        "y": 240,
        "wires": [
            [
                "26095e1a8765435f"
            ]
        ]
    },
    {
        "id": "32f8d9ac97e0dc3d",
        "type": "debug",
        "z": "5b890ccf.cfd994",
        "name": "MQTT Log",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 610,
        "y": 280,
        "wires": []
    },
    {
        "id": "de50e0ca8fb829f4",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "",
        "func": "msg.topic = \"INSERT INTO smartwater\n(dev_id, level, rainfall, flow)\" + \n\" VALUES(\" + \nmsg.payload.dev_id + \",\", \" +\nmsg.payload.level + \",\", \" +\nmsg.payload.rainfall + \",\", \" +\nmsg.payload.flow +\n\")\" +\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 420,
        "y": 280,
        "wires": [
            [
                "32f8d9ac97e0dc3d",
                "5bddcf457244127"
            ]
        ]
    },
    {
        "id": "26095e1a8765435f",
```

```
    "type": "json",
    "z": "5b890ccf.cfd994",
    "name": "",
    "property": "payload",
    "action": "",
    "pretty": false,
    "x": 270,
    "y": 280,
    "wires": [
        [
            "de50e0ca8fb829f4",
            "d2b73f6ee32154e0",
            "2c3ad4b412708bcd",
            "912098f194a76ef2",
            "034d5e39764ad47b",
            "5ffbb0b1b58f11eb",
            "cdcf79b2260f5497",
            "1a2c7fcacf9cb07da",
            "1f2356cbc8d15404"
        ]
    ]
},
{
    "id": "d52308cc1f9c1820",
    "type": "debug",
    "z": "5b890ccf.cfd994",
    "name": "OK",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "statusVal": "",
    "statusType": "auto",
    "x": 330,
    "y": 60,
    "wires": []
},
{
    "id": "cde681d0057aa22f",
    "type": "debug",
    "z": "5b890ccf.cfd994",
    "name": "Error",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "statusVal": "",
    "statusType": "auto",
    "x": 670,
```

```
        "y": 60,
        "wires": []
    },
    {
        "id": "8120b93037058f0b",
        "type": "debug",
        "z": "5b890ccf.cfd994",
        "name": "WS Log",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 620,
        "y": 360,
        "wires": []
    },
    {
        "id": "d5c74ba7671b8c83",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "",
        "func": "msg.topic = \"INSERT INTO smartwater\n(dev_id, level, rainfall, flow)\" + \nVALUES(\" + \nmsg.payload.dev_id + \",\" +\nmsg.payload.level + \",\" +\nmsg.payload.rainfall + \",\" +\nmsg.payload.flow +\n\")\"\\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 420,
        "y": 400,
        "wires": [
            [
                "8120b93037058f0b",
                "5bddcf457244127"
            ]
        ]
    },
    {
        "id": "a87789c5831a96ee",
        "type": "json",
        "z": "5b890ccf.cfd994",
        "name": "",
        "property": "payload",
        "action": "",
        "pretty": false,
        "x": 270,
```

```
        "y": 400,
        "wires": [
            [
                "d5c74ba7671b8c83",
                "d2b73f6ee32154e0",
                "2c3ad4b412708bcd"
            ]
        ]
    },
    {
        "id": "902b1c025d5c11f1",
        "type": "websocket in",
        "z": "5b890ccf.cfd994",
        "name": "",
        "server": "4f1951193068098c",
        "client": "",
        "x": 120,
        "y": 400,
        "wires": [
            [
                "a87789c5831a96ee"
            ]
        ]
    },
    {
        "id": "7103f54b72a2c9a3",
        "type": "comment",
        "z": "5b890ccf.cfd994",
        "name": "HTTP Server",
        "info": "",
        "x": 110,
        "y": 40,
        "wires": []
    },
    {
        "id": "199a3a91f0130ffb",
        "type": "comment",
        "z": "5b890ccf.cfd994",
        "name": "MQTT Server",
        "info": "",
        "x": 110,
        "y": 200,
        "wires": []
    },
    {
        "id": "0c72fbcf385a0d9e",
        "type": "comment",
        "z": "5b890ccf.cfd994",
        "name": "Websocket",
        "info": "",
        "x": 100,
        "y": 360,
        "wires": []
    }
```

```
},
{
  "id": "60b8038414a1adcf",
  "type": "ui_gauge",
  "z": "5b890ccf.cfd994",
  "name": "",
  "group": "3cd2e5646bddc1d5",
  "order": 0,
  "width": 0,
  "height": 0,
  "gtype": "gage",
  "title": "Rain Gauge",
  "label": "mm/10 minutes",
  "format": "{{{value}}}",
  "min": 0,
  "max": 10,
  "colors": [
    "#00b500",
    "#e6e600",
    "#ca3838"
  ],
  "seg1": "",
  "seg2": "",
  "className": "",
  "x": 630,
  "y": 480,
  "wires": []
},
{
  "id": "5d52fd32f61a070a",
  "type": "ui_chart",
  "z": "5b890ccf.cfd994",
  "name": "",
  "group": "3cd2e5646bddc1d5",
  "order": 1,
  "width": 0,
  "height": 0,
  "label": "Water Level (cm)",
  "chartType": "line",
  "legend": "false",
  "xformat": "HH:mm:ss",
  "interpolate": "step",
  "nodata": "",
  "dot": false,
  "ymin": "",
  "ymax": "",
  "removeOlder": 1,
  "removeOlderPoints": "",
  "removeOlderUnit": "3600",
  "cutout": 0,
  "useOneColor": false,
  "useUTC": false,
  "colors": [
```

```
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 650,
    "y": 440,
    "wires": [
        []
    ]
},
{
    "id": "d2b73f6ee32154e0",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj1 = msg.payload;\nmsg.payload =\nobj1.level;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 440,
    "wires": [
        [
            "5d52fd32f61a070a"
        ]
    ]
},
{
    "id": "2c3ad4b412708bcd",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj = msg.payload;\nmsg.payload =\nobj.rainfall;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 480,
```

```
        "wires": [
            [
                "60b8038414a1adcf"
            ]
        ]
    },
{
    "id": "912098f194a76ef2",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj = msg.payload;\nmsg.payload =\nobj.rainfall;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 560,
    "wires": [
        [
            "f024e5c8a52acc8a"
        ]
    ]
},
{
    "id": "034d5e39764ad47b",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj = msg.payload;\nmsg.payload =\nobj.flow;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 600,
    "wires": [
        [
            "62a7521022e1cb2e"
        ]
    ]
},
{
    "id": "62a7521022e1cb2e",
    "type": "ui_gauge",
    "z": "5b890ccf.cfd994",
    "name": "",
    "group": "8c341cdd237f621f",
    "order": 2,
```

```
        "width": 0,
        "height": 0,
        "gtype": "gage",
        "title": "Water Flow",
        "label": "l/s",
        "format": "{{value}}",
        "min": 0,
        "max": "1000",
        "colors": [
            "#00b500",
            "#e6e600",
            "#ca3838"
        ],
        "seg1": "",
        "seg2": "",
        "className": "",
        "x": 630,
        "y": 600,
        "wires": []
    },
    {
        "id": "f024e5c8a52acc8a",
        "type": "ui_chart",
        "z": "5b890ccf.cfd994",
        "name": "",
        "group": "8c341cdd237f621f",
        "order": 1,
        "width": 0,
        "height": 0,
        "label": "Rainfall",
        "chartType": "bar",
        "legend": "false",
        "xformat": "HH:mm:ss",
        "interpolate": "linear",
        "nodata": "",
        "dot": false,
        "ymin": "",
        "ymax": "",
        "removeOlder": 1,
        "removeOlderPoints": "",
        "removeOlderUnit": "3600",
        "cutout": 0,
        "useOneColor": false,
        "useUTC": false,
        "colors": [
            "#1f77b4",
            "#aec7e8",
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#8c564b",
            "#e41a1c"
        ]
    }
]
```

```
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 620,
    "y": 560,
    "wires": [
        []
    ]
},
{
    "id": "932a9b8571b65766",
    "type": "ui_chart",
    "z": "5b890ccf.cfd994",
    "name": "",
    "group": "8c341cdd237f621f",
    "order": 2,
    "width": 0,
    "height": 0,
    "label": "Water Level",
    "chartType": "line",
    "legend": "false",
    "xformat": "HH:mm:ss",
    "interpolate": "linear",
    "nodata": "",
    "dot": false,
    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 630,
    "y": 520,
    "wires": [
        []
    ]
}
```

```
        ],
    },
{
    "id": "5ffbb0b1b58f11eb",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj = msg.payload;\nmsg.payload =\nobj.level;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 520,
    "wires": [
        [
            [
                "932a9b8571b65766"
            ]
        ]
    ],
},
{
    "id": "9b7b061f33dea219",
    "type": "http in",
    "z": "5b890ccf.cfd994",
    "name": "Get Data",
    "url": "/flood/node1",
    "method": "get",
    "upload": false,
    "swaggerDoc": "",
    "x": 100,
    "y": 80,
    "wires": [
        [
            [
                "cfb8d82279080505",
                "d2b73f6ee32154e0",
                "2c3ad4b412708bcd"
            ]
        ]
    ],
},
{
    "id": "1d600e624990207c",
    "type": "mqtt out",
    "z": "5b890ccf.cfd994",
    "name": "",
    "topic": "flood/node2/led",
    "qos": "0",
    "retain": "",
    "respTopic": "",
    "contentType": "",
    "userProps": "",
    "correl": "",
```

```

    "expiry": "",
    "broker": "f9a971b7c57d8b6c",
    "x": 200,
    "y": 580,
    "wires": []
},
{
    "id": "1a2c7fcraf9cb07da",
    "type": "function",
    "z": "5b890ccf.cfd994",
    "name": "",
    "func": "var obj = msg.payload;\nmsg.payload =\nobj.rainfall;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 420,
    "y": 700,
    "wires": [
        [
            "87b3ee63d26ce078"
        ]
    ]
},
{
    "id": "87b3ee63d26ce078",
    "type": "ui_chart",
    "z": "5b890ccf.cfd994",
    "name": "",
    "group": "68d767ed82174033",
    "order": 1,
    "width": 0,
    "height": 0,
    "label": "Rainfall",
    "chartType": "bar",
    "legend": "false",
    "xformat": "HH:mm:ss",
    "interpolate": "linear",
    "nodata": "",
    "dot": false,
    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",

```

```
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 620,
    "y": 700,
    "wires": [
        []
    ]
},
{
    "id": "b9ca3b469e9ca168",
    "type": "ui_chart",
    "z": "5b890ccf.cfd994",
    "name": "",
    "group": "68d767ed82174033",
    "order": 2,
    "width": 0,
    "height": 0,
    "label": "Water Level",
    "chartType": "line",
    "legend": "false",
    "xformat": "HH:mm:ss",
    "interpolate": "linear",
    "nodata": "",
    "dot": false,
    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
```

```
        "useDifferentColor": false,
        "className": "",
        "x": 630,
        "y": 660,
        "wires": [
            []
        ]
    },
    {
        "id": "cdcf79b2260f5497",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "",
        "func": "var obj = msg.payload;\nmsg.payload =\nobj.level;\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 420,
        "y": 660,
        "wires": [
            [
                "b9ca3b469e9ca168"
            ]
        ]
    },
    {
        "id": "1f2356cbc8d15404",
        "type": "function",
        "z": "5b890ccf.cfd994",
        "name": "",
        "func": "var obj = msg.payload;\nmsg.payload =\nobj.flow;\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 420,
        "y": 740,
        "wires": [
            [
                "b1764384d33fa351"
            ]
        ]
    },
    {
        "id": "b1764384d33fa351",
        "type": "ui_gauge",
        "z": "5b890ccf.cfd994",
        "name": "",
```

```
"group": "68d767ed82174033",
"order": 2,
"width": 0,
"height": 0,
"gttype": "gage",
"title": "Water Flow",
"label": "l/s",
"format": "{{value}}",
"min": 0,
"max": "1000",
"colors": [
    "#00b500",
    "#e6e600",
    "#ca3838"
],
"seg1": "",
"seg2": "",
"className": "",
"x": 630,
"y": 740,
"wires": []
},
{
    "id": "31e8a2aae5237f55",
    "type": "ui_switch",
    "z": "5b890ccf.cfd994",
    "name": "led button",
    "label": "LED",
    "tooltip": "",
    "group": "68d767ed82174033",
    "order": 3,
    "width": 0,
    "height": 0,
    "passthru": true,
    "decouple": "false",
    "topic": "flood/node1/led",
    "topicType": "msg",
    "style": "",
    "onvalue": "true",
    "onvalueType": "bool",
    "onicon": "",
    "oncolor": "",
    "offvalue": "false",
    "offvalueType": "bool",
    "officon": "",
    "offcolor": "",
    "animate": false,
    "className": "",
    "x": 100,
    "y": 520,
    "wires": [
        [
            "1d600e624990207c"
    ]
]
```

```
        ]
    ]
},
{
  "id": "d770c87012ef53a7",
  "type": "mqtt in",
  "z": "5b890ccf.cfd994",
  "name": "",
  "topic": "flood/node2",
  "qos": "2",
  "datatype": "auto",
  "broker": "f9a971b7c57d8b6c",
  "nl": false,
  "rap": false,
  "x": 110,
  "y": 300,
  "wires": [
    [
      [
        "26095e1a8765435f"
      ]
    ]
  ],
  {
    "id": "fe8cc6b6.d28218",
    "type": "MySQLdatabase",
    "name": "",
    "host": "localhost",
    "port": "3306",
    "db": "banjir_db",
    "tz": "",
    "charset": "UTF8"
  },
  {
    "id": "f9a971b7c57d8b6c",
    "type": "mqtt-broker",
    "name": "My Broker",
    "broker": "localhost",
    "port": "1883",
    "clientid": "smarthome28",
    "usetls": false,
    "protocolVersion": "4",
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willPayload": ""
  }
]
```

```
        "willPayload": "",  
        "willMsg": {},  
        "sessionExpiry": ""  
    },  
    {  
        "id": "4f1951193068098c",  
        "type": "websocket-listener",  
        "path": "/flood/node1",  
        "wholemsg": "false"  
    },  
    {  
        "id": "3cd2e5646bddc1d5",  
        "type": "ui_group",  
        "name": "Node 1",  
        "tab": "9c0d3f283c093e0f",  
        "order": 1,  
        "disp": true,  
        "width": "6",  
        "collapse": false,  
        "className": ""  
    },  
    {  
        "id": "8c341cdd237f621f",  
        "type": "ui_group",  
        "name": "Node 1-1",  
        "tab": "9c0d3f283c093e0f",  
        "order": 2,  
        "disp": true,  
        "width": "6",  
        "collapse": false,  
        "className": ""  
    },  
    {  
        "id": "68d767ed82174033",  
        "type": "ui_group",  
        "name": "Node 2",  
        "tab": "9c0d3f283c093e0f",  
        "order": 3,  
        "disp": true,  
        "width": "6",  
        "collapse": false,  
        "className": ""  
    },  
    {  
        "id": "9c0d3f283c093e0f",  
        "type": "ui_tab",  
        "name": "Multi-Protocol MisREd",  
        "icon": "dashboard",  
        "disabled": false,  
        "hidden": false  
    }  
]
```

4. Buka Arduino IDE, install library **JSON 5**. Setelah itu, upload script program berikut ke ESP32 untuk melakukan transmisi data dummy menuju Node-Red menggunakan protokol HTTP metode Get.

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "GANTI DENGAN SSID TATHERING";
const char* password = "PASSWD TATHERING";

//sesuaikan dengan IP dan path masing-masing server
String serverName = "http://192.168.0.124:1880/flood/node1";

unsigned long lastTime = 0;
// Set timer to 5 seconds (5000)
unsigned long timerDelay = 5000;

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

  Serial.println("Timer set to 5 seconds (timerDelay variable), it will take 5 seconds before publishing the first reading.");
}

void loop() {
  //Send an HTTP Get request every 5 detik
  float dev_id=28, level=5, rainfall=10.2, flow=12;
  if ((millis() - lastTime) > timerDelay) {
    //Check WiFi connection status
    if(WiFi.status()== WL_CONNECTED){
      HTTPClient http;

      String serverPath = serverName + "?dev_id=" + dev_id + "&level=" + level + "&rainfall=" + rainfall + "&flow=" + flow;

      // Your Domain name with URL path or IP address with path
      http.begin(serverPath.c_str());

      // Send HTTP GET request
      int httpResponseCode = http.GET();

      if (httpResponseCode>0) {
        Serial.print("HTTP Response code: ");
      }
    }
  }
}
```

```

        Serial.println(httpResponseCode);
        String payload = http.getString();
        Serial.println(payload);
    }
    else {
        Serial.print("Error code: ");
        Serial.println(httpResponseCode);
    }
    //Free resources
    http.end();
}
else {
    Serial.println("WiFi Disconnected");
}
lastTime = millis();
}
}

```

5. Buka serial monitor, debug pada Node Red, dan dashboard Node-Red, kemudian dokumentasikan outputnya.
6. Buatlah flow chart program ESP32 diatas.
7. Buka Arduino IDE dan upload script program berikut ke ESP32 untuk melakukan transmisi data dummy menuju Node-Red menggunakan protokol HTTP metode POST.

```

#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "GANTI DENGAN SSID TATHERING";
const char* password = "PASSWD TATHERING";

//Your Domain name with URL path or IP address with path
const char* serverName = "http://192.168.0.124:1880/flood/node1";

// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an int.
unsigned long lastTime = 0;
//Timer set to 10 minutes (600000)
//unsigned long timerDelay = 600000;
// Set timer to 5 seconds (5000)
unsigned long timerDelay = 5000;

void setup() {
    Serial.begin(115200);

    WiFi.begin(ssid, password);
    Serial.println("Connecting");
    while(WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
}

```

```

Serial.print("Connected to WiFi network with IP Address: ");
Serial.println(WiFi.localIP());

Serial.println("Timer set to 5 seconds (timerDelay variable), it will take 5 seconds before
publishing the first reading.");
}

void loop() {
//Send an HTTP POST request every 10 minutes
String dev_id="28", level="8", rainfall="9.2", flow="10";
if ((millis() - lastTime) > timerDelay) {
//Check WiFi connection status
if(WiFi.status()== WL_CONNECTED){
WiFiClient client;
HTTPClient http;

// Your Domain name with URL path or IP address with path
http.begin(client, serverName);

// Specify content-type header
//http.addHeader("Content-Type", "application/x-www-form-urlencoded");
// Data to send with HTTP POST
//String httpRequestData =
"api_key=tPmAT5Ab3j7F9&sensor=BME280&value1=24.25&value2=49.54&value3=1005.14";
// Send HTTP POST request
//int httpResponseCode = http.POST(httpRequestData);

// If you need an HTTP request with a content type: application/json, use the following:
http.addHeader("Content-Type", "application/json");
// JSON data to send with HTTP POST
String httpRequestData = "{\"dev_id\":\"" + dev_id + "\",\"level\":\"" + level +
"\",\"rainfall\":\"" + rainfall + "\",\"flow\":\"" + flow + "\"}";
// Send HTTP POST request
int httpResponseCode = http.POST(httpRequestData);

Serial.print("HTTP Response code is: ");
Serial.println(httpResponseCode);
http.end();
}
else {
Serial.println("WiFi Disconnected");
}
lastTime = millis();
}
}

```

8. Buka serial monitor, debug pada Node Red, dan dashboard Node-Red, kemudian dokumentasikan outputnya.
9. Buatlah flow chart program ESP32 diatas.

C. Transmisi Data Menggunakan Protokol MQTT

1. Buatlah program seperti script dibawah ini, kemudian upload program tersebut ke ESP32.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

// Update these with values suitable for your network.

const char* ssid = "GANTI DENGAN SSID TATHERING";
const char* password = "PASSWD TATHERING";
const char* mqtt_server = "192.168.0.124";

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {

    delay(10);
// We start by connecting to a WiFi network
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

randomSeed(micros());

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void reconnect() {
// Loop until we're reconnected
```

```
while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
        Serial.println("connected");
    } else {
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" try again in 5 seconds");
        // Wait 5 seconds before retrying
        delay(5000);
    }
}

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);

}

void loop() {

    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    char payload[200];

    /* for JSON6
    StaticJsonBuffer<200> doc;
    doc["dev_id"] = 28;
    doc["level"] = 25;
    doc["rainfall"] = 5.25;
    doc["flow"] = 10;
```

```

    serializeJson(doc, payload); */

StaticJsonBuffer<200> jsonBuffer;
JsonObject& doc = jsonBuffer.createObject();

doc["dev_id"] = 28;
doc["level"] = 25;
doc["rainfall"] = 5.25;
doc["flow"] = 10;

doc.printTo(payload);

client.publish("flood/node1", payload);
delay(10000);
}

```

2. Buka serial monitor, debug pada Node Red, dan dashboard Node-Red, kemudian dokumentasikan outputnya.
3. Buatlah flow chart dari program di atas.

D. Akuisi Data dan Kendali Perangkat IoT Menggunakan Protokol MQTT

1. Buatlah program seperti script dibawah ini, kemudian upload program tersebut ke ESP32.

```

#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

/* change it with your ssid-password */
const char* ssid = "GANTI DENGAN SSID TATHERING";
const char* password = "PASSWD TATHERING";
/* this is the IP of PC/raspberry where you installed MQTT Server
on Wins use "ipconfig"
on Linux use "ifconfig" to get its IP address */
const char* mqtt_server = "192.168.0.124";
float temperature = 0;

/* create an instance of PubSubClient client */
WiFiClient espClient;
PubSubClient client(espClient);

```

```
/*LED GPIO pin*/
const char led = 2;

/* topics */
#define TEMP_TOPIC "flood/node2"
#define LED_TOPIC "flood/node2/led" /* true=on, false=off */

long lastMsg = 0;
char msg[20];

void receivedCallback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message received: ");
    Serial.println(topic);

    Serial.print("payload: ");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();
    /* we got '1' -> on */
    if ((char)payload[0] == 't') {
        digitalWrite(led, HIGH);
    } else {
        /* we got '0' -> on */
        digitalWrite(led, LOW);
    }
}

void mqttconnect() {
    /* Loop until reconnected */
    while (!client.connected()) {
        Serial.print("MQTT connecting ...");
        /* client ID */
        String clientId = "ESP32Client";
        /* connect now */
        if (client.connect(clientId.c_str())) {
            Serial.println("connected");
            /* subscribe topic with default QoS 0*/
            client.subscribe(LED_TOPIC);
```

```
        } else {
            Serial.print("failed, status code =");
            Serial.print(client.state());
            Serial.println("try again in 5 seconds");
            /* Wait 5 seconds before retrying */
            delay(5000);
        }
    }
}

void setup() {
    Serial.begin(115200);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    /* set led as output to control led on-off */
    pinMode(led, OUTPUT);

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    /* configure the MQTT server with IPaddress and port */
    client.setServer(mqtt_server, 1883);
    /* this receivedCallback function will be invoked
     * when client received subscribed topic */
    client.setCallback(receivedCallback);
}
void loop() {
    /* if client was disconnected then try to reconnect again */
    if (!client.connected()) {
        mqttconnect();
    }
}
```

```

}

/* this function will listen for incoming
subscribed topic-process-invoke receivedCallback */
client.loop();
/* we measure temperature every 3 secs
we count until 3 secs reached to avoid blocking program if using delay()*/
long now = millis();
if (now - lastMsg > 5000) {
    lastMsg = now;

    char payload[200];
    StaticJsonBuffer<200> jsonBuffer;
    JsonObject& doc = jsonBuffer.createObject();

    doc["dev_id"] = 28;
    doc["level"] = 25;
    doc["rainfall"] = 5.25;
    doc["flow"] = 10;

    doc.printTo(payload);
    /* publish the message */
    client.publish(TEMP_TOPIC, payload);
}

}

```

2. Buka serial monitor dan Dashboard Node-Red, kemudian klik tombol switch pada dashboard untuk mengendalikan nyala LED.
3. Dokumentasikan outputnya, kemudian buatlah flow chart program diatas.
4. Buatlah program seperti script dibawah ini untuk membuat control LED ESP32 menggunakan suara.
5. Install library Adafruit IO Arduino, agar program dapat diverifikasi.

```

#include <WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

#define WLAN_SSID      "SSID TATHERING"
#define WLAN_PASS      "PASSWD"
#define AIO_SERVER     "io.adafruit.com"
#define AIO_SERVERPORT 1883

```

```

#define AIO_USERNAME "Apriantoro28"
#define AIO_KEY      "f70f9dd6b6754a778f357491c25b9fe5"

int output=2;

WiFiClient client; // Create an ESP8266 WiFiClient class to connect to the MQTT server.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Subscribe led = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/led");

void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);

  // Connect to WiFi access point.

  Serial.println();
  Serial.println();

  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);

  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi connected");
  Serial.println("IP address: "); Serial.println(WiFi.localIP());
  mqtt.subscribe(&led);
}

uint32_t x=0;

void loop() {

MQTT_connect();

```

```

Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &led) {
        Serial.print(F("Got: "));
        Serial.println((char *)led.lastread);
        if (!strcmp((char *) led.lastread, "1"))
        {
            digitalWrite(2, HIGH);
        }
        else
        {
            digitalWrite(2, LOW);
        }
    }
}

void MQTT_connect() {

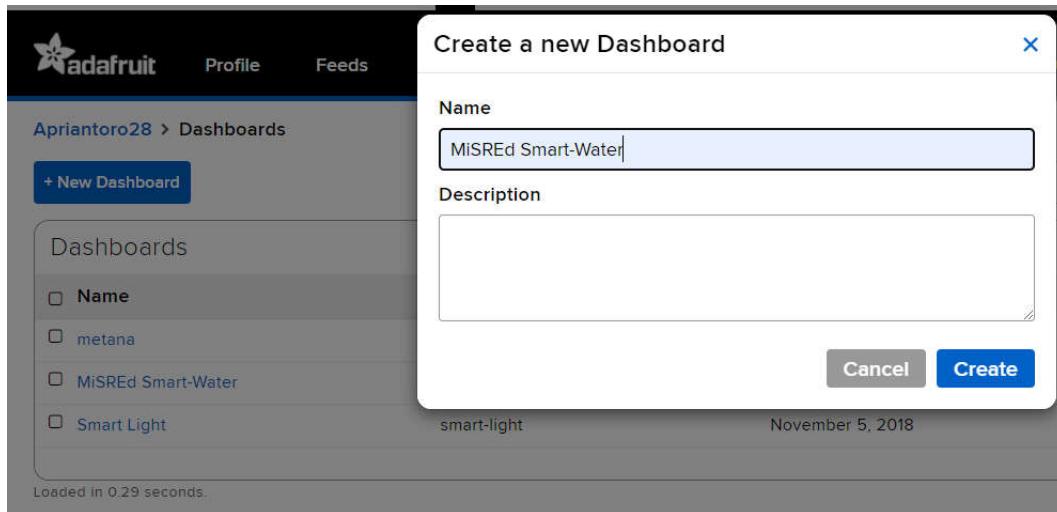
    int8_t ret;
    // Stop if already connected.
    if (mqtt.connected()) {
        return;
    }
    Serial.print("Connecting to MQTT... ");

    uint8_t retries = 3;

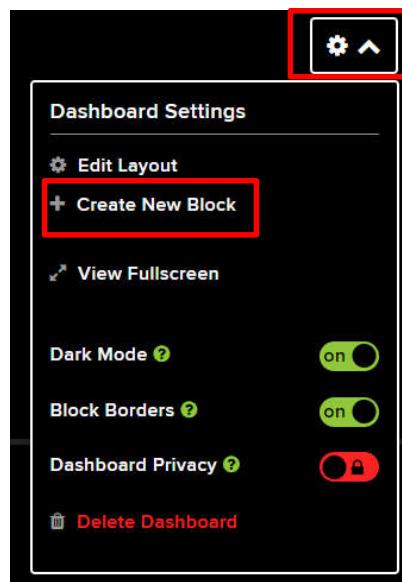
    while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
        Serial.println(mqtt.connectErrorString(ret));
        Serial.println("Retrying MQTT connection in 5 seconds...");
        mqtt.disconnect();
        delay(5000); // wait 5 seconds
        retries--;
        if (retries == 0) {
            // basically die and wait for WDT to reset me
            while (1);
        }
    }
    Serial.println("MQTT Connected!");
}

```

6. Buatlah akun pada laman **adafruit.io**
7. Kemudian buatlah dashboard seperti gambar berikut.



8. Setelah itu, buka dashboard dengan cara klik pada nama dashboard yang telah dibuat.
9. Klik icon gear (setting), pilih create new block.

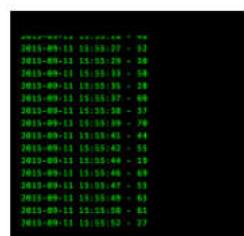
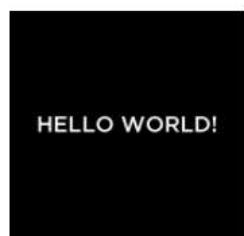
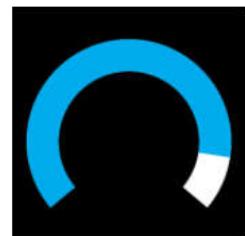
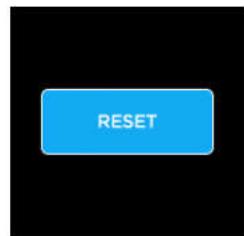
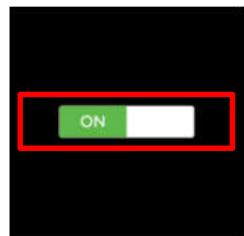


10. Pilih **widget ON/OFF switch**, kemudian buat name dengan nama **led**.

Create a new block

X

Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.



Default

v

Feed Name	Last value	Recorded	
<input type="checkbox"/> arus	0.16	almost 3 years	🔒
<input type="checkbox"/> image		about 2 years	🔒
<input type="checkbox"/> led	0	5 minutes	🔒
<input type="checkbox"/> photocell	4705	over 2 years	🔒
<input type="checkbox"/> Relay1	0	11 minutes	🔒
<input type="checkbox"/> Relay2		almost 3 years	🔒
<input type="checkbox"/> Relay3		almost 3 years	🔒
<input type="checkbox"/> Relay4		almost 3 years	🔒
<input type="checkbox"/> Water Level		about 24 hours	🔒

led

Create

11. Konfigurasi widget On/Off seperti berikut.

Block settings

X

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)

LED Control

Button On Text

ON

Button On Value (uses On Text if blank)

1

Button Off Text

OFF

Button Off Value (uses Off Text if blank)

0

Block Preview

LED Control

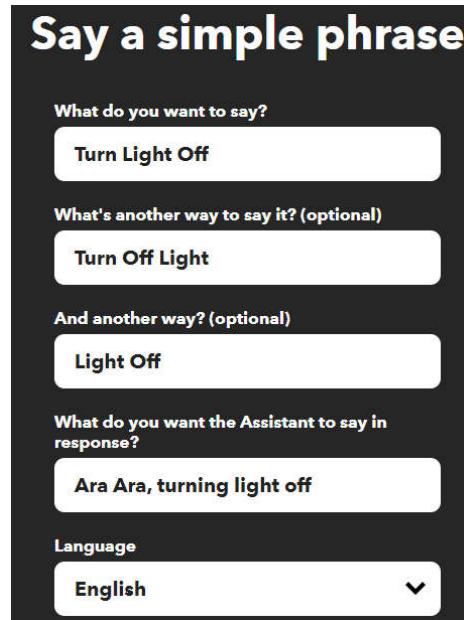


Toggle A toggle button is useful if you have an ON or OFF type of state. You can

12. Buka **ifttt.com** , kemudian login atau buat akun menggunakan email Google yang tersambung dengan Smartphone Android.
13. Setelah login, klik menu **Create** untuk membuat applet. Klik pada tulisan **If This**, kemudian pada kolom pencarian, ketikkan **Google Assistant**.

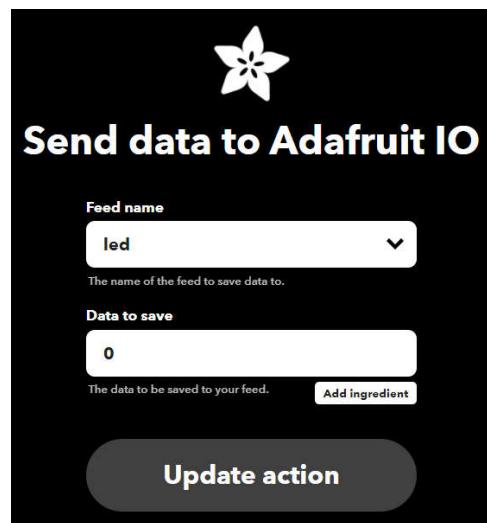
The screenshot shows the IFTTT interface. At the top, there's a navigation bar with the IFTTT logo, a 'Win a year of Pro+' button, and links for 'My Applets', 'Explore', 'Developers', and a 'Create' button which is highlighted with a red box. Below this, the main area is titled 'My Applets' and features a large 'Create your own' button. Underneath is a large grey button divided into two sections: 'If This' on top and 'Then That' on the bottom, separated by a vertical line.

14. Buatlah konfigurasi seperti pada gambar berikut.



15. Kemudian pada **Then That**, pilih **Adafruit.io**.

16. Buatlah konfigurasi seperti gambar berikut.



17. Ulangi langkah 13 hingga 16 untuk membuat applet yang berfungsi mengendalikan LED agar menyala, dengan mengubah parameter OFF menjadi ON, dan logika 0 menjadi 1.

18. Buka Google Assistant, kemudian pastikan akun Google yang digunakan sudah sesuai dengan ifttt.com.

19. Ucapkan perintah sesuai dengan konfigurasi yang telah dibuat.

20. Dokumentasikan outputnya dan buatlah flowchart program tersebut.

7. PERTANYAAN DAN TUGAS

1. Buatlah Multi-Protocol Server (HTTP dan MQTT) pada Node-Red untuk aplikasi Smart-Home. Pada dashboard terdiri dari 2 Node, Node 1 untuk menampilkan data monitoring suhu dan kelembapan menggunakan sensor DHT 11. Pada Node 2 dikhusukan untuk kendali lampu LED sebanyak 5 buah. Protokol MQTT digunakan untuk mengirim data dan mengendalikan lampu LED.

1. NO. JOBSHEET : 5

2. JUDUL : PEMROGRAMAN DASAR NODE-RED

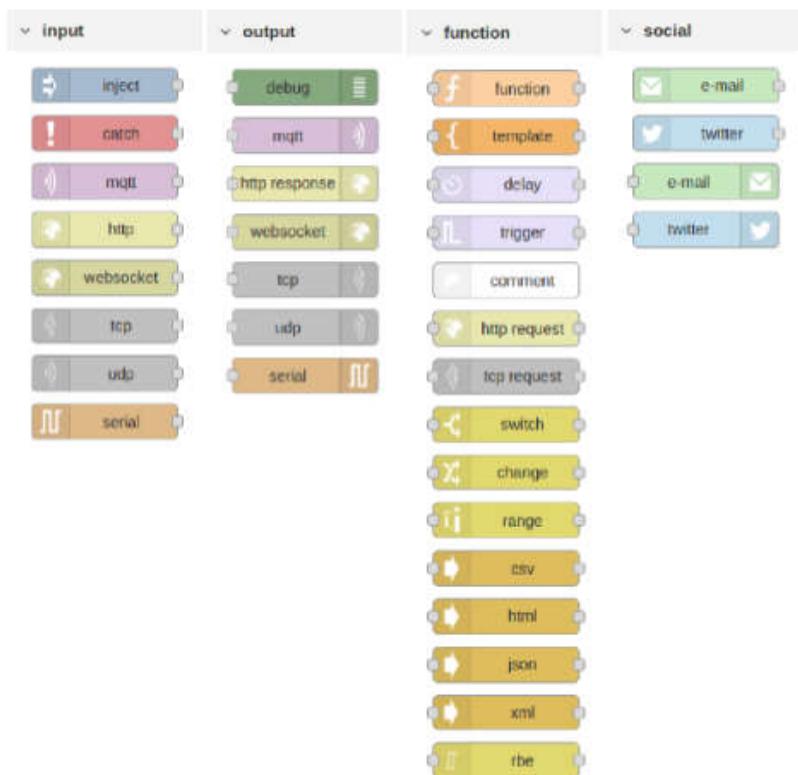
3. TUJUAN

- 1) Mahasiswa dapat memahami dasar pemrograman pada Node-Red.
- 2) Mahasiswa dapat membuat application server sederhana menggunakan Node-Red.

4. ALAT DAN BAHAN

- 1) Perangkat terpasang Node-Red

5. TEORI SINGKAT



Gambar 1. Bagian-bagian node pada Node-Red

Node-RED adalah alat pemrograman berbasis grafis untuk menyatukan perangkat keras, API, dan layanan online dengan cara baru dan menarik. Node-Red menyediakan editor berbasis browser yang membuatnya mudah untuk menyatukan flow program menggunakan berbagai node dalam palet yang dapat digunakan ke runtime dalam satu klik. Node-Red mempunyai banyak fitur yang mana salah satunya dapat digunakan sebagai backend server IoT. Bahasa pemrograman yang digunakan oleh Node-Red adalah JavaScript.

Ketika Node-Red telah terpasang pada perangkat seperti Komputer, Raspberry Pi atau perangkat lainnya, pemrograman dapat dimulai dengan satu set default node. Ada delapan kategori utama node dalam instalasi default: input, output, fungsi, sosial, penyimpanan, analisis, advanced dan Raspberry Pi. Hal tersebut dapat dilihat pada Gambar 1.

6. LANGKAH KERJA

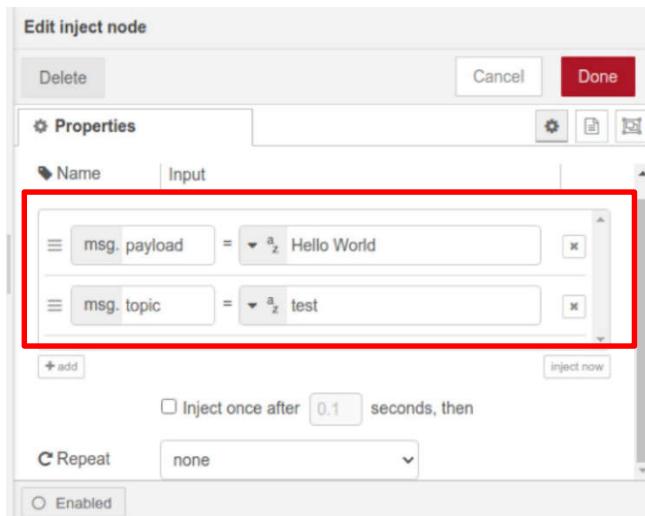
A. Basic Flow

- 1) Pastikan perangkat komputer terpasang Node-Red.
- 2) Buatlah Basic Flow seperti pada Gambar 2.



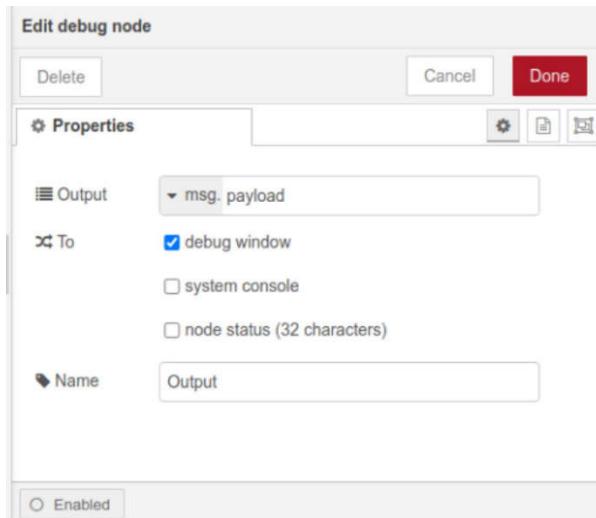
Gambar 2. Basic flow programming

- 3) Double klik pada node input, kemudian konfigurasi seperti pada Gambar 3.



Gambar 3. Konfigurasi inject/input node

- 4) Lakukan hal yang sama seperti pada langkah 3 untuk node output / debug node. Hal tersebut dapat dilihat pada Gambar 4.



Gambar 4. Konfigurasi debug/output node

- 5) Klik tombol “Deploy” yang terletak pada pojok kanan atas UI untuk menjalankan program.
- 6) Dokumentasikan output dari program tersebut.

B. Menggunakan Function Node

- 1) Buatlah flow fungsi output tunggal seperti pada Gambar 5.



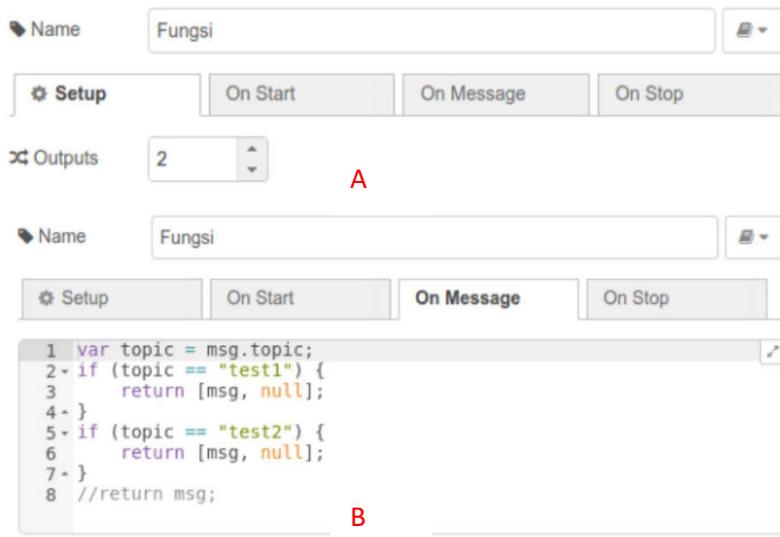
Gambar 5. Flow fungsi output tunggal

- 2) Konfigurasikan node Input1. Isikan payload dengan “Hello World”, dan topic berupa “test1”.
- 3) Deploy program dan dokumentasikan hasilnya.
- 4) Buatlah flow fungsi output berganda yang berfungsi memisahkan pesan seperti Gambar 6.



Gambar 6. Flow fungsi output berganda

- 5) Konfigurasikan node Input2. Isikan payload dengan “Expeliarmus”, dan topic berupa “test2”.
- 6) Konfigurasikan node fungsi seperti Gambar 7.

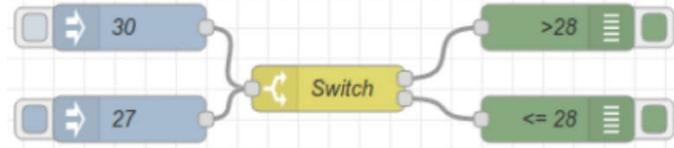


Gambar 7. Flow fungsi output berganda

- 7) Deploy program dan dokumentasikan hasilnya.

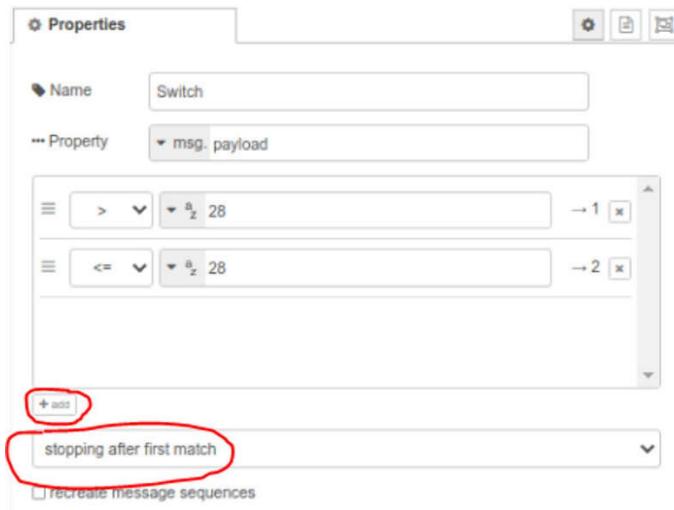
C. Menggunakan Switch Node

- 1) Buatlah flow seperti pada Gambar 8.



Gambar 8. Flow switch node

- 2) Konfigurasi Input/Inject Node. Isikan Payload pada Inject Node 1 dengan angka 28. Kemudian pada Inject Node 2, isikan Payload dengan angka 27.
- 3) Konfigurasi Switch Node seperti Gambar 9.



Gambar 9. Konfigurasi switch node

- 4) Deploy flow dan dokumentasikan hasilnya.

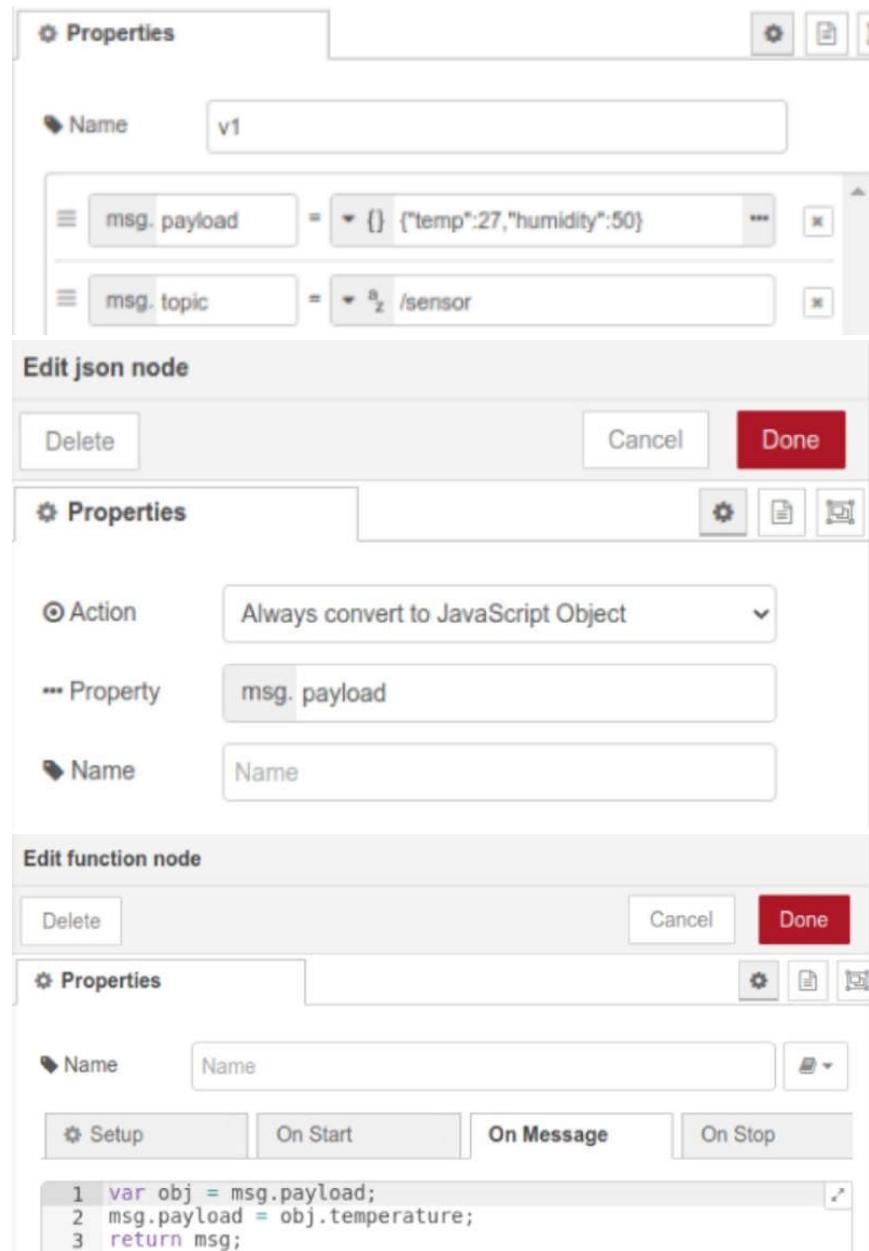
D. Menggunakan JSON Parsing

- 1) Buatlah flow seperti Gambar 10.



Gambar 10. Flow JSON Parser

- 2) Konfigurasikan Inject Node, JSON Parser Node, dan Function Node seperti Gambar 11.



Gambar 11. Konfigurasi inject node

- 3) Deploy flow dan dokumentasikan hasilnya.

7. PERTANYAAN & TUGAS

- 1) Buatlah arsitektur sistem IoT menggunakan semua fungsi yang ada pada Hands-On. Format data yang dikirim adalah multi-input dengan format data String, Boolean, Number dan JSON (3 data). Setiap input mempunyai ditampilkan dalam Debug Node yang berbeda.

- 1. NO. JOBSHEET : 6**
- 2. JUDUL** : TRANSMISI DATA MENGGUNAKAN MESSAGE QUEUING TELEMETRY TRANSPORT (MQTT) PROTOCOL

3. TUJUAN

- 1) Mahasiswa dapat memahami alur kerja, kegunaan dan manfaat protokol MQTT.
- 2) Mahasiswa dapat memahami dan mengimplementasikan protokol MQTT pada sistem IoT untuk monitoring dan kendali.

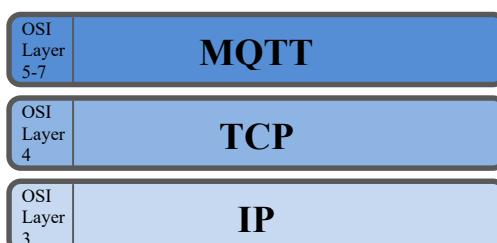
4. ALAT DAN BAHAN

- | | |
|------------------|--------------------|
| 1) ESP32 | 5) Sensor DHT11 |
| 2) Breadboard | 6) LED |
| 3) Kabel jumper | 7) Multimeter |
| 4) Potensiometer | 8) Resistor 1K Ohm |

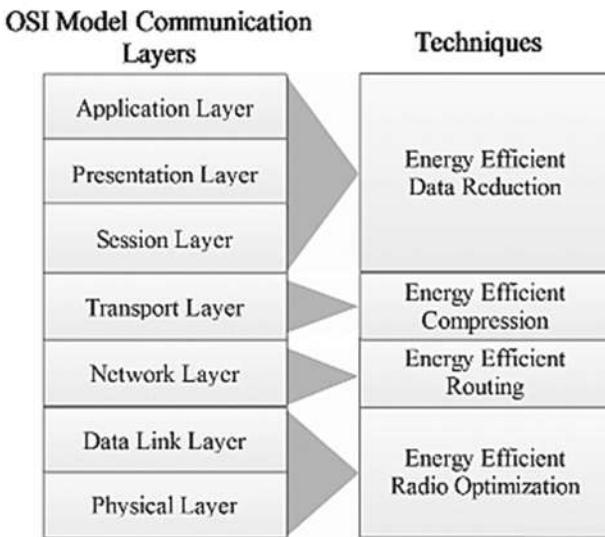
5. TEORI SINGKAT

Message Queuing Telemetry Transport (MQTT) adalah protokol komunikasi yang berjalan di atas *stack* TCP/IP, didesain untuk komunikasi *Machine-to-Machine* (M2M), bersifat *open sources* dan *lightweight*, mempunyai *protocol overhead* yang rendah (minimum 2 bytes) sehingga berefek pada konsumsi daya yang kecil dan mampu bekerja pada *latency* yang tinggi serta *bandwidth* yang kecil (Ullas *et al.*, 2014).

MQTT bekerja di atas TCP/IP *stack* di level *application layer* pada standar sistem OSI (Pal, Ghosh and Bhattacharya, 2017). Protokol ini menerapkan teknik kompresi data dan data *reduction* untuk melakukan efisiensi energi (Ali, Shah and Arshad, 2016). Hal tersebut dapat dilihat pada Gambar 5.1 dan Gambar 5.2.



Gambar 5.1. OSI *Layer* level protokol MQTT



Gambar 5.2. Teknik efisiensi energi pada OSI Layer
 (Sumber : Ali, Shah and Arshad, 2016)

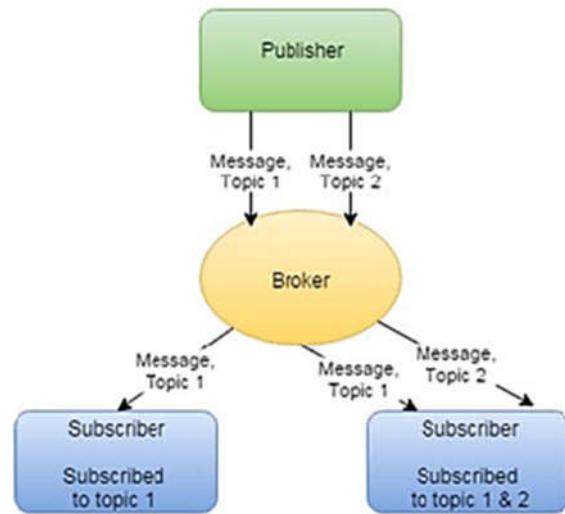
Menurut pendapat Pal, Ghosh dan Bhattacharya (2017) MQTT merupakan jenis protokol *data-agnostic* yang mana data yang mampu dikirimkan tidak terbatas pada jenis data tertentu, namun semua jenis data dapat dikirimkan seperti data *binary*, *text*, XML maupun JSON.

Protokol MQTT menggunakan model *publish/subscribe* untuk komunikasi data (Krishna *et al.*, 2017). Publisher mempunyai peran sebagai pengirim data atau *sender*, sedangkan *subscriber* berperan sebagai penerima data atau *receivers* dan terdapat broker yang berperan sebagai *middleman* yang bertugas meneruskan pesan (*message*) dari *publisher* menuju *subscriber*. Pesan di dalam broker dibedakan dan difilter menggunakan *topic* yang mana setiap pesan yang dikirimkan ke broker diberi *tag* atau penanda oleh *publisher* untuk membedakan setiap data (Pal, Ghosh and Bhattacharya, 2017).

Metode *publish/subscribe* mempunyai kentungan yaitu antara *publisher* dan *subscriber* bersifat *loosely coupled*, artinya mereka mampu tetap bekerja tanpa saling bergantung satu sama lain atau tidak saling terikat terutama pada tempat, waktu dan sinkronisasi. Pal, Ghosh dan Bhattacharya (2017) dalam penelitiannya menyebutkan bahwa *Publisher* dan *subscriber* tetap akan bekerja walaupun mereka tidak saling mengetahui letak atau keberadaan satu sama lain karena adanya *broker* diantara *publisher* dan *subscriber* atau sering disebut dengan *space decoupling*.

Time decoupling merupakan keuntungan yang lainnya dari metode ini, yang mana antara *publisher* dan *subscriber* tidak harus aktif atau terkoneksi bersamaan. *Publisher* tidak akan melakukan *blocking* ketika memproduksi sebuah *event* baru, misalnya *subscriber* bisa saja *disconnect* setelah melakukan *subscribe* ke *broker*, beberapa saat kemudian *subscriber connect* kembali ke *broker*, maka *subscriber* akan tetap menerima data yang sebelumnya tertunda (*pending*) (Pal, Ghosh and Bhattacharya, 2017). Metode tersebut disebut juga *mode offline*.

Synchronization decoupling adalah kondisi di mana pengaturan *event* baik itu penerimaan atau pengiriman pesan di sebuah *node* hingga tidak saling mengganggu satu sama lain (Alhakbani, Hassan and Ykhlef, 2017). Metode kerja *publish/subscribe* yang digunakan protokol MQTT dapat dilihat pada Gambar 5.3.



Gambar 5.3. Cara kerja *publish/subscribe*

Khrishna *et al* (2017) dalam penelitiannya menyebutkan bahwa metode *publish/subscribe* seperti yang ditunjukkan Gambar 2.4 sangat kontras jika dibandingkan dengan metode komunikasi data menggunakan *client/server* tradisional yang mana antara pengirim dan penerima saling bergantung satu sama lain (*tightly coupled*). Server tidak dapat mengirim data jika *client* tidak aktif secara bersamaan atau sedang tidak melakukan *listening* pada suatu *topic* tertentu. Metode tersebut mulai ditinggalkan karena sudah tidak relevan untuk konsep komunikasi IoT yang menuntut data dikirim secara *realtime* dan akurat walaupun *sender* dan *receiver* berada di daerah dengan koneksi yang *intermittent* atau *unreliable*.

Keuntungan yang lain dari penggunaan metode *publish/subscribe* yaitu metode ini bersifat *scalable*, artinya kita dapat menambahkan broker lain ke dalam sistem IoT untuk mengurangi beban pada broker tunggal dengan cara membagi *topic* di antara kedua broker tersebut (Krishna *et al.*, 2017).

Kekurangan dari penggunaan metode ini yaitu meskipun bersifat *loosely coupled* terkait dengan waktu, tempat dan sinkronisasi, namun di sisi pengiriman data metode ini bersifat *tightly coupled*. Menurut Khrishna *et al* (2017), jika terdapat perubahan yang dilakukan pada arah pengiriman data, misalnya perubahan *channel* dimana data akan disimpan, maka semua *subscriber* harus dimodifikasi agar dapat melakukan *subscribe* pada *topic* terbaru.

Berdasarkan tujuan protokol MQTT agar mudah untuk diimplementasikan, terdapat beberapa sinyal kontrol yang didefinisikan oleh MQTT untuk berinteraksi, namun hanya ada enam sinyal kontrol yang menjadi bagian terpenting yang dipakai oleh *client*. Keenam sinyal kontrol tersebut adalah sebagai berikut.

1. *Connect*

Sinyal kontrol ini adalah pesan pertama yang harus merupakan paket *CONNECT* yang mana mengidentifikasi informasi tentang *client* dan *server* setelah koneksi jaringan antara *client* dan *server* terbentuk.

2. *Keep Alive*

Keep alive merupakan sebuah spesifikasi berapa lama sebuah *client* dapat aktif meskipun tanpa melakukan pengiriman sebuah data sebelum pada akhirnya melakukan *disconnecting*. Sinyal kontrol ini diatur pada saat *connect*.

3. *Disconnect*

Disconnect merupakan proses menunggu *client* untuk menyelesaikan pekerjaannya melakukan *subscribe*. Proses ini juga bagian dari TCP *session* untuk melakukan *terminate*.

4. *Subscribe*

Subscribe adalah proses mengirim sebuah pesan *SUBSCRIBE* dari *client* ke *server* untuk membuat sebuah proses *subscribe* pada satu atau beberapa *topic*.

5. *Unsubscribe*

Unsubscribe merupakan proses mengirim sebuah pesan *UNSUBSCRIBE* dari *client* ke *server* untuk membuat sebuah proses *unsubscribe* pada satu atau beberapa *topic*.

6. *Publish*

Publish adalah proses mengirim sebuah pesan *PUBLISH* dari *client* ke *server* atau dari *server* ke *client* untuk mengangkut sebuah pesan atau data.

MQTT mengizinkan setiap *user* untuk menggunakan *Quality of Service* (QoS) yang berbeda sesuai dengan kebutuhannya. Protokol ini mempunyai tiga level QoS yang mendefinisikan tiga level berbeda dalam menjamin atau memberikan garansi terhadap konsistensi pengiriman pesan. *Subscriber* dan *broker* menyediakan mekanisme penyimpanan dan pengiriman kembali pesan, sehingga hal ini meningkatkan konsistensi data akibat kegagalan jaringan, *restart* dari aplikasi maupun penyebab lainnya (Fathia and Said, 2017). Tingkatan QoS yang diterapkan oleh protokol MQTT adalah sebagai berikut.

1. QoS 0

Menurut Fathia dan Said (2017), QoS 0 dapat melakukan pengiriman pesan ke *broker* hanya sekali (*at most once / fire and forget*). Pesan yang terkirim tergantung dari *reliability stack* TCP atau tergantung dari *realibility* jaringan. Apabila terdapat pesan yang mengalami kegagalan dalam proses transmisi, maka *broker* tidak akan meminta untuk mengirimkan pesan kembali.

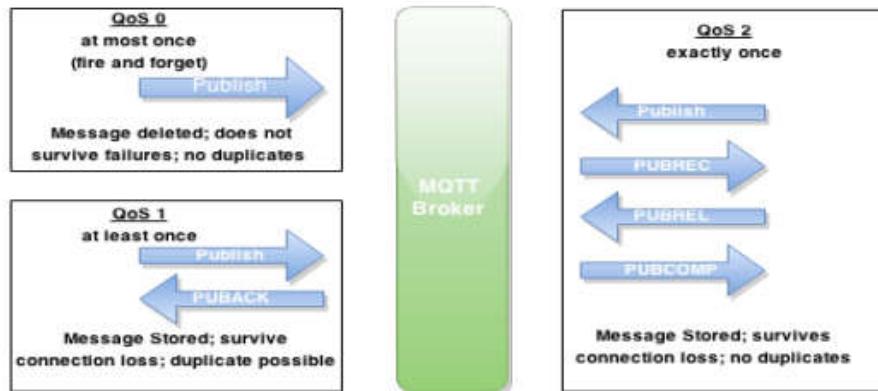
2. QoS 1

QoS 1 melakukan pengiriman pesan ke *broker* setidaknya satu kali (*at least one*) dengan pemberian sinyal kontrol ACK (*acknowledgment*) pada setiap pesan yang dikirimkan. Pesan yang dikirimkan dijamin akan sampai ke broker, namun setiap pesan tersebut tidak dijamin bersih dari adanya duplikasi (Fathia and Said, 2017).

3. QoS 2

Fathia dan Said (2017) mengatakan bahwa QoS 2 merupakan tingkatan tertinggi dari QoS MQTT. Pesan yang dikirimkan diterima satu kali (*exactly one*) dengan garansi semua pesan yang dikirim pasti diterima dan tidak ada pesan yang

terduplikasi. MQTT menggunakan ID pada setiap pesan untuk melakukan *filter* agar tidak ada pesan yang terduplikasi. Hal itu dapat dilihat pada Gambar 5.4.



Gambar 5.4. Proses komunikasi setiap level QoS protokol MQTT
(Sumber : <http://www.ossmendor.com/2015/04/internet-of-things-mqtt.html>)

Menurut Pal, Ghosh dan Bhattacharya (2017), pesan di dalam protokol MQTT mempunyai dua bagian yaitu *fixed part* dan *variable part*. *Fixed part* merupakan bagian penting untuk semua pesan, sedangkan *variable part* merupakan bagian yang dapat berubah tergantung dari pesan yang akan dikirim.

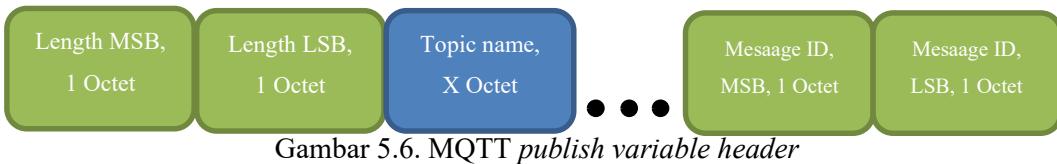
Fixed part terdiri dari dua *bytes* yaitu untuk *field* yang pertama terdiri dari jenis pesan (*type message*) akan terkirim, DUP Flag , level QoS yang digunakan dan *Retain*. Sedangkan untuk *field* yang kedua terdiri dari *Remaining Length* yang mengindikasikan jumlah oktet tersisa dari pesan yang mana termasuk dari variabel *header* dan *payload* (Pal, Ghosh and Bhattacharya, 2017). Bagian dari *fixed header* dapat dilihat pada Gambar 5.5.



Gambar 5.5. MQTT *fixed header*

Pal, Ghosh dan Bhattacharya (2017) berpendapat bahwa *Variable part* atau *variable header* merupakan bagian yang ukurannya dapat diubah. Bagian ini digunakan untuk keperluan *publish message* yang di dalamnya terdapat nama *topic* sebagai ID pesan. *Length field* pada *header* terdiri dari dua *bytes* dan

mengindikasikan nomor *octet* yang diperlukan untuk merepresentasikan nama *topic*. Hal tersebut dapat dilihat pada Gambar 5.6.

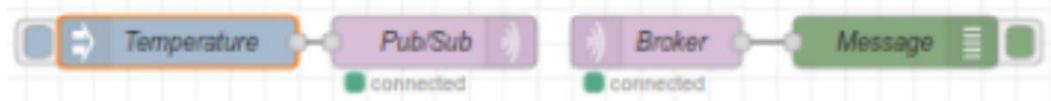


Gambar 5.6. MQTT *publish variable header*

6. LANGKAH PERCOBAAN

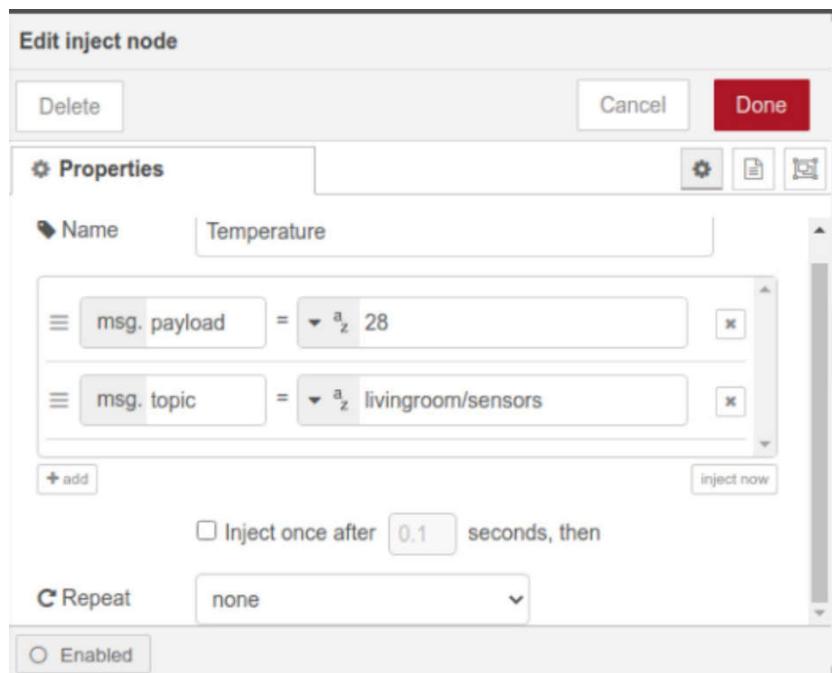
A. Koneksi MQTT Broker

1. Buka Node-Red
2. Kemudian buatlah tab baru untuk memulai membuat flow pemrograman backend-server menggunakan protokol MQTT.
3. Buatlah flow program seperti pada Gambar 6.1.



Gambar 6.1. Flow dasar transmisi data menggunakan protokol MQTT

4. Konfigurasikan Temperature Node seperti Gambar di bawah ini.



Gambar 6.2. Konfigurasi pada Temperature Node

5. Kemudian konfigurasikan Pub/Sub Node seperti pada Gambar 6.3. Klik pada tanda lingkaran merah untuk membuat koneksi dengan broker MQTT.

Edit mqtt out node

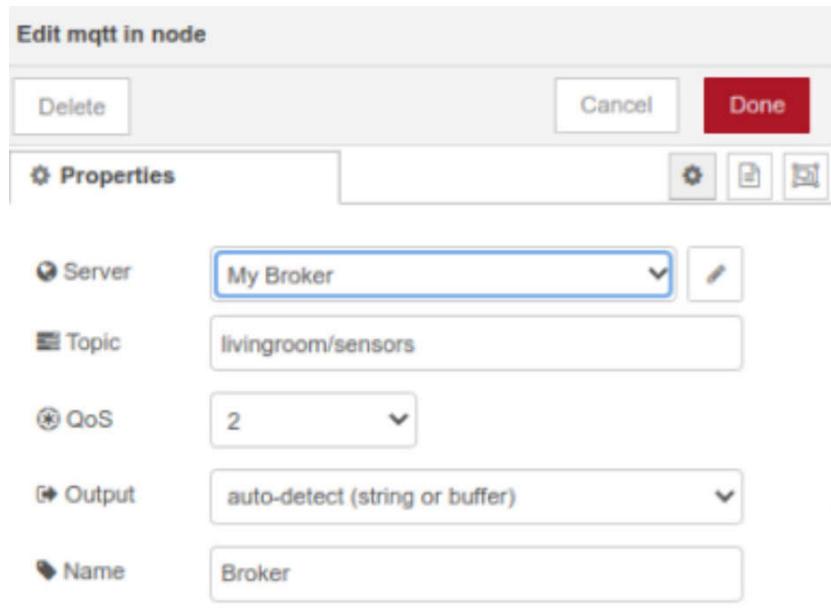
<input type="button" value="Delete"/>	<input type="button" value="Cancel"/>	<input type="button" value="Done"/>
Properties		
<input checked="" type="radio"/> Server	My Broker	<input type="button" value="edit"/>
<input type="radio"/> Topic	livingroom/sensors	
<input checked="" type="radio"/> QoS	0	<input type="radio"/> Retain
<input type="radio"/> Name	Pub/Sub	
Tip: Leave topic, qos or retain blank if you want to set them via msg properties.		

Edit mqtt out node > Edit mqtt-broker node

<input type="button" value="Delete"/>	<input type="button" value="Cancel"/>	<input type="button" value="Update"/>
Properties		
<input type="radio"/> Name	My Broker	
<input type="radio"/> Connection <input type="radio"/> Security <input type="radio"/> Messages		
<input checked="" type="radio"/> Server	localhost	Port 1883
<input type="checkbox"/> Use TLS		
<input checked="" type="radio"/> Protocol	MQTT V3.1.1	
<input type="radio"/> Client ID	Leave blank for auto generated	
<input type="radio"/> Keep Alive	60	
<input type="radio"/> Session	<input checked="" type="checkbox"/> Use clean session	
<input type="radio"/> Enabled	<input type="radio"/> 2 nodes use this config	On all flows

Gambar 6.3. Konfigurasi pada Pub/Sub Node

6. Setelah itu, konfigurasikan Broker Node seperti pada Gambar 6.4. Sesuaikan dengan broker server sebelumnya, agar tercipta koneksi. Perhatikan juga topic yang digunakan, agar pesan dapat dikirim pada address yang benar.

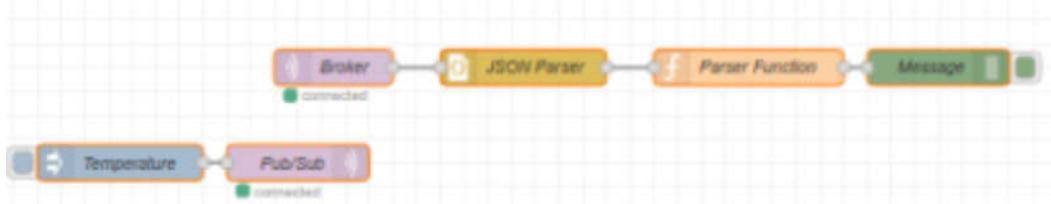


Gambar 6.4. Konfigurasi pada Broker Node

- Setelah semua telah dikonfigurasi, deploy flow dan dokumentasikan hasil outputnya.

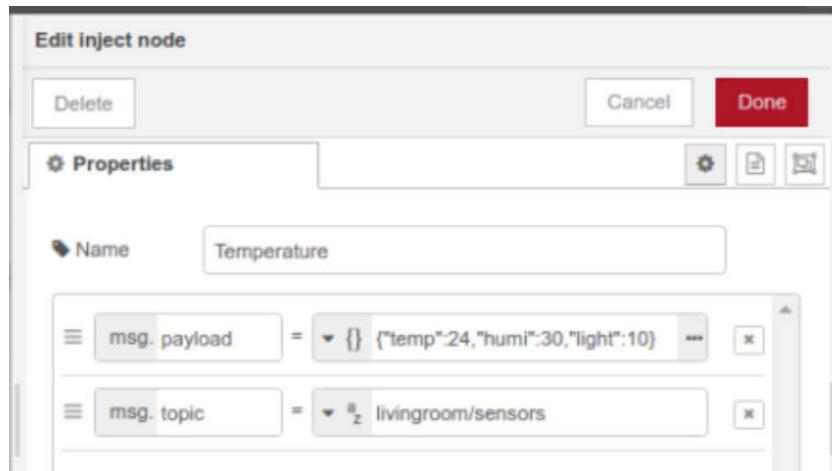
B. Menerima Data JSON Melalui Protokol MQTT

- Buatlah flow program seperti pada Gambar 6.5.



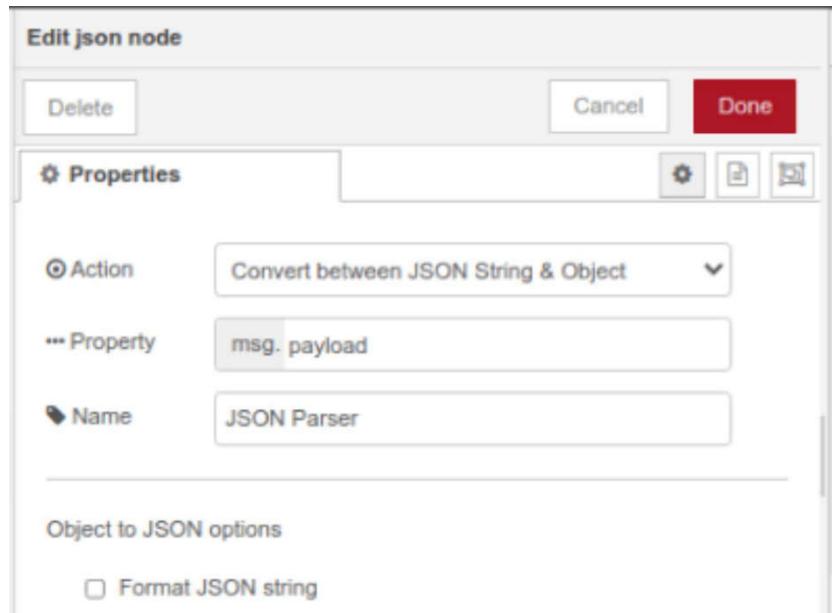
Gambar 6.5. Flow pemrosesan data JSON melalui protokol MQTT

- Flow pada Gambar 6.5 merupakan pengembangan dari Gambar 6.1 dengan menambahkan JSON Parser dan Parser Function Node.
- Setelah flow program dibuat, konfigurasikan Temperature Node seperti Gambar 6.6.



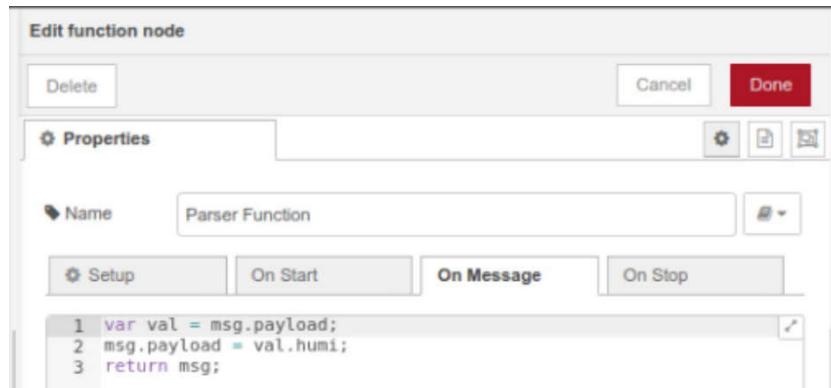
Gambar 6.6. Konfigurasi pada Temperature Node

4. Kemudian konfigurasi JSON Parser Node seperti pada Gambar 6.7.



Gambar 6.7. Konfigurasi pada JSON Parser Node

5. Setelah itu, konfigurasi pula Parser Function Node seperti Gambar 6.8, agar data JSON dapat dipisahkan dan diambil yang diperlukan saja.

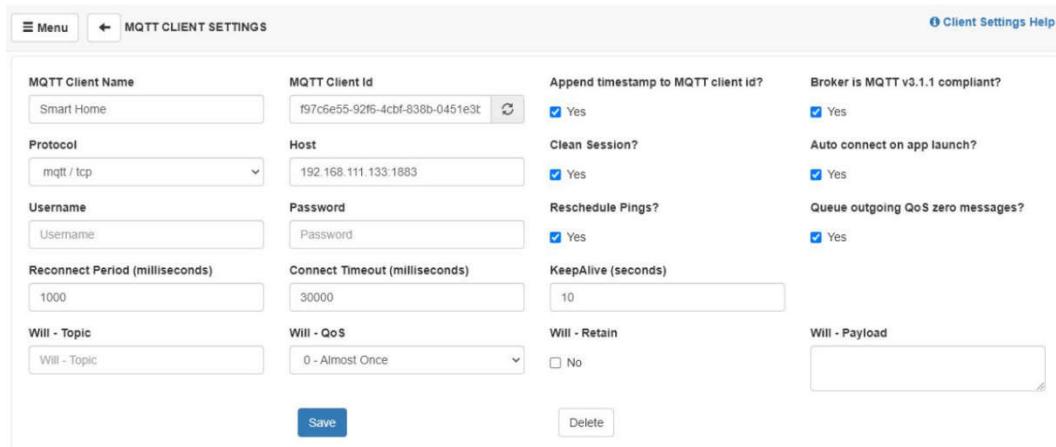


Gambar 6.8. Konfigurasi pada Parser Function Node

6. Deploy flow program, kemudian dokumentasikan hasilnya.
7. Kembangkan flow program tersebut agar mempunya 2 input (Inject Node). Input pertama seperti contoh, kemudian pisahkan datanya agar bisa tertampil dalam log yang berbeda. Sementara itu, Input yang lain menggunakan topic *kitchen/sensors*, dengan jenis sensor flame : 0 , metane : 0, temp : 24, humi : 38. Tampilkan datanya dalam log yang berbeda-beda.

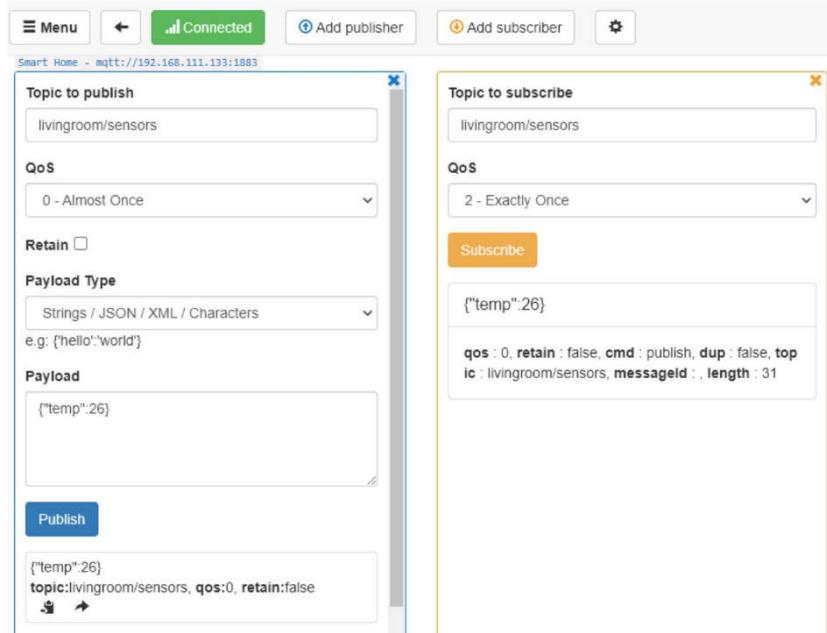
C. Mengirim Dummy Data untuk Simulasi I/O Menggunakan Hardware

1. Buka browser Google Chrome yang terinstall pada Windows.
2. Install ekstensi MQTTbox untuk Google Chrome.
3. Setelah itu, buka MQTTbox, klik Create Clients. Setelah itu, konfigurasikan MQTTbox seperti Gambar 6.9. Sesuaikan IP Address Host dengan IP Address MQTT Server pada VM Ubuntu. Untuk mengetahui alamat IP, pada Terminal, ketikkan perintah ifconfig.



Gambar 6.9. Konfigurasi MQTTbox untuk koneksi ke Broker

- Setelah itu, klik Save. Kemudian konfigurasikan topic, QoS, Payload Type, dan Payload seperti Gambar 6.10.



Gambar 6.10. Konfigurasi MQTTbox Client

- Kemudian, pada terminal Ubuntu, ketikkan perintah sudo ufw allow 1883 untuk membuka port 1883 agar tidak di-block Firewall.
- Untuk mencoba publish/mengirim data, klik tombol Publish. Sebelumnya klik tombol Subscribe untuk mengetahui data yang dikirim dapat diterima kembali oleh Clients.
- Dokumentasikan hasilnya.
- Kembangkan flow program yang sudah dibuat agar dapat menerima data berupa String, Boolean dan Number. Kemudian tampilkan data tersebut secara terpisah pada masing-masing log.

7. PERTANYAAN DAN TUGAS

- Kembangkan flow program yang telah dibuat agar ketika terdapat input nilai metane > 5, maka akan ditampilkan pada log atau debug node “Warning”. Jika input metane <=5, tampilkan data pada debug node “Save”.
- Buatlah agar flow program yang terakhir dapat menyimpan data ke dalam database mysql. Tambahkan kolom data timestamp pada database.

3. Pada MQTTBox, tambahkan 2 publisher dengan klik button Add Publisher. Setting QoS pada masing-masing publisher menggunakan QoS yang berbeda. Catat waktu data yang masuk ke mysql. Berikan analisis dan kesimpulan berdasarkan data tersebut.

- 1. NO. JOBSHEET : 7**
- 2. JUDUL** : TRANSMISI DATA MENGGUNAKAN HYPER TEXT TRANSFER PROTOCOL (HTTP)

3. TUJUAN

- 1) Mahasiswa dapat memahami alur kerja, kegunaan dan manfaat protokol HTTP.
- 2) Mahasiswa dapat memahami dan mengimplementasikan protokol HTTP pada sistem IoT untuk monitoring dan kendali.

4. ALAT DAN BAHAN

- 1) Komputer terpasang Node Red
- 2) Postman

5. TEORI SINGKAT

Hypertext Transfer Protocol (HTTP) adalah sebuah protokol jaringan lapisan aplikasi yang digunakan untuk sistem informasi terdistribusi, kolaboratif, dan menggunakan hipermédia. Penggunaannya banyak pada pengambilan sumber daya yang saling terhubung dengan tautan, yang disebut dengan dokumen hiperteks, yang kemudian membentuk World Wide Web pada tahun 1990 oleh fisikawan Inggris, Tim Berners-Lee. Hingga kini, ada dua versi mayor dari protokol HTTP, yakni HTTP/1.0 yang menggunakan koneksi terpisah untuk setiap dokumen, dan HTTP/1.1 yang dapat menggunakan koneksi yang sama untuk melakukan transaksi. Dengan demikian, HTTP/1.1 bisa lebih cepat karena memang tidak usah membuang waktu untuk pembuatan koneksi berulang-ulang.

Pengembangan standar HTTP telah dilaksanakan oleh Konsorsium World Wide Web (World Wide Web Consortium/W3C) dan juga Internet Engineering Task Force (IETF), yang berujung pada publikasi beberapa dokumen Request for Comments (RFC), dan yang paling banyak dirujuk adalah RFC 2616 (yang dipublikasikan pada bulan Juni 1999), yang mendefinisikan HTTP/1.1.

HTTP adalah sebuah protokol meminta/menjawab antara klien dan server. Sebuah klien HTTP (seperti web browser atau robot dan lain sebagainya), biasanya memulai permintaan dengan membuat hubungan ke port tertentu di sebuah server Webhosting tertentu (biasanya port 80). Klien yang mengirimkan permintaan HTTP juga dikenal dengan user agent. Server yang meresponsnya,

yang menyimpan sumber daya seperti berkas HTML dan gambar, dikenal juga sebagai origin server. Di antara user agent dan juga origin server, bisa saja ada penghubung, seperti halnya proxy, gateway, dan juga tunnel.

HTTP tidaklah terbatas untuk penggunaan dengan TCP/IP, meskipun HTTP merupakan salah satu protokol aplikasi TCP/IP paling populer melalui Internet. Memang HTTP dapat diimplementasikan di atas protokol yang lain di atas Internet atau di atas jaringan lainnya. seperti disebutkan dalam “*implemented on top of any other protocol on the Internet, or on other networks.*”, tapi HTTP membutuhkan sebuah protokol lapisan transport yang dapat diandalkan. Protokol lainnya yang menyediakan layanan dan jaminan seperti itu juga dapat digunakan.

6. LANGKAH PERCOBAAN

A. Instalasi SQL Server (MySQL)

1. Buka Terminal Ubuntu.
2. Kemudian install mysql-server menggunakan perintah berikut.

```
sudo apt update && sudo apt install mysql-server
```

3. Tunggu proses instalasi sampai selesai. Kemudian gunakan perintah berikut untuk memastikan mysql telah terpasang dan aktif berjalan.

```
systemctl is-active mysql
```

```
systemctl is-enabled mysql
```

4. Langkah selanjutnya adalah melakukan instalasi keamanan untuk database yang akan digunakan (set username dan password, dll).

```
sudo mysql_secure_installation
```

5. Kemudian setelah selesai instalasi, lakukan pengaturan username dan password untuk user root di localhost seperti pada Gambar 6.1.
6. Setelah itu ketik perintah **exit** untuk keluar dari mysql shell. Kemudian login menggunakan user root dan password yang telah dibuat sebelumnya.

```
sudo mysql -u root -p
```

7. Buat database menggunakan perintah berikut.

```
CREATE DATABASE nama_database;
```

```

labiot@labiot:~$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21
Server version: 5.7.35-0ubuntu0.18.04.2 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ALTER USER 'root'@'localhost'
-> IDENTIFIED WITH mysql_native_password BY 'labiot123';
Query OK, 0 rows affected (0,00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,01 sec)

```

Gambar 6.1. Setting password pada user root di localhost

8. Lakukan pemeriksaan pada daftar database untuk memastikan bahwa database yang telah dibuat sudah berhasil terbentuk menggunakan perintah pada baris pertama. Kemudian pilih data database yang telah kita buat seperti pada script perintah barus kedua berikut.

SHOW databases;

USE nama_database;

9. Buat tabel menggunakan perintah seperti pada Gambar 6.2. (nama dan row tabel menyesuaikan project masing-masing).

```

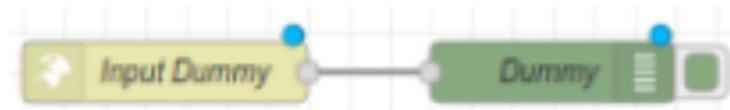
mysql> CREATE TABLE `banjir_db`.`livingroom_table` (
->   `id` INT(45) NOT NULL AUTO_INCREMENT,
->   `time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
->   `temp` VARCHAR(45) NULL,
->   `humi` VARCHAR(45) NULL,
->   `light` VARCHAR(45) NULL,
->   `dev_id` VARCHAR(45) NULL,
->   PRIMARY KEY (`id`));
Query OK, 0 rows affected (0,02 sec)

```

Gambar 6.2. Perintah untuk membuat tabel pada database

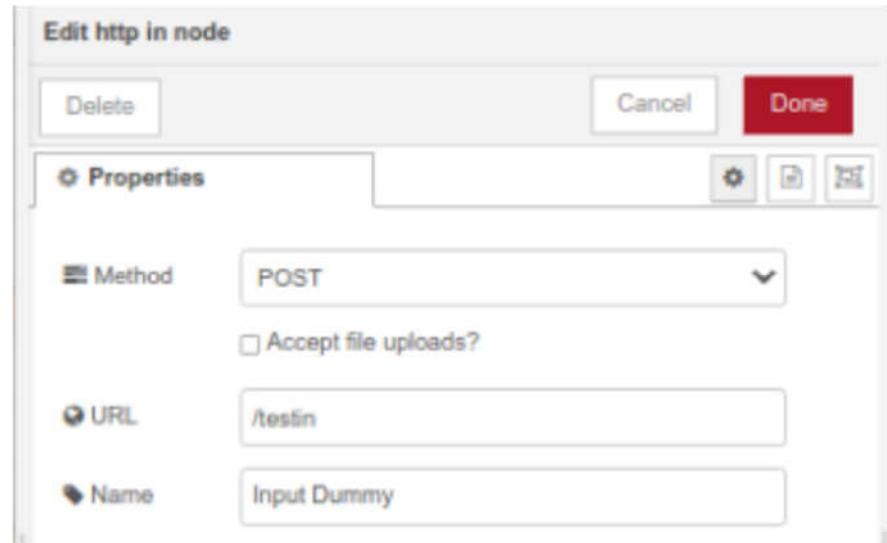
B. Basic Flow Transmisi Data Menggunakan Protokol HTTP

1. Install Postman pada Komputer Windows, kemudian lakukan pendaftaran akun.
2. Buatlah flow program seperti pada Gambar 6.3. Gunakan HTTP In node dan Debug Node.



Gambar 6.3. Basic flow transmisi data menggunakan protokol HTTP

- Konfigurasi Input Dummy node seperti Gambar 6.4.



Gambar 6.4. Konfigurasi pada Input Dummy Node

- Deploy flow program, kemudian masuk ke Postman. Konfigurasi Postman seperti pada Gambar 6.5A untuk mengirim data String dengan format encoded utf-8 dan Gambar 6.5B untuk mengirim data dalam format JSON. Klik Send untuk mengirim data. (Isian pada Body sama, Header berbeda).
- Langkah selanjutnya, dokumentasikan hasil keluaran dari flow program.

Screenshot A (UTF-8 String):

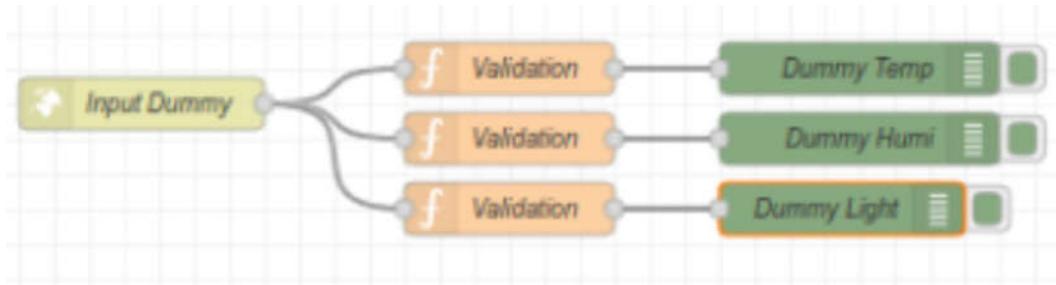
Key	Value	Description
Data	Hello World	
New key	Value	Description

Screenshot B (JSON):

Key	Value	Description
Content-Type	application/json	
New key	Value	Description

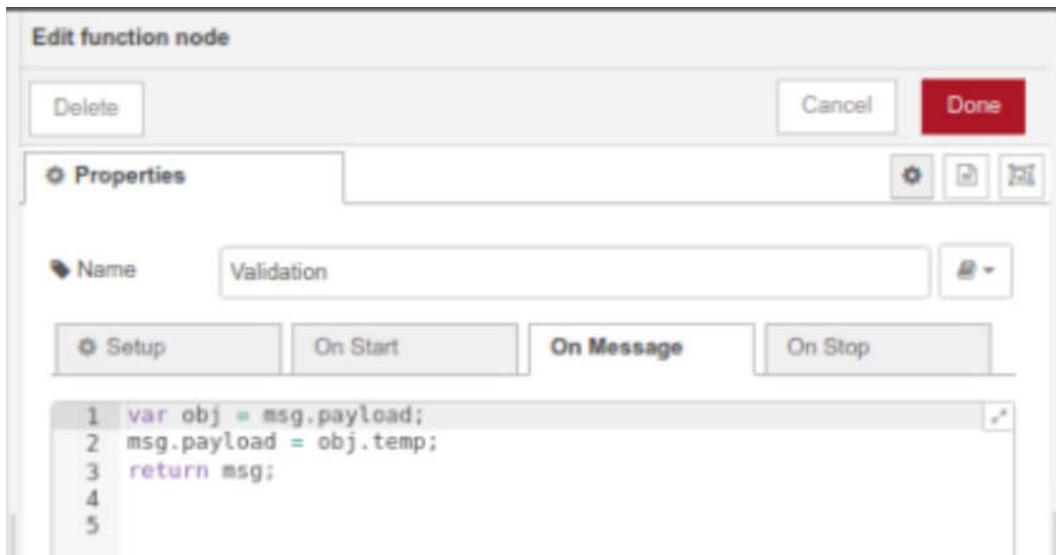
Gambar 6.6. Konfigurasi pada Postman untuk pengiriman data menggunakan format: (A) UTF-8 String , (B) JSON

- Setelah itu, buatlah flow program seperti Gambar 6.6 untuk melakukan parsing data JSON.



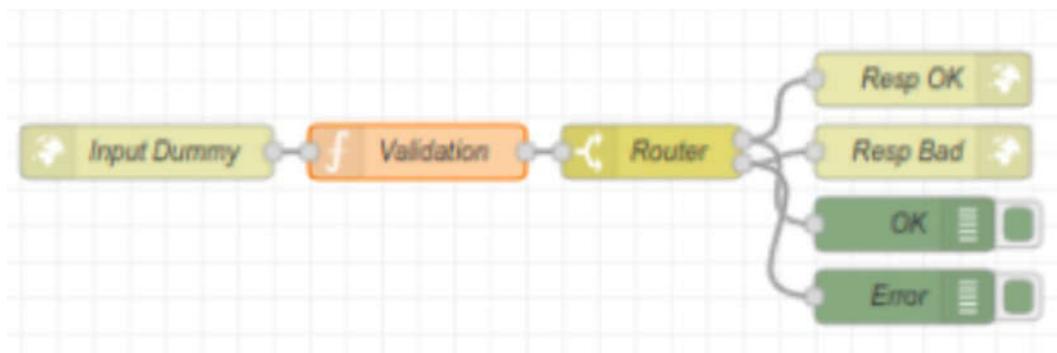
Gambar 6.6. Flow program untuk parsing data JSON

- Kemudian konfigurasi Function/Validation node seperti pada Gambar 6.7. Ubah *obj.variabel_datamu* pada setiap node, sesuai dengan project masing-masing.



Gambar 6.7. Konfigurasi pada Validation node

- Deploy program dan dokumentasikan hasil keluarannya.
- Buatlah flow program seperti pada Gambar 6.8 untuk simulasi transmisi data pada protokol HTTP dengan respons (acknowledgement/ack).



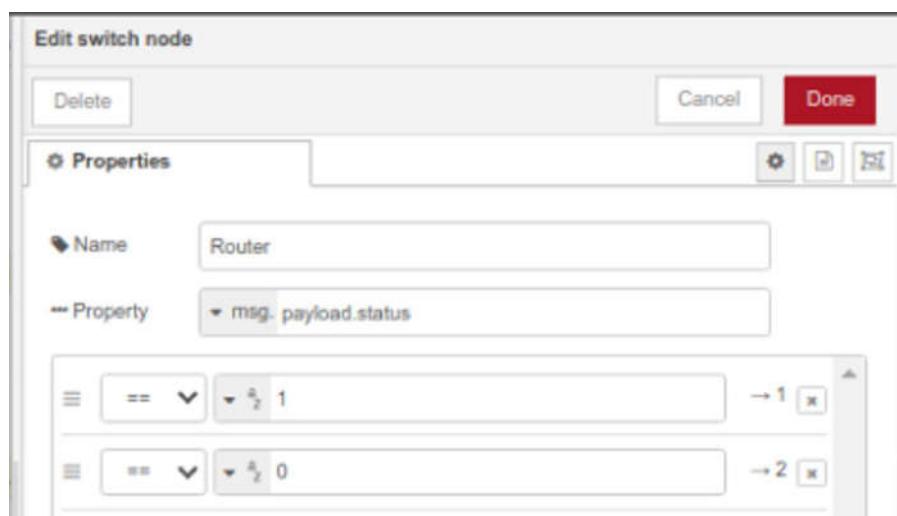
Gambar 6.8. Flow program transmisi data pada protokol HTTP dengan respons

10. Konfigurasi Validation node seperti Gambar 6.9.



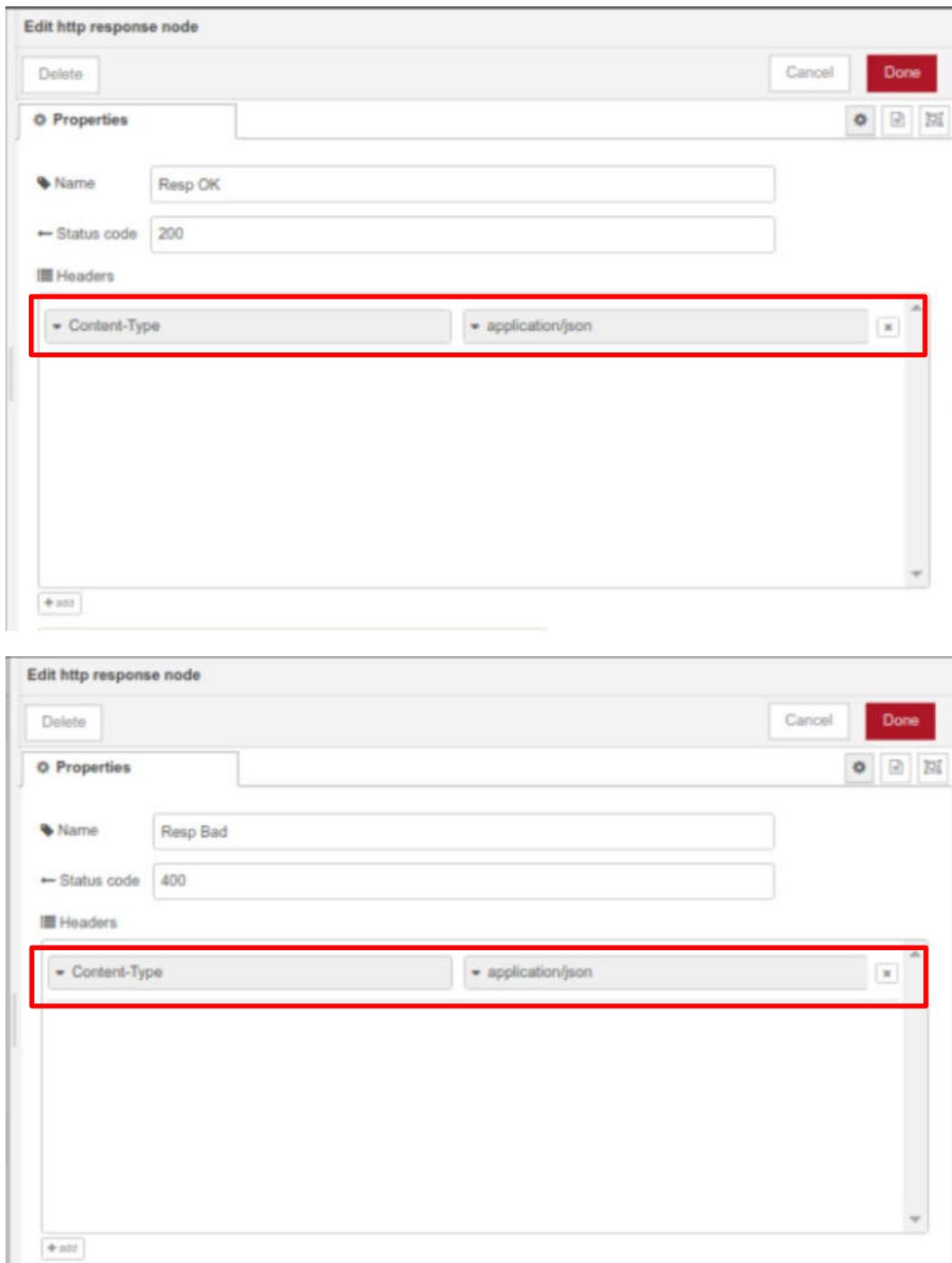
Gambar 6.9. Konfigurasi pada Validation node

11. Kemudian, konfigurasi Router Node seperti Gambar 6.10.



Gambar 6.10. Konfigurasi pada Router node

12. Setelah itu, konfigurasi HTTP Response (Resp Ok dan Resp Bad) node seperti Gambar 6.11.



Gambar 6.11. Konfigurasi pada HTTP Response node

13. Kemudian kirimkan data JSON menggunakan Postman seperti pada pada Gambar 6.12 dan Gambar 6.13. Shape bertanda merah merupakan response dari server ketika data berhasil terkirim dan gagal terkirim.

The screenshot shows the Postman interface with a successful API call. The URL is 192.168.111.133:1880/testin. The 'Body' tab is selected, showing a JSON payload with keys temp, humi, and light. The response status is 200 OK, and the JSON response body is displayed as:

```

1  {
2   "temp": "28",
3   "humi": "35",
4   "light": "2",
5   "status": 1
6 }

```

Gambar 6.12. Konfigurasi pada Postman dan response data berhasil terkirim

The screenshot shows the Postman interface with an unsuccessful API call. The URL is 192.168.111.133:1880/testin. The 'Body' tab is selected, showing a JSON payload with keys temp, humi, and voltage. The response status is 400 Bad Request, and the JSON response body is displayed as:

```

1  {
2   "temp": "28",
3   "humi": "35",
4   "voltage": "2",
5   "status": 0
6 }

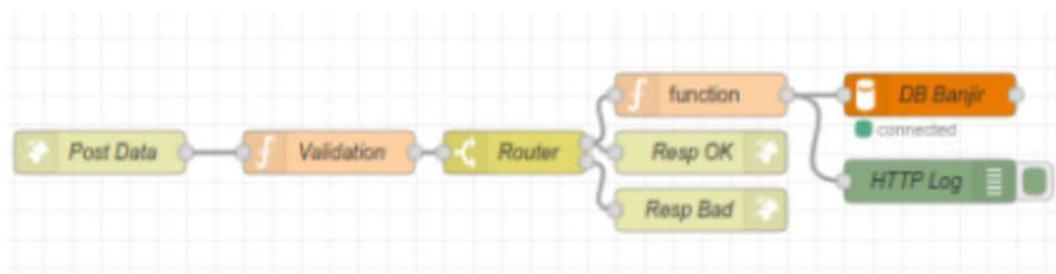
```

Gambar 6.13. Konfigurasi pada Postman dan response data gagal terkirim

14. Setelah itu,dokumentasikan hasilnya.

C. Mengirim Data JSON ke Server dan Menyimpannya ke Database

1. Buatlah flow program seperti Gambar 6.14.
2. Konfigurasi Validation, Router, Resp OK dan Resp Bad node seperti Gambar 6.9-6.11.
3. Edit konfigurasi URL pada Post Data Node menjadi **/sensor/node1**.



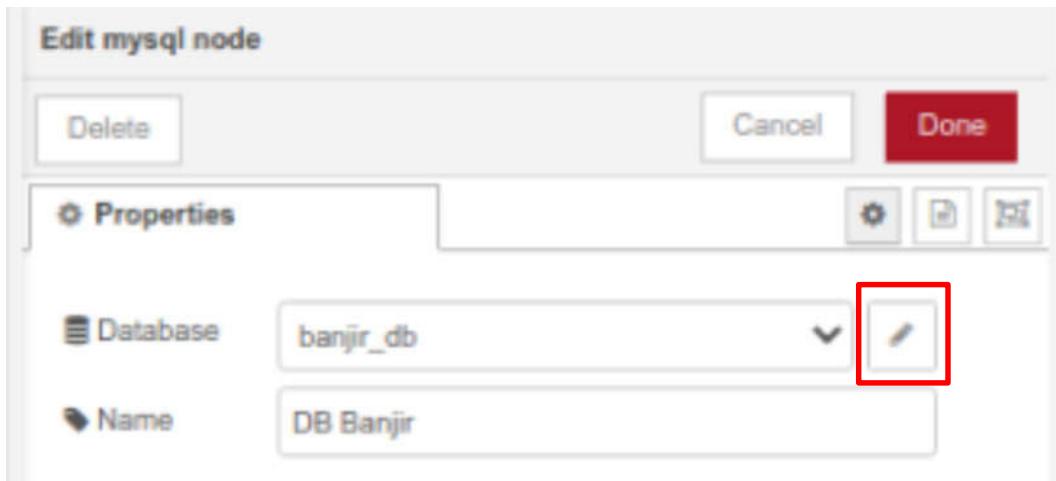
Gambar 6.14. Flow program mekanisme menyimpan data ke dalam database

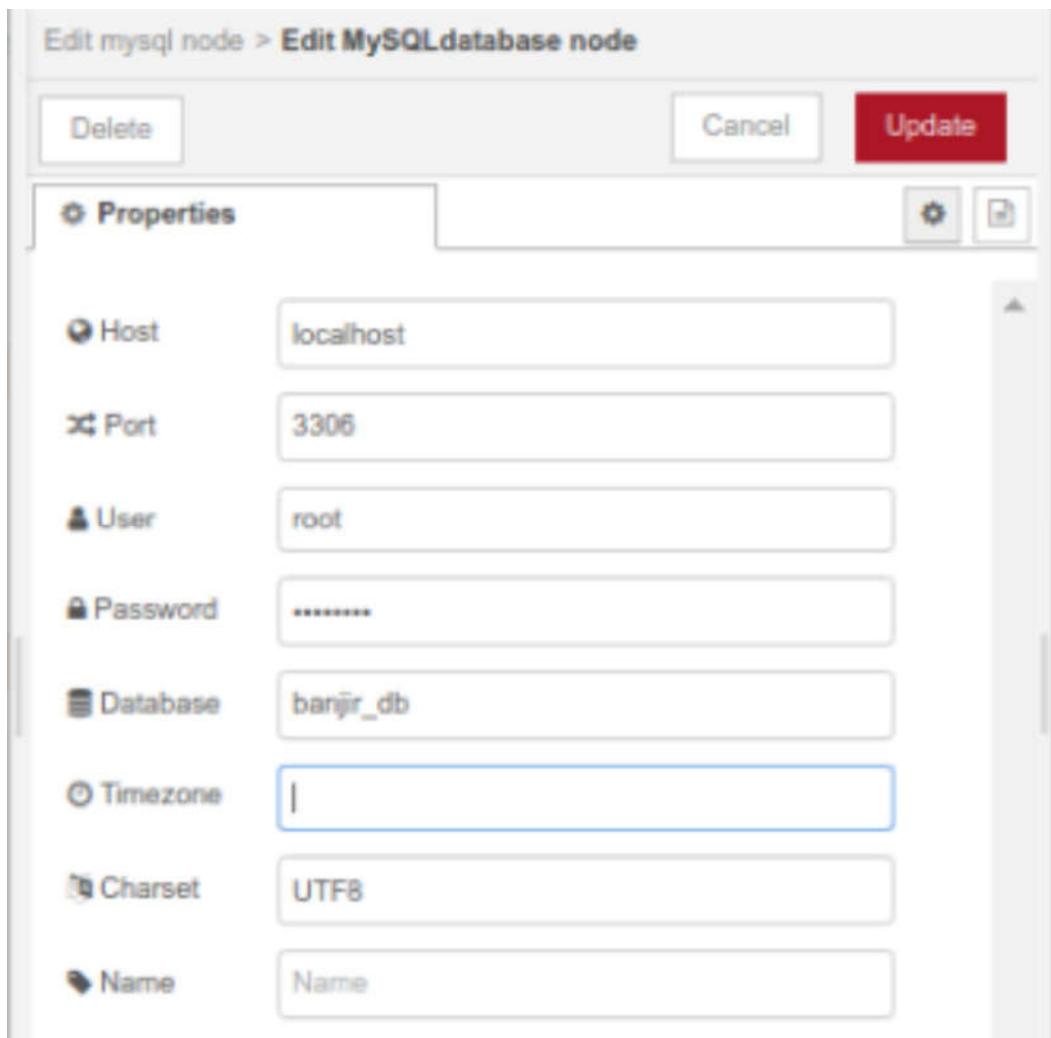
- Setelah itu, konfigurasi Function node untuk query input data ke dalam database seperti Gambar 6.15.



Gambar 6.15. Konfigurasi Function node query input data ke dalam database

- Konfigurasi Database node seperti Gambar 6.16. (Menyesuaikan dengan nama database masing-masing).





Gambar 6.16. Konfigurasi Database node

- Setelah itu, deploy program untuk menjalankan semua script.
- Buka Postman, ubah URL sesuai pengaturan pada Post Data node, kemudian kirimkan data berformat JSON seperti Gambar 6.17.
- Dokumentasikan hasil keluaran pada Debug di Node Red.

The screenshot shows a POST request configuration in Postman. The URL is set to 192.168.111.133:1880/sensor/node1. The Headers tab is selected, showing 'Content-Type: application/x-www-form-urlencoded'. The Body tab is selected, showing 'x-www-form-urlencoded' as the type. The data is listed in a table:

Key	Value	Description
dev_id	24	
rainfall	8	
level	9	

Gambar 6.17. Konfigurasi Postman untuk mengirim data JSON ke Database

7. PERTANYAAN DAN TUGAS

1. Buatlah flow program MQTT yang telah dibuat pada Jobsheet sebelumnya, kemudian gabungkan dalam satu Tab dengan flow program HTTP. Setelah itu, kondisikan agar data yang dikirim dari MQTT Clients dapat disimpan pada database dan tabel yang sama pada mekanisme protokol HTTP.
2. Dokumentasikan hasilnya dan buatlah analisis perbandingan kecepatan transmisi data diantara kedua protokol.

- 1. NO. JOBSHEET : 8**
- 2. JUDUL** : TRANSMISI DATA MENGGUNAKAN WEBSOCKET
- 3. TUJUAN**

- 1) Mahasiswa dapat memahami alur kerja, kegunaan dan kelebihan Websocket.
- 2) Mahasiswa dapat memahami dan mengimplementasikan Websocket pada sistem IoT untuk monitoring dan kendali.

4. ALAT DAN BAHAN

- 1) Komputer terpasang Node Red
- 2) Postman
- 3) Simple Websocket Clients

5. TEORI SINGKAT

WebSocket adalah cara baru untuk komunikasi antara klien dan server tanpa mengirimkan informasi tambahan yang tidak perlu melalui protokol HTTP. WebSockets menggunakan protokolnya sendiri yang dipaparkan oleh IETF. Selain memiliki protokol sendiri, WebSockets juga memiliki API yang dapat digunakan oleh aplikasi web untuk membuka dan menutup hubungan dan untuk mengirim dan menerima pesan yang disebut sebagai WebSockets API. Komunikasi dua arah penuh antara server dan klien yang lebih ringan dibandingkan dengan metode HTTP tradisional dengan bantuan WebSockets.

Websocket memungkinkan server untuk mendorong data kepada klien yang terhubung. Selain itu, penggunaan websocket dapat mengurangi lalu lintas jaringan yang tidak perlu dan latency dengan cara menggunakan full duplex melalui koneksi tunggal. Manfaat lain yang dimiliki oleh websocket adalah kompatibel dengan pre-WebSocket dunia dengan cara beralih dari koneksi HTTP ke WebSockets.

Websocket mempunyai beberapa kelebihan, antara lain sebagai berikut.

- a) Mendukung komunikasi Duplex
- b) Lebih cepat dari HTTP
- c) Meningkatkan efisiensi komunikasi antara Client dan Server
- d) Penggantian HTTP menggunakan TCP

Namun, websocket juga mempunyai kekurangan, antara lain sebagai berikut.

- a) Mengambil alih protokol komunikasi antara Client dan Server untuk koneksi tertentu.
- b) Web Browser harus sepenuhnya support HTML5.

Latar belakang yang mendasari terciptanya websocket adalah permintaan beberapa client yang mengharuskan developer bisa membuat aplikasi berbasis web secara real time. Aplikasi real time adalah ketika ada perubahan data maka saat itu juga website di browser klien juga ada perubahan, minimal muncul notifikasi. Ada alternatif lain untuk permasalahan developer tersebut, di antaranya adalah metode polling dan long polling.

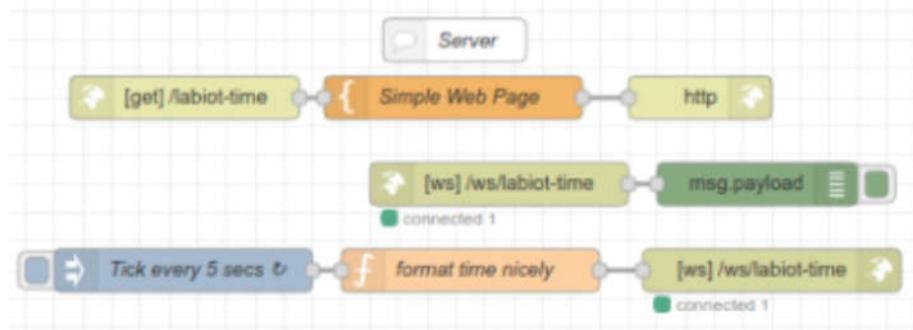
Metode polling ini mengirimkan request data ke server secara terus menerus. Jika hanya satu client yang melakukan request berulang seperti itu, mungkin tidak menyebabkan masalah. Namun jika ada beberapa client yang mengakses satu server dan berulangkali melakukan metode Polling, maka server akan jadi sibuk dan rentan terkena serangan DDOS. Hal tersebut dapat ditanggulangi menggunakan metode long polling.

Metode long polling adalah metode polling dengan interval waktu yang berkala. Jadi request tidak sesering metode Polling. Manfaatnya adalah server jauh lebih stabil dibandingkan dengan metode polling. Namun permasalahannya adalah Long Polling tidak mendukung aplikasi real time, karena terdapat interval waktu yang digunakan.

6. LANGKAH PERCOBAAN

A. Dasar Protokol Websocket

1. Buka Node-Red.
2. Buatlah flow program seperti Gambar 6.1.
3. Konfigurasi HTTP In Node ([get]/labiot-time) Pilih Method = Get dan buat URL dengan nama *labiot-time*.



Gambar 6.1. Setting password pada user root di localhost

4. Pada Simple Web Page Node, isikan kolom template html menggunakan script berikut ini.

```

<!DOCTYPE HTML>
<html>
  <head>
    <title>Simple Live Display</title>
    <script type="text/javascript">
      var ws;
      var wsUri = "ws:";
      var loc = window.location;
      console.log(loc);
      if (loc.protocol === "https:") { wsUri = "wss:"; }
      // This needs to point to the web socket in the Node-RED flow
      // ... in this case it's ws/simple
      wsUri += "//" + loc.host + loc.pathname.replace("labiot-time", "ws/labiot-time");

      function wsConnect() {
        console.log("connect", wsUri);
        ws = new WebSocket(wsUri);
        //var line = ""; // either uncomment this for a building list of messages
        ws.onmessage = function(msg) {
          var line = ""; // or uncomment this to overwrite the existing message
          // parse the incoming message as a JSON object
          var data = msg.data;
          //console.log(data);
          // build the output from the topic and payload parts of the object
          line += "<p>" + data + "</p>";
          // replace the messages div with the new "line"
          document.getElementById('messages').innerHTML = line;
          //ws.send(JSON.stringify({data: data}));
        }
        ws.onopen = function() {
          // update the status div with the connection status
          document.getElementById('status').innerHTML = "connected";
        }
      }
    </script>
  </head>
  <body>
    <div id="status"></div>
    <div id="messages"></div>
  </body>
</html>

```

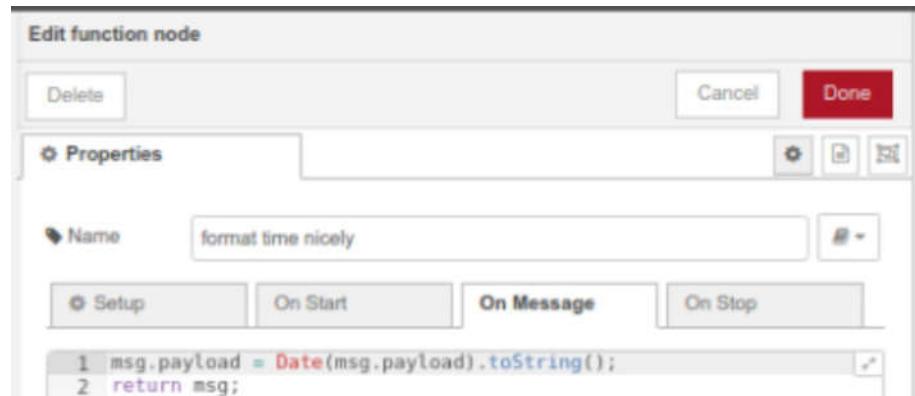
```

        //ws.send("Open for data");
        console.log("connected");
    }
    ws.onclose = function() {
        // update the status div with the connection status
        document.getElementById('status').innerHTML = "not connected";
        // in case of lost connection tries to reconnect every 3 secs
        setTimeout(wsConnect,3000);
    }
}

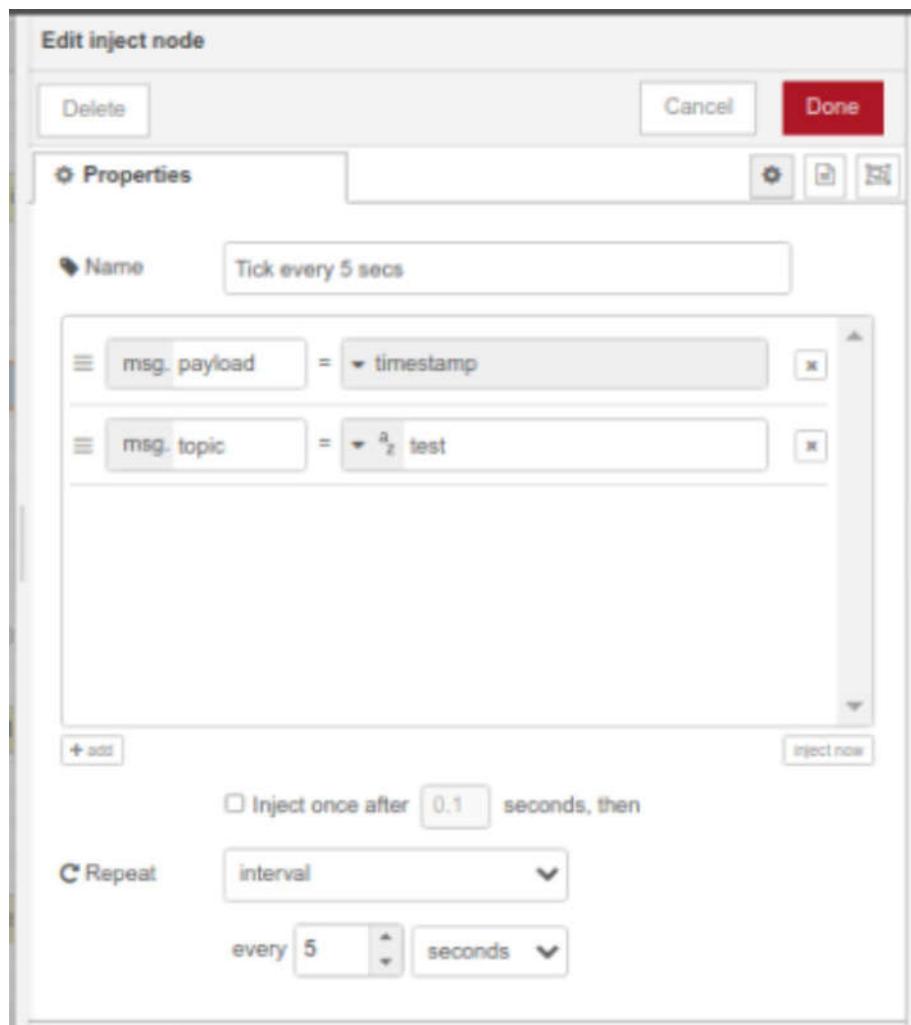
function doit(m) {
    if (ws) { ws.send(m); }
}
</script>
</head>
<body onload="wsConnect()" onunload="ws.disconnect();">
<font face="Arial">
<h1>Simple Live Display</h1>
<div id="messages"></div>
<button type="button" onclick='doit("Pesan dari Lab IoT");'>Click to send
message</button>
<hr/>
<div id="status">unknown</div>
</font>
</body>
</html>

```

5. Kemudian, konfigurasikan Inject Node seperti Gambar 6.3 agar dapat memberikan update waktu setiap 5 detik pada server.
6. Konfigurasi Function Node agar dapat mengubah data waktu menjadi format String seperti Gambar 6.2.

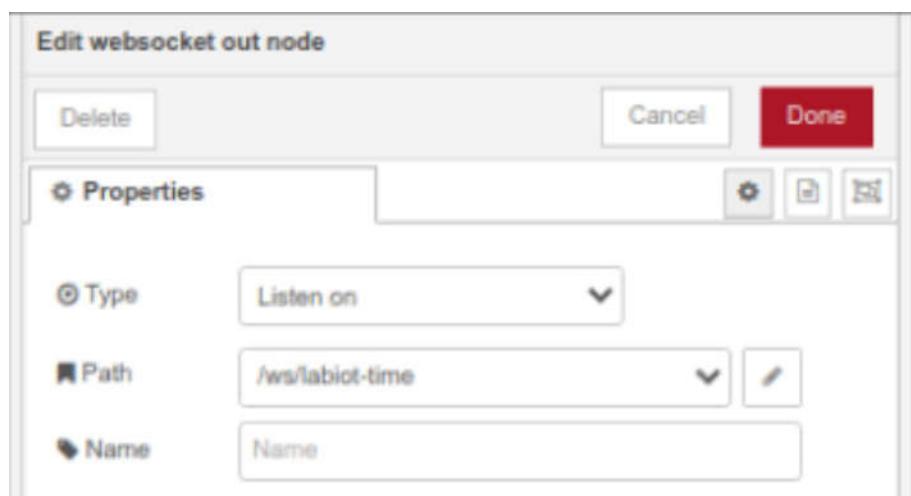


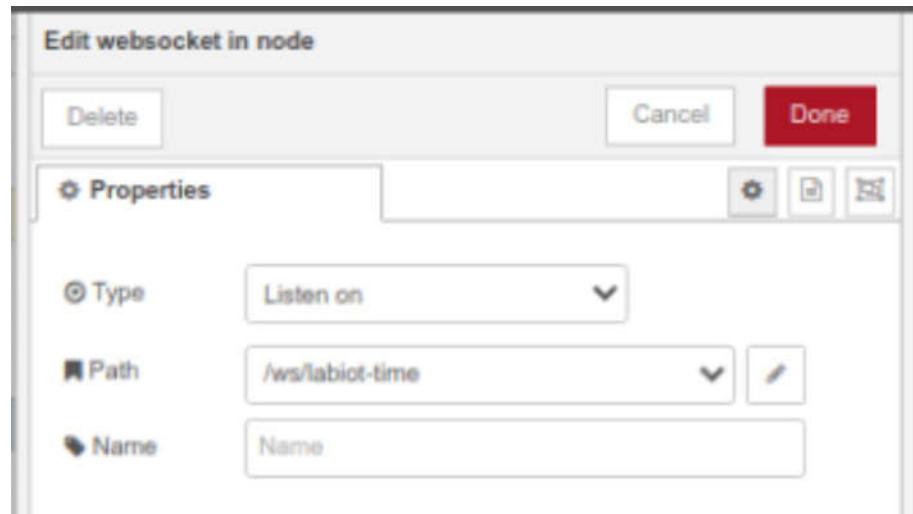
Gambar 6.2. Konfigurasi Function Node



Gambar 6.3. Konfigurasi Inject Node

7. Konfigurasi Websocket Out dan In Node ([ws]/ws/labiot-time) seperti Gambar 6.4.





Gambar 6.4. Konfigurasi Websocket Out dan In Node

8. Kemudian deploy flow program dan buka tab baru pada browser dengan isian URL adalah localhost :1880/labiot-time, seperti Gambar 6.5.

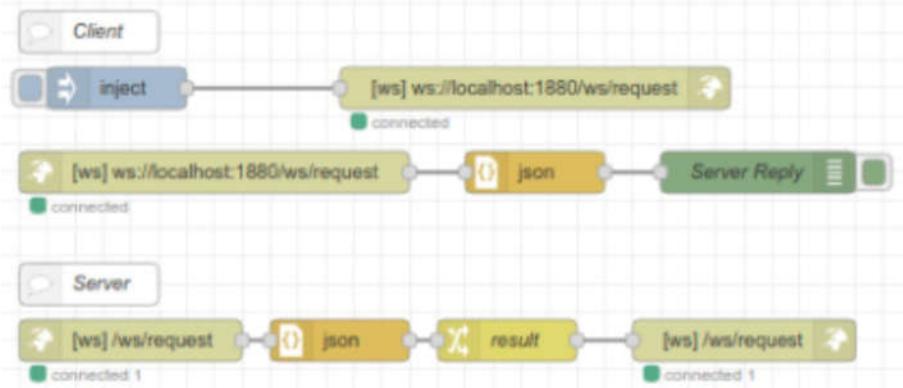


Gambar 6.5. Testing pada Web Page

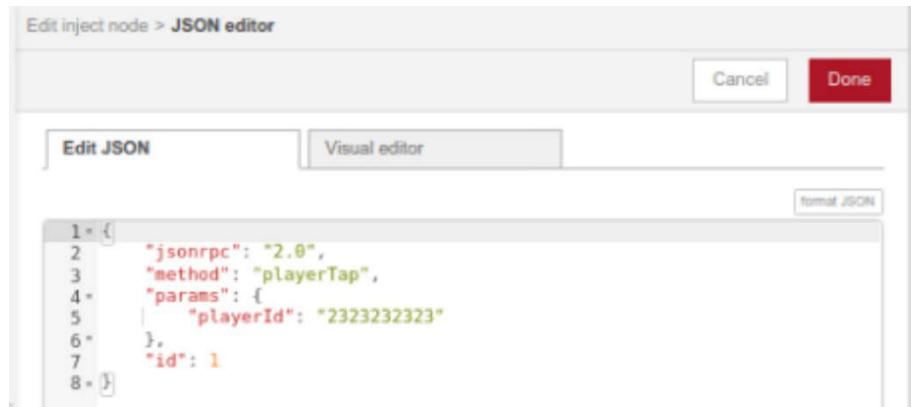
9. Amati yang terjadi dan analisis cara kerjanya menggunakan flow chart. Klik tombol Send Message dan lihat hasilnya di Debug. Setelah itu, dokumentasikan hasilnya.

B. Remote Procedure Call (RPC) via Websocket

1. Buatlah flow program seperti pada Gambar 6.6.
2. Konfigurasi Inject Node dengan msg.payload format JSON seperti Gambar 6.7.



Gambar 6.6. Flow program PRC via Websocket



Gambar 6.7. Payload JSON Inject Node

- Pada Websocket Out Node yang terhubung dengan Inject Node, konfigurasikan seperti berikut.

Type = Connect To

URL = ws://localhost:1880/ws/request

- Pada Websocket In Node disisi Client konfigurasi node tersebut seperti berikut.

Type = Connect To

URL = ws://localhost:1880/ws/request

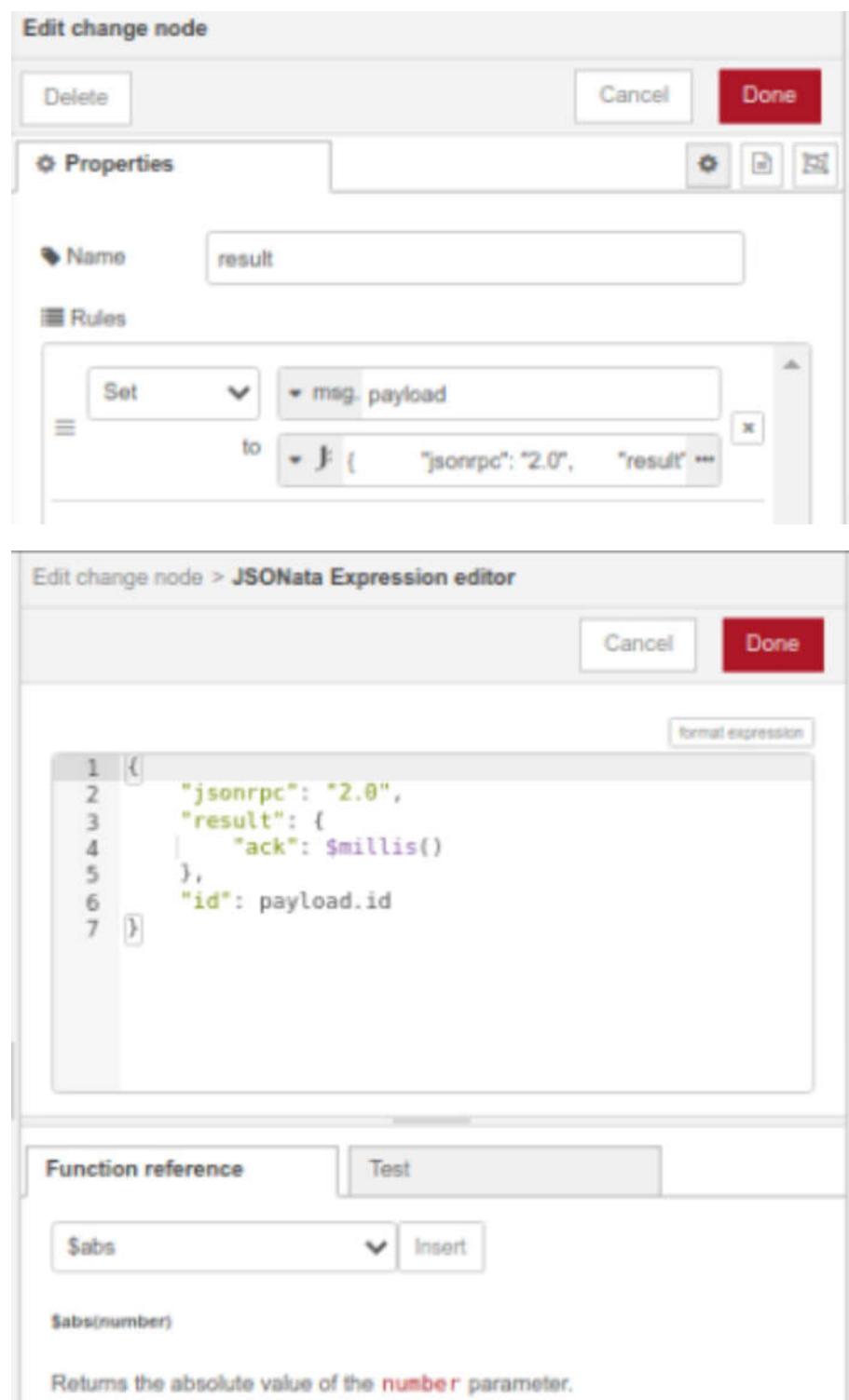
- Konfigurasi JSON Node dengan kolom isin Action adalah *Convert between JSON String & Object*.

- Pada Websocket In dan Out Node disisi Server, buat konfigurasi seperti berikut.

Type = Listen On

URL = /ws/request

7. Setelah itu, konfigurasikan Result Node seperti Gambar 6.8. Kemudian deploy program.



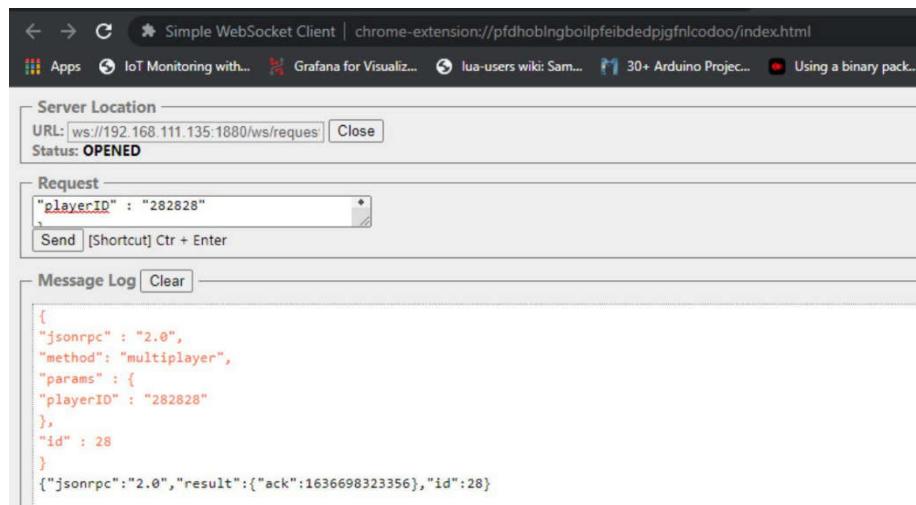
Gambar 6.8. Konfigurasi pada Result Node

- Buka Google Chrome pada computer Windows, kemudian install extension Simple Websocket Client.
- Setelah terinstall, buka Simple Websocket Client dan konfigurasi seperti berikut ini.

URL : ws://IP Address Ubuntu:1880/ws/request

Request : isikan seperti Gambar 6.7

Setelah URL diisi, klik Open dan tunggu sampai Connected. Kemudian isikan data JSON pada kolom Request dan klik send untuk mengirim data.

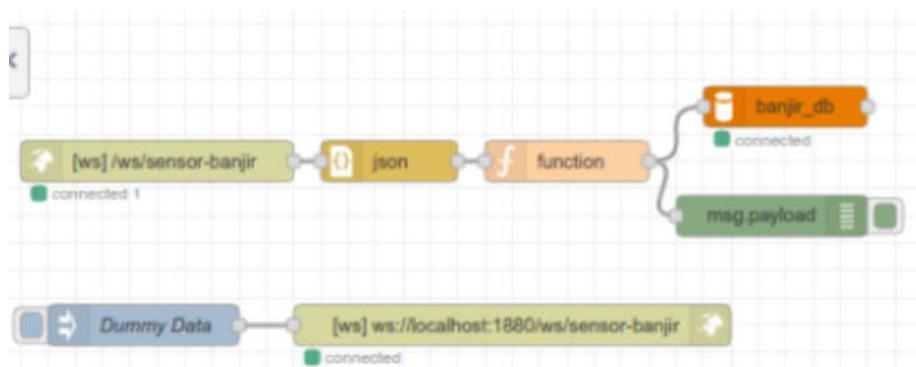


Gambar 6.9. Konfigurasi dan hasil testing pengiriman data melalui Websocket

- Dokumentasikan hasil keluarannya dan analisis cara kerja sistemnya menggunakan flow chart.

C. Mengirim Data JSON ke Server dan Menyimpannya ke Database

- Buatlah flow program seperti Gambar 6.10.

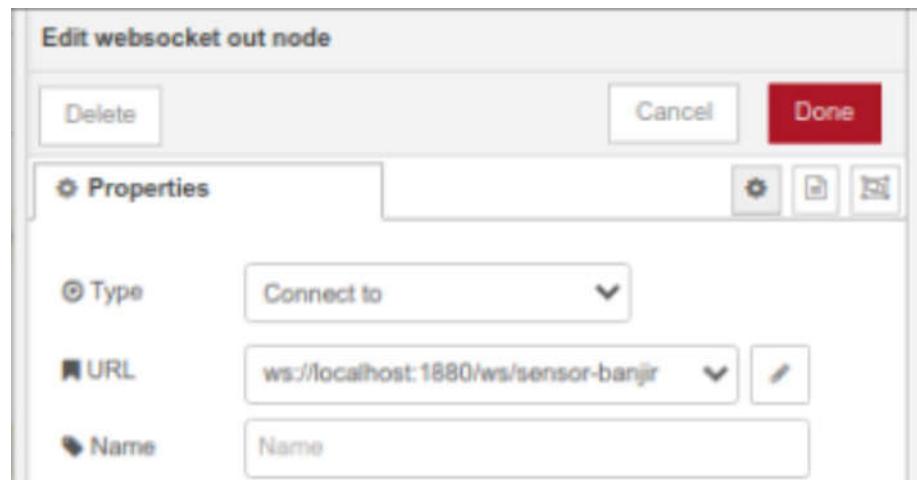


Gambar 6.10. Flow program Websocket dengan database

2. Konfigurasi Dummy Data Node menggunakan msg.payload format JSON.

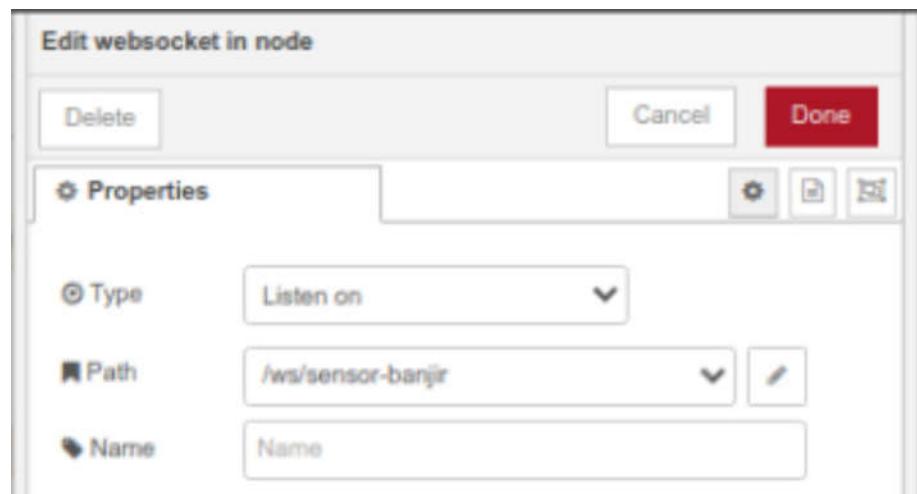
```
{  
  "dev_id": 28,  
  "rainfall": 10,  
  "level": 5  
}
```

3. Konfigurasikan Websocket Out Node seperti Gambar 6.11.



Gambar 6.11. Konfigurasi pada Websocket Out Node

4. Kemudian konfigurasi Websocket In Node seperti Gambar 6.12.



Gambar 6.12. Konfigurasi pada Websocket In Node

5. Setelah itu, konfigurasi Function node untuk query input data ke dalam database seperti Gambar 6.13.

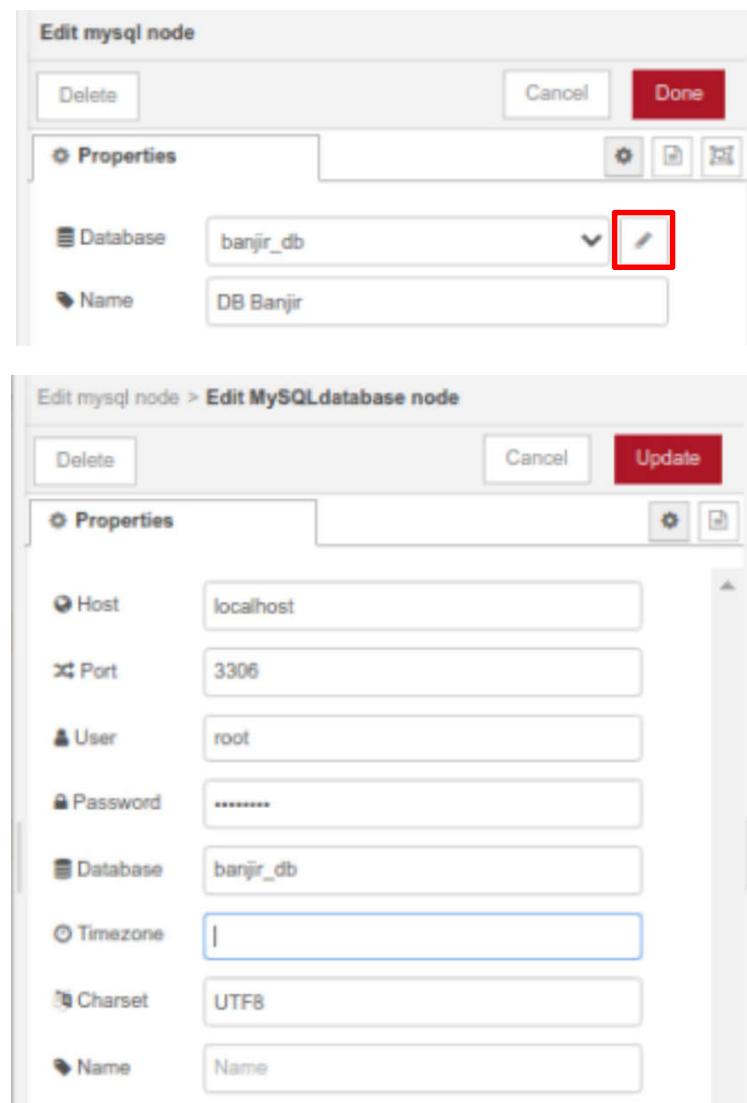
Edit function node > JavaScript editor

```
1 msg.topic = "INSERT INTO banjir_table (dev_id, rainfall, level)" +
2 " VALUES(" +
3 msg.payload.dev_id + "," +
4 msg.payload.rainfall + "," +
5 msg.payload.level + ")"
6 return msg;
```

Cancel Done

Gambar 6.13. Konfigurasi Function node query input data ke dalam database

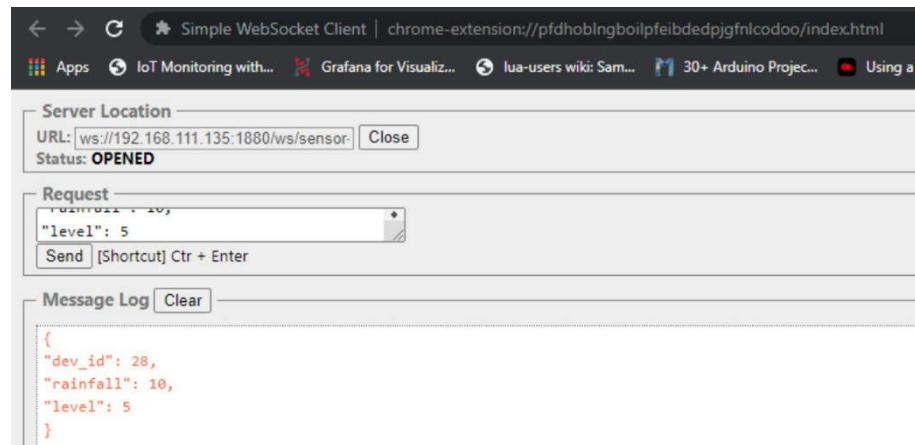
6. Konfigurasi Database node seperti Gambar 6.14. (Menyesuaikan dengan nama database masing-masing).



Gambar 6.14. Konfigurasi Database node

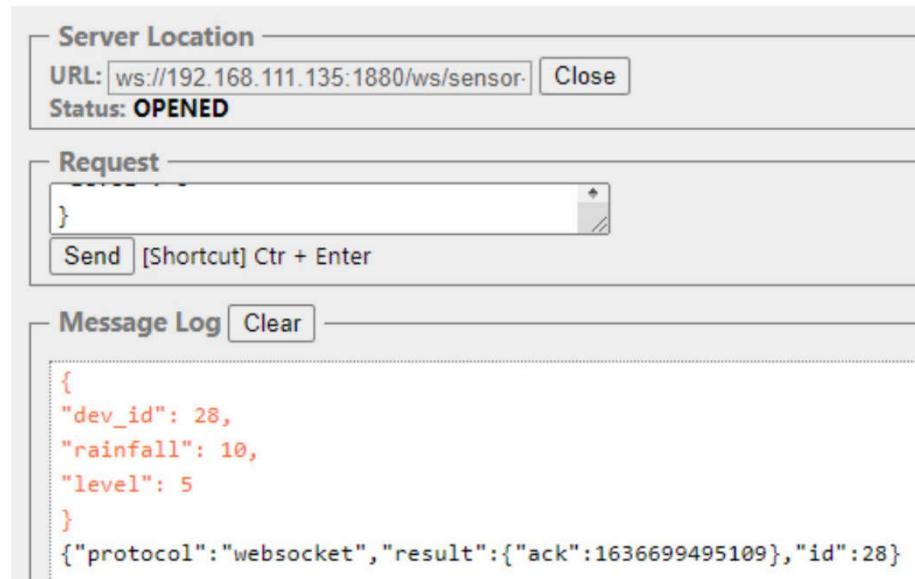
7. Setelah itu, deploy program untuk menjalankan semua script.

- Klik tombol pada Dummy Data Node dan dokumentasikan hasilnya. Analisis cara kerja flow program menggunakan flow chart.
- Buka Simple Websocket Client, konfigurasikan seperti Gambar 6.15. Kemudian lihat hasilnya di Debug dan dokumentasikan hasilnya.



Gambar 6.15. Konfigurasi pada Simple Websocket Client

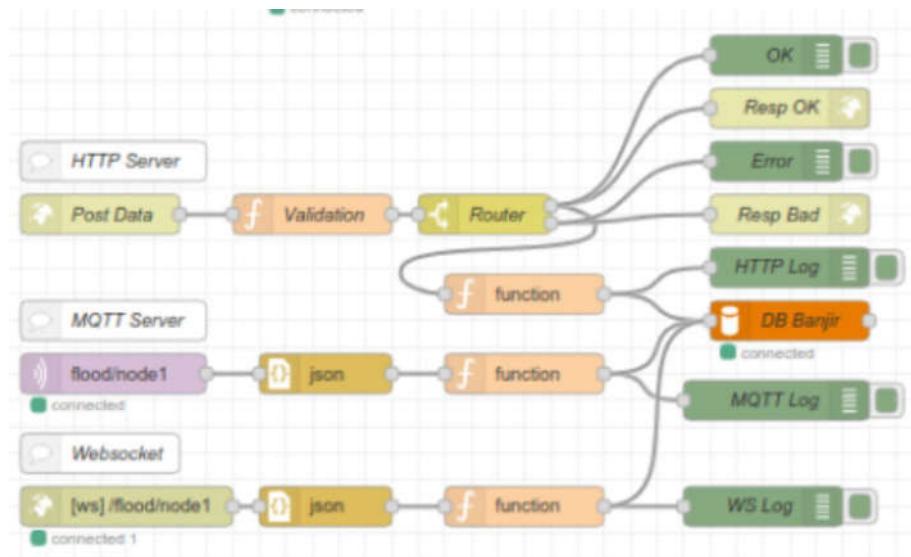
- Kembangkan program agar ketika data dikirimkan melalui Simple Webpage Client, terdapat balasan dari Server seperti Gambar 6.16.



Gambar 6.16. Konfigurasi dan balasan dari server pada Simple Websocket Client

7. PERTANYAAN DAN TUGAS

- Buatlah flow program Multi-Protocol IoT Server seperti pada Gambar 6.17. Buatlah konfigurasinya, kemudian lakukan testing pada masing-masing protokol dan dokumentasikan hasilnya.



Gambar 6.17. Flow program Multi-Protocol IoT Server

1. NO. JOBSHEET : 9

2. JUDUL : VISUALISASI DATA MENGGUNAKAN NODE-RED

DASHBOARD

3. TUJUAN

- 1) Mahasiswa dapat memahami alur kerja pengiriman data secara end-to end pada sistem IoT.
- 2) Mahasiswa dapat merancang, melakukan konfigurasi dan menampilkan data sensor pada dashboard Node-Red untuk sistem IoT.

4. ALAT DAN BAHAN

- 1) Komputer terpasang Node Red
- 2) Sensor Dummy Data Software (MQTT.box, Postman, Simple Websocket Clients)

5. TEORI SINGKAT

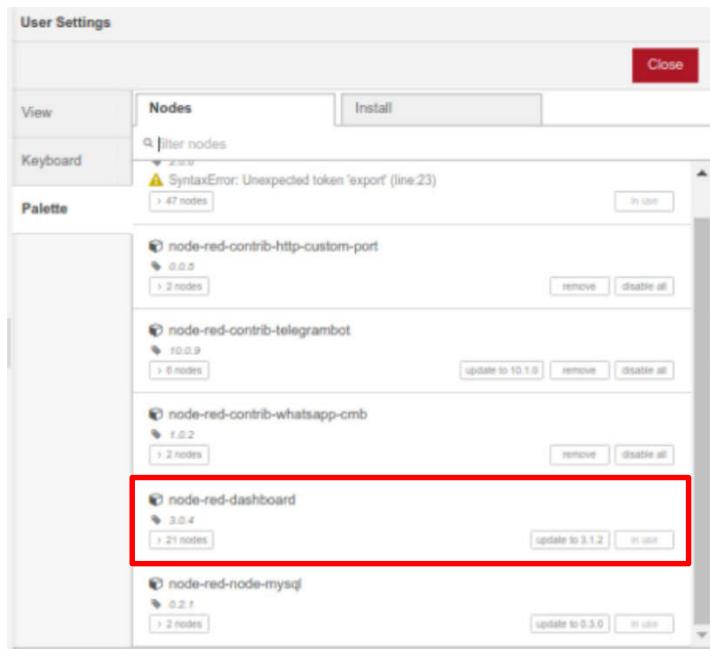
Penggunaan internet saat ini telah berkembang dan masuk dalam semua aspek kehidupan manusia. Hampir setiap peralatan yang digunakan oleh manusia terhubung ke internet, seperti mobil, jam tangan, peralatan rumah tangga, bahkan binatang peliharaan pun dapat dipakaikan kalung yang dapat mengirimkan data keberadaannya. Hal tersebut kemudian dikenal sebagai Internet of Things (IoT). IoT memerlukan elemen dashboard/plaform yang dapat mengintrepretasikan dan memvisualisasikan data-data yang dikirim ke server, agar mudah di baca dan dipahami oleh pengguna.

Platform IoT adalah suatu ekosistem yang digabungkan untuk menjadi wadah pembuatan produk dan solusi IoT agar efisien dan tidak memakan banyak waktu. Platform IoT disini sebagai lingkungan IoT yang siap dipakai untuk suatu produk atau bisnis. Platform IoT dapat digunakan untuk mengumpulkan data dari berbagai sumber, menyimpan data, menampilkan data, mengontrol perangkat, mengelola inventory perangkat dan lain-lain. Sebelum membangun sebuah platform, terdapat beberapa hal yang perlu dipertimbangkan, antara lain protokol komunikasi yang digunakan, jenis koneksi, jenis jaringan dan format data.

6. LANGKAH PERCOBAAN

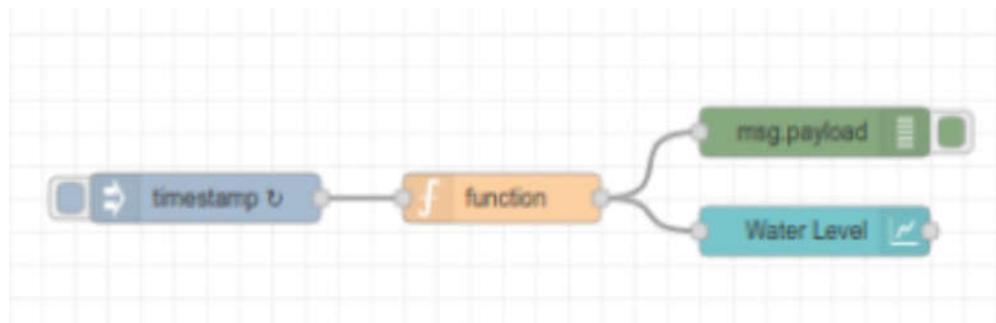
A. Dashboard Dasar

1. Buka Node-Red pada VM Ubuntu.
2. Install Pallete Dashboard dengan cara klik icon menu pada pojok kanan atas navigation bar. Kemudian pilih **Manage Palette**. Pada Tab Menu bilah kiri, klik **Palette**, klik tab **Install**, kemudian install **node-red-dashboard**. Hal tersebut dapat dilihat pada Gambar 6.1.



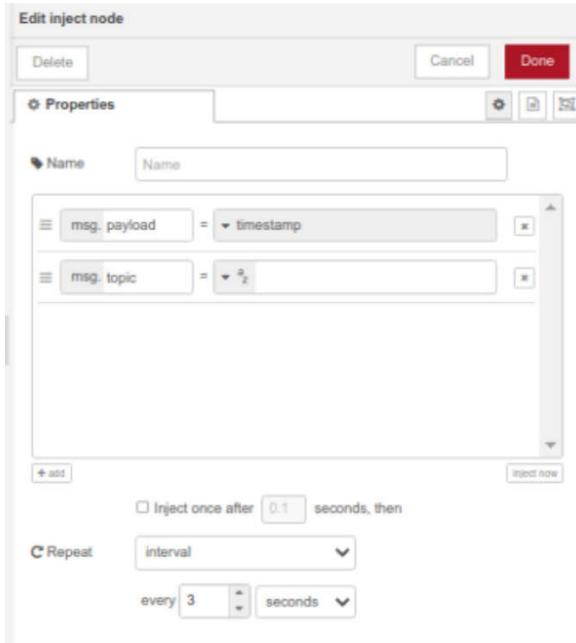
Gambar 6.1. Install Node-Red Dashboard

3. Tunggu hingga instalasi selesai.
4. Kemudian, buatlah flow program seperti pada Gambar 6.2.



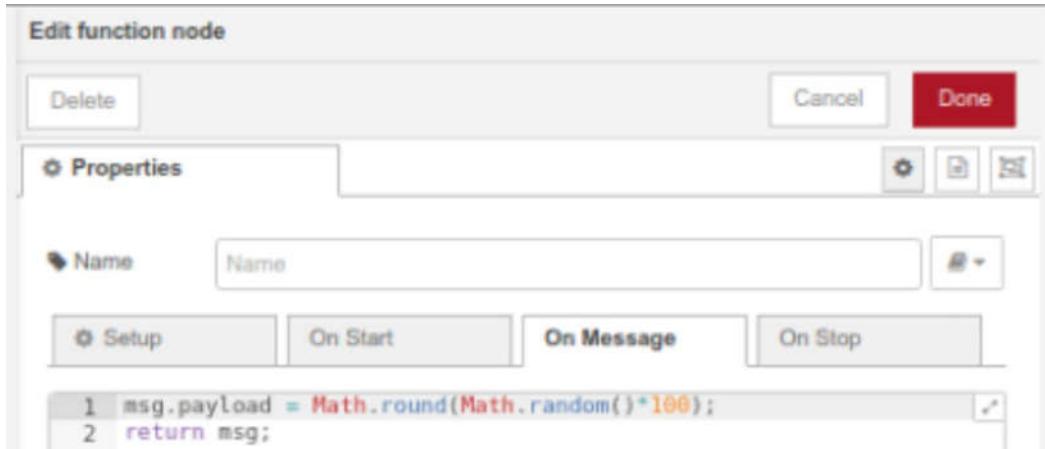
Gambar 6.2. Flow program dasar

- Konfigurasikan Inject Node / Timestamp Node agar melakukan pengiriman data setiap 3 detik sekali. Hal tersebut dapat dilihat pada Gambar 6.3.



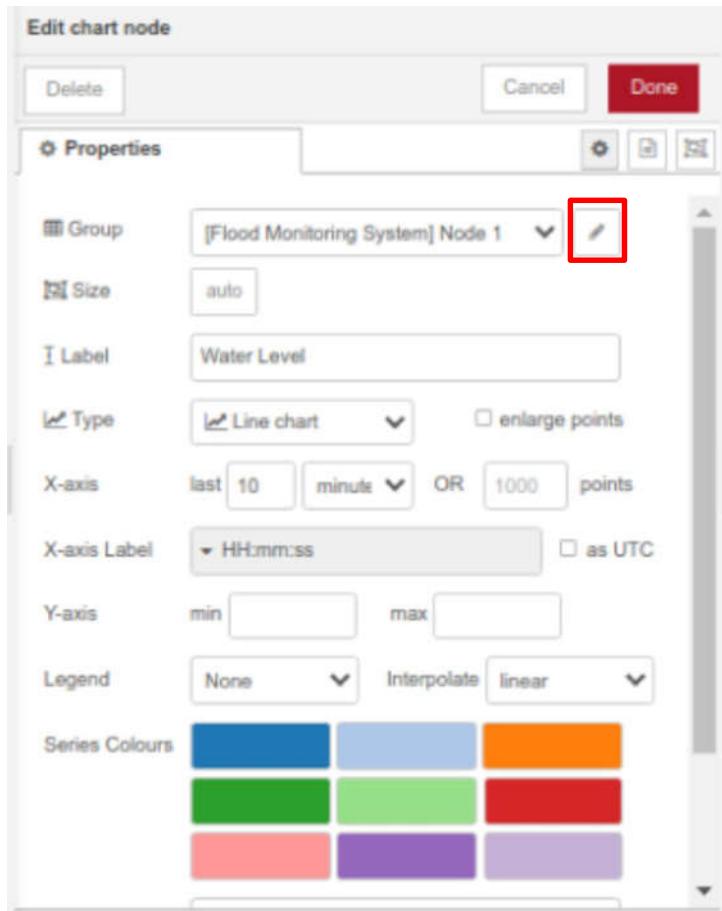
Gambar 6.3. Konfigurasi inject node agar mengirim data 3 detik sekali

- Kemudian, konfigurasi Function Node seperti Gambar 6.4.



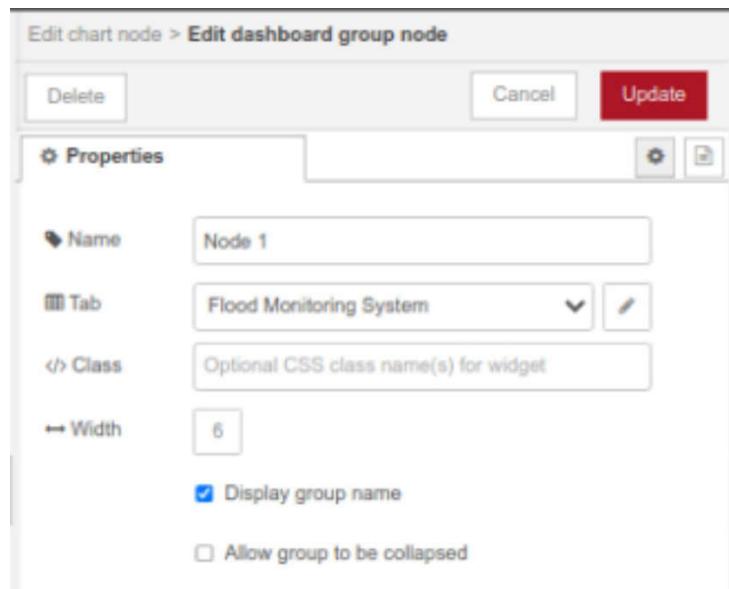
Gambar 6.4. Konfigurasi pada function node

- Setelah itu, konfigurasi Chart Node seperti pada Gambar 6.5. Buatlah Group dengan nama Flood Monitoring System. Klik pada tombol yang diberi tanda kotak merah. Isi Label dengan nama Water Level untuk memberikan judul pada grafik.



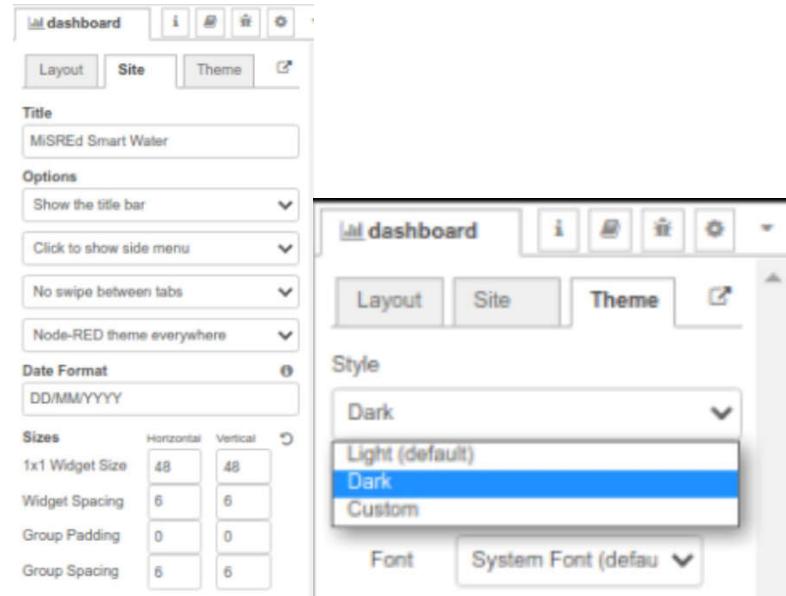
Gambar 6.5. Konfigurasi utama pada Chart Node

8. Setelah klik tombol edit yang bertanda kotak merah, kemudian konfigurasikan Name dan Tab, seperti Gambar 6.6



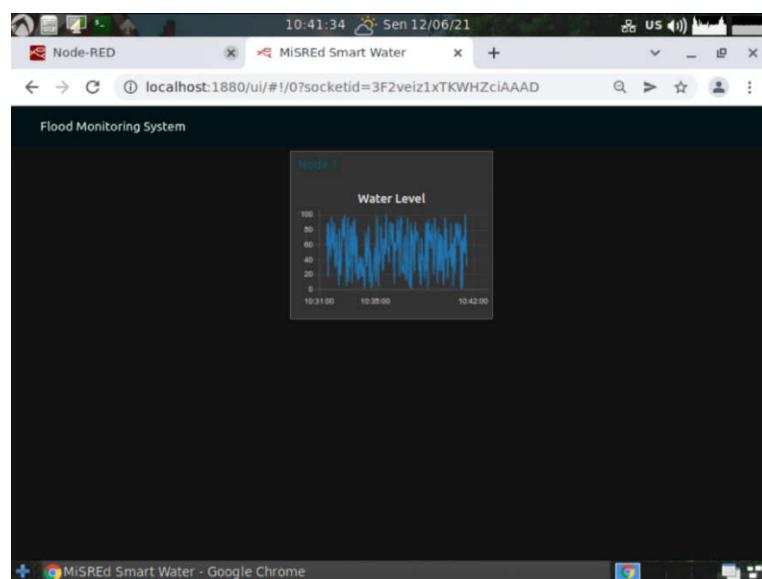
Gambar 6.6. Konfigurasi pada edit dashboard group node

- Kemudian, lakukan konfigurasi pada tampilan dashboard dengan cara klik menu drop down (satu group dengan debug), pilih dashboard. Pada Layout Tab, konfigurasi Title dengan nama MiSREd Smart Water. Pada Theme Tab, pilih tema sesuai keinginan (dark). Hal tersebut dapat dilihat pada Gambar 6.7.



Gambar 6.7. Konfigurasi tampilan dashboard

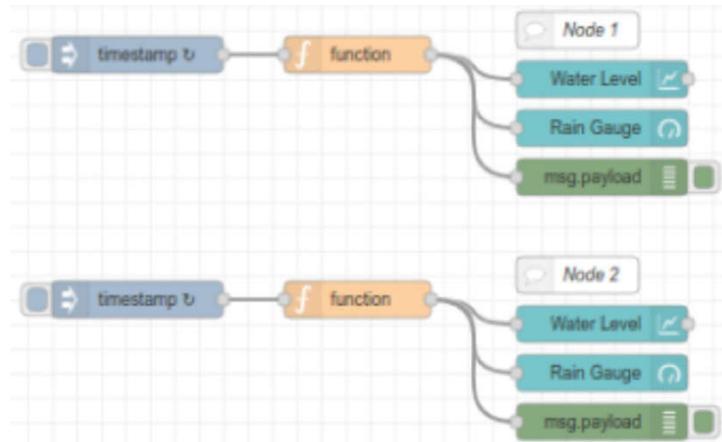
- Setelah itu, buka Tab baru pada Browser, masukkan URL : ip address:1880/ui , untuk mengakses dashboard Node-Red. Tampilan dashboard akan seperti Gambar 6.8. Dokumentasikan hasilnya.



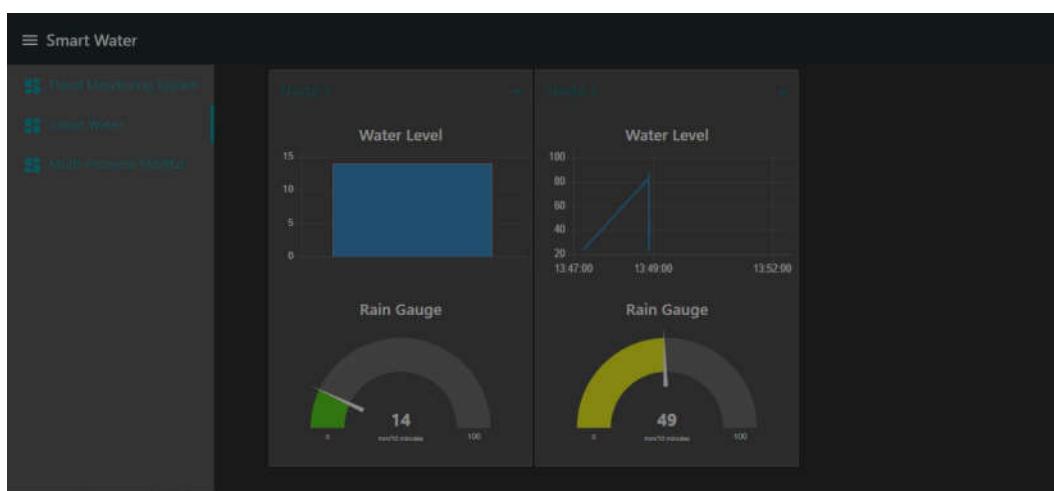
Gambar 6.8. Tampilan dashboard IoT

7. PERTANYAAN DAN TUGAS

1. Buatlah flow program seperti pada Gambar 7.1. Konfigurasi flow tersebut agar mempunyai tampilan seperti Gambar 7.2.
2. Berdasarkan flow program Multi-Protocol Server pada Tugas Jobsheet 5, buatlah agar dashboard dapat menampilkan setiap data yang dikirim dari protokol MQTT, HTTP dan Websocket. Gunakan chart yang relevan untuk menampilkan data dari project yang dibuat. Chart Node 1 merupakan tampilan untuk data dari MQTT. Chart Node 2 merupakan tampilan untuk data dari HTTP. Chart Node 3 merupakan tampilan untuk data dari Websocket. Buatlah diagram alir / flow chart dari program yang dibuat. Kemudian dokumentasikan proses dan outputnya.



Gambar 7.1. Flow program Multi Node Dashboard



Gambar 7.2. Tampilan dashboard Multi Node Dashboard

- 1. NO. JOBSHEET : 10**
- 2. JUDUL : VISUALISASI DATA MENGGUNAKAN GRAFANA**
- 3. TUJUAN**
 - 1) Mahasiswa dapat memahami alur kerja pengiriman data secara end-to end pada sistem IoT.
 - 2) Mahasiswa dapat merancang, melakukan konfigurasi dan menampilkan data sensor pada analytical dashboard Grafana untuk sistem IoT.
- 4. ALAT DAN BAHAN**
 - 1) Komputer terpasang Node Red
 - 2) Sensor Dummy Data Software (MQTT.box, Postman, Simple Websocket Clients)
- 5. TEORI SINGKAT**

Penggunaan internet saat ini telah berkembang dan masuk dalam semua aspek kehidupan manusia. Hampir setiap peralatan yang digunakan oleh manusia terhubung ke internet, seperti mobil, jam tangan, peralatan rumah tangga, bahkan binatang peliharaan pun dapat dipakaikan kalung yang dapat mengirimkan data keberadaannya. Hal tersebut kemudian dikenal sebagai Internet of Things (IoT). IoT memerlukan elemen dashboard/plaform yang dapat mengintrepretasikan dan memvisualisasikan data-data yang dikirim ke server, agar mudah di baca dan dipahami oleh pengguna.

Platform IoT adalah suatu ekosistem yang digabungkan untuk menjadi wadah pembuatan produk dan solusi IoT agar efisien dan tidak memakan banyak waktu. Platform IoT disini sebagai lingkungan IoT yang siap dipakai untuk suatu produk atau bisnis. Platform IoT dapat digunakan untuk mengumpulkan data dari berbagai sumber, menyimpan data, menampilkan data, mengontrol perangkat, mengelola inventory perangkat dan lain-lain. Sebelum membangun sebuah platform, terdapat beberapa hal yang perlu dipertimbangkan, antara lain protokol komunikasi yang digunakan, jenis koneksi, jenis jaringan dan format data.

Salah satu platform yang sering digunakan untuk visualisasi data dan analisis adalah Grafana. Grafana adalah sebuah software opensource yang didesain untuk membaca data metrics untuk kemudian mengubah data-data tersebut menjadi sebuah grafik atau sebuah data tertulis. Software ini banyak sekali digunakan

untuk melakukan analisis data dan monitoring. Grafana mendukung banyak storage backends yang berbeda untuk data time series (Source Data). Setiap source data memiliki Query Editor tertentu yang disesuaikan untuk fitur dan kemampuan tertentu.

6. LANGKAH PERCOBAAN

A. Instalasi Grafana

1. Buka Sakura dan ketikkan script berikut untuk melakukan instalasi Grafana pada VM Ubuntu.

```
sudo apt-get install -y adduser libfontconfig1  
wget https://dl.grafana.com/oss/release/grafana_6.7.0_amd64.deb  
sudo dpkg -i grafana_6.7.0_amd64.deb
```

2. Tunggu hingga proses instalasi selesai.
3. Start Grafana menggunakan script berikut ini.

```
sudo systemctl start grafana-server
```

4. Lakukan pengecekan dan pastikan Grafana telah terpasang dan berjalan pada VM Ubuntu menggunakan script berikut ini.

```
sudo systemctl status grafana-server
```

5. Setelah itu, buat rules pada firewall untuk membuka port 3000, agar Grafana dapat diakses dari komputer lain.

```
sudo ufw allow 3000
```

B. Konfigurasi Grafana

1. Buatlah tabel baru dengan format seperti berikut.

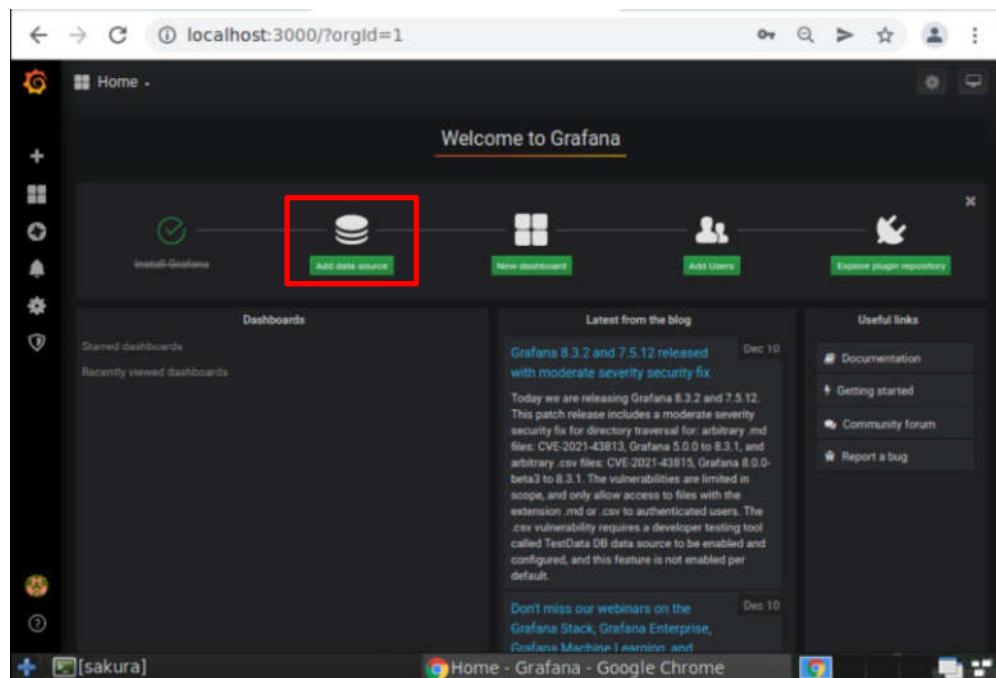
```
CREATE TABLE 'banjir_db'.'smartwater' (  
'id' INT (64) NOT NULL AUTO_INCREMENT,  
'time' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
'dev_id' VARCHAR(64) NULL,
```

```

'level' DECIMAL(16,2) NULL,
'rainfall' DECIMAL(16,2) NULL,
'flow' DECIMAL(16,2) NULL,
PRIMARY KEY ('ID'));

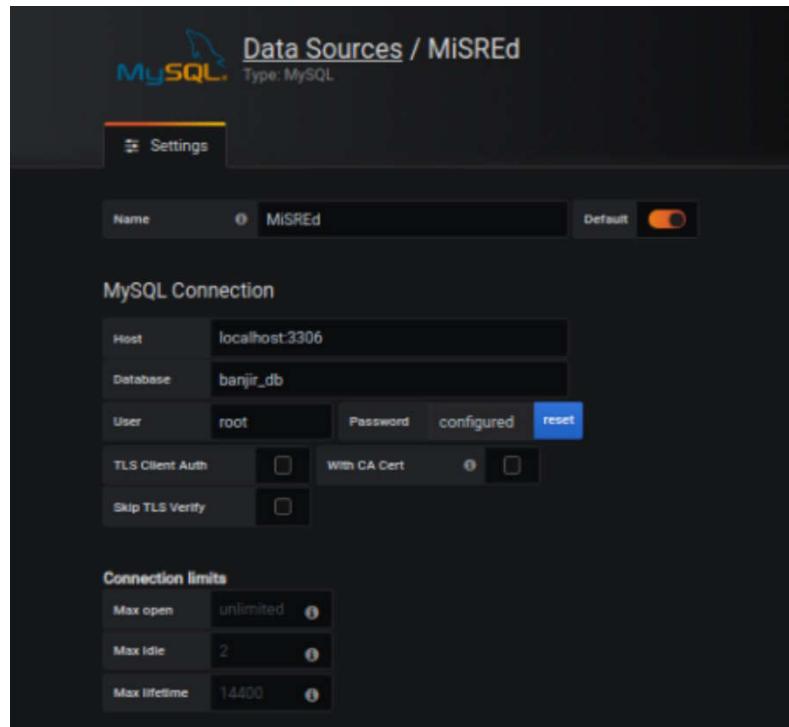
```

2. Login ke Grafana dengan membuka tab baru pada browser. Ketikkan URL ***ip_address_VM_Ubuntu:3000***. Isi username dan password dengan ***admin***. Setelah itu, lakukan penggantian password untuk menjaga keamanan dashboard Grafana.
3. Setelah password autentikasi diganti, Grafana akan mengarahkan pada landing page seperti pada Gambar 6.1. Kemudian, klik menu ***Create a Data Source***.



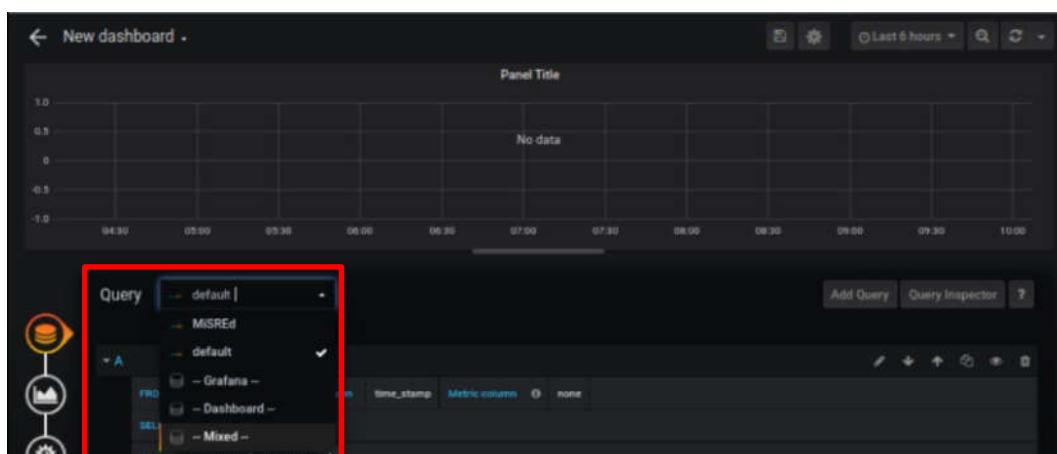
Gambar 6.1. Landing page Grafana

4. Setelah klik menu tersebut, Grafana akan mengarahkan pada pilihan database. Pada kolom pencarian, ketikka ***Mysql*** kemulian klik ***select***.
5. Kemudian, isikan host dan nama database dan credential database seseuai dengan database yang telah dibuat sebelumnya atau seperti pada Gambar 6.2.



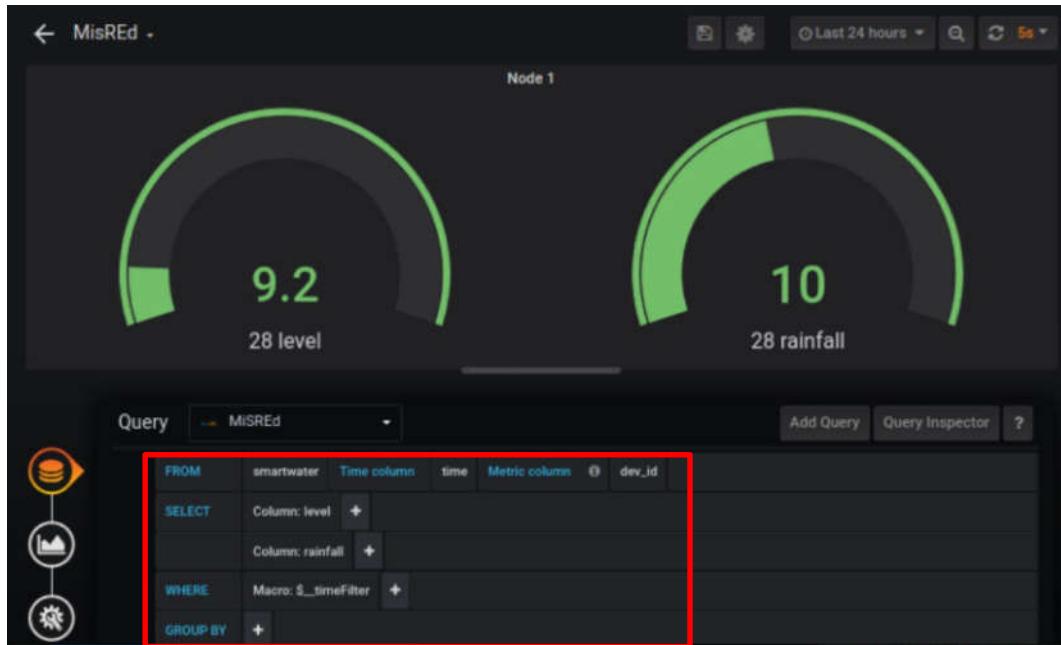
Gambar 6.2. Konfigurasi data sources

6. Setelah itu kembali ke landing page dan pilih menu ***Build a Dashboard***.
7. Kemudian, Grafana akan mengarahkan pada tampilan untuk memilih query dan virtualisasi. Pada tahap ini, klik menu **Add Query**.
8. Setelah klik menu tersebut, Grafana akan mengarahkan pada tampilan konfigurasi tampilan grafik/chart dashboard yang dikehendaki user.
9. Pada sub menu Query, klik dropdown menu, kemudian pilih data sources yang telah dibuat sebelumnya (MiSREd). Hal tersebut dapat dilihat pada Gambar 6.3.



Gambar 6.3. Memilih data sources pada setting dashboard

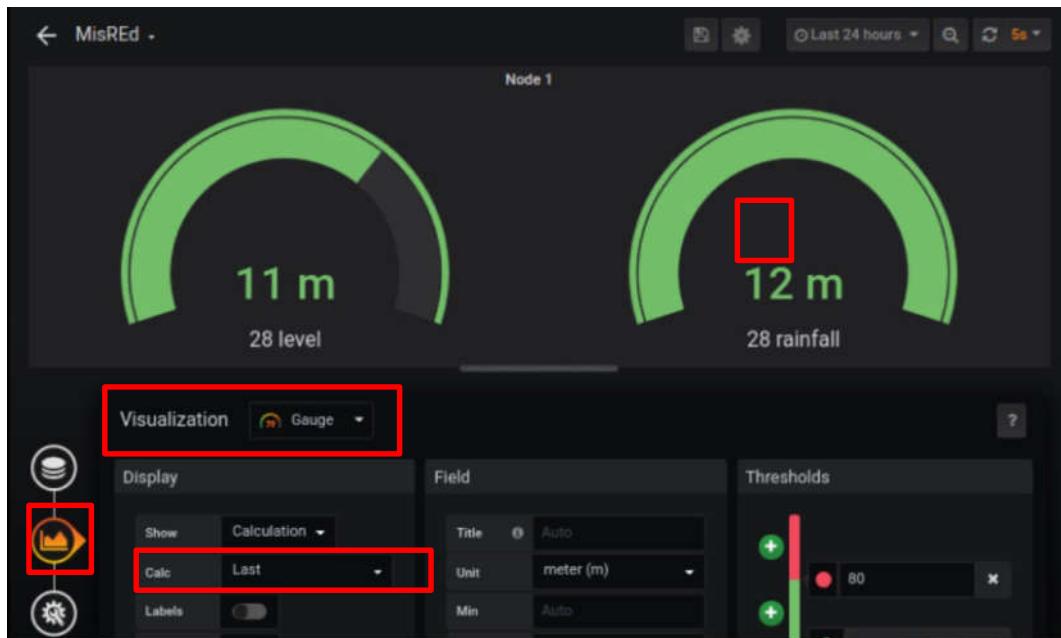
10. Setelah itu sesuaikan query seperti pada Gambar 6.4.



Gambar 6.4. Konfigurasi query database

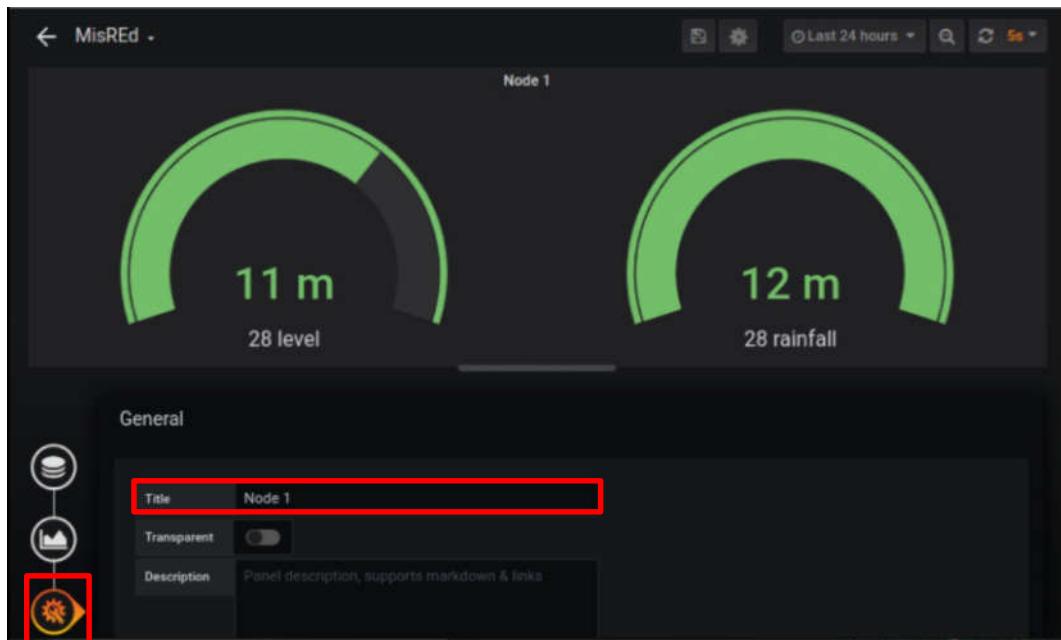
11. Untuk mengganti tampilan grafik, klik pada menu bilah kiri urutan kedua.

Kemudian sesuaikan konfigurasi, agar data yang tertampil merupakan data terakhir yang masuk pada Database. Hal tersebut dapat dilihat pada gambar 6.5



Gambar 6.5. Konfigurasi tampilan dashboard

- Untuk mengganti judul pada dashboard, klik menu bilah kiri urutan ketiga, seperti Gambar 6.6. Kemudian, isikan judul yang sesuai.



Gambar 6.6. Konfigurasi judul dashboard

- Setelah konfigurasi selesai, klik menu save pada menu bilah atas.
- Dokumentasikan hasil pembuatan dashboard.

7. PERTANYAAN DAN TUGAS

- Buatlah flow program Multi-Protocol Server untuk aplikasi Smart-Home yang terdiri dari sensor suhu, sensor kelembapan, sensor cahaya, sensor inframerah dan sensor gas methane. Buatlah agar setiap sensor mempunyai tampilan grafik yang berbeda.

1. NO. JOBSHEET : 11**2. JUDUL : MEMBANGUN SISTEM SMART PARKING
BERBASIS MICRO-SERVICES****3. TUJUAN**

- 1) Mahasiswa dapat memahami terkait arsitektur micro-services untuk membangun layanan IoT.
- 2) Mahasiswa dapat melakukan instalasi docker, backend dan frontend server berbasis Tarantool.

4. ALAT DAN BAHAN

- 1) Komputer dengan VM Ubuntu 18.04.

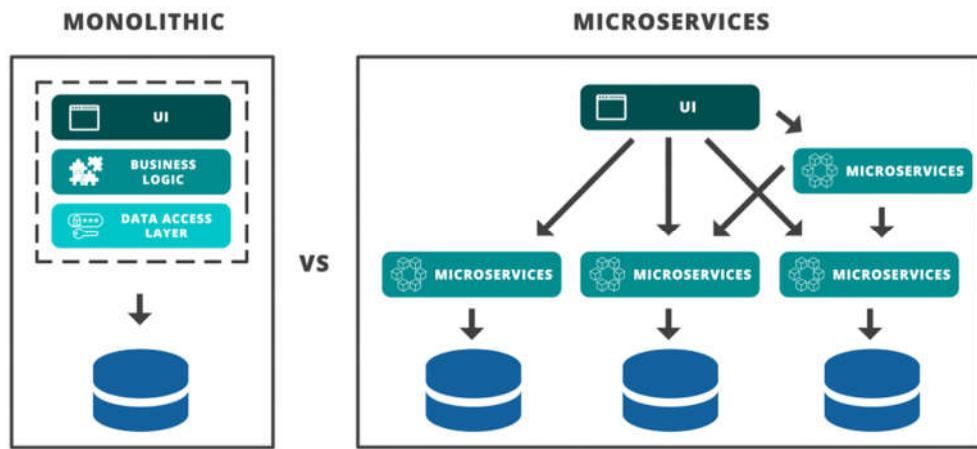
5. TEORI SINGKAT

Microservices adalah suatu framework Architecture yang dipakai sebagai model dalam pembuatan aplikasi cloud yang modern. Di dalam microservices setiap aplikasi dibangun sebagai sekumpulan service dan setiap layanan berjalan dalam processnya sendiri. Masing-masing dari aplikasi tersebut saling berkomunikasi melalui API (Application Programming Interface).

Microservices merupakan Architecture dalam pembangunan aplikasi cloud yang modern. Architecture microservices bersifat terdistribusi, sehingga perubahan pada program yang dilakukan oleh satu tim developer tidak mengganggu keseluruhan aplikasi. Manfaat utama dalam mempergunakan microservices adalah agar team developer mampu mengembangkan aplikasi secara cepat dengan membuat komponen-komponen dari aplikasi berjalan secara independen sehingga dapat memenuhi kebutuhan bisnis yang terus menerus berubah.

Cara pembangunan aplikasi yang seperti ini dioptimalkan dengan mempergunakan DevOps (Development and Operation) dan CI / CD (Continuous Integration and Continuous Delivery). Perbedaan Architecture Microservices dengan pendekatan yang lebih tradisional seperti Monolithic Architecture adalah bagaimana framework ini memecah aplikasi menjadi fungsi intinya. Setiap fungsi ini disebut sebagai service, dapat dibangun dan dijalankan secara independen, yang berarti service tersebut dapat berfungsi (dan gagal) tanpa berdampak negatif pada fungsi-fungsi yang lain. Framework ini membantu dalam sisi teknologi dari

DevOps dalam pelaksanaan continuous integration dan continuous delivery (CI / CD) sehingga menjadi lebih mulus dan dapat tercapai.



6. LANGKAH PERCOBAAN

1. Lakukan instalasi docker dan docker compose.
2. Buatlah container pada docker.
3. Install Tarantool images pada container yang telah dibuat.
4. Install NGINX dan NGINX upstream module images pada container yang berbeda.
5. Konfigurasi Tarantool seperti pada resources pada link berikut ini.

<https://drive.google.com/drive/folders/1TXejjX-a9T7OGBDvmbiVgokSUapMjRIa?usp=sharing>

6. Jalankan program tersebut hingga berfungsi seperti pada video link di atas.
7. Dokumentasikan hasilnya dalam bentuk video.

7. PERTANYAAN DAN TUGAS

1. NO. JOBSHEET : 12

**2. JUDUL : MEMBANGUN SISTEM SMART WATER
PEMANTAU LIMBAH**

3. TUJUAN

- 1) Mahasiswa dapat memahami dan merancang sistem IoT pemantau parameter limbah industri.
- 2) Mahasiswa dapat melakukan fabrikasi sistem IoT pemantau limbah.

4. ALAT DAN BAHAN

- 1) Komputer terinstal Arduino IDE dan Easy EDA.
- 2) Komponen sensor dan komponen elektronik lengkap.

5. TEORI SINGKAT

Sistem pemantauan limbah cair industri adalah suatu sistem yang dirancang untuk memantau dan mengelola limbah cair yang dihasilkan oleh industri. Limbah cair ini dapat mencakup air buangan, limbah proses, atau sisa-sisa produksi yang berpotensi mencemari lingkungan. Sistem ini bertujuan untuk memastikan bahwa limbah cair yang dihasilkan memenuhi standar lingkungan yang ditetapkan oleh pemerintah dan lembaga lingkungan terkait.

SPARING adalah singkatan dari "Sistem Pemantauan Limbah Cair Industri." Sistem ini merupakan inisiatif yang dikeluarkan oleh Kementerian Lingkungan Hidup dan Kehutanan Indonesia untuk memantau dan mengelola limbah cair industri di seluruh negara. Tujuannya adalah untuk meningkatkan pengawasan dan pengendalian limbah cair industri serta memastikan kepatuhan terhadap peraturan lingkungan.

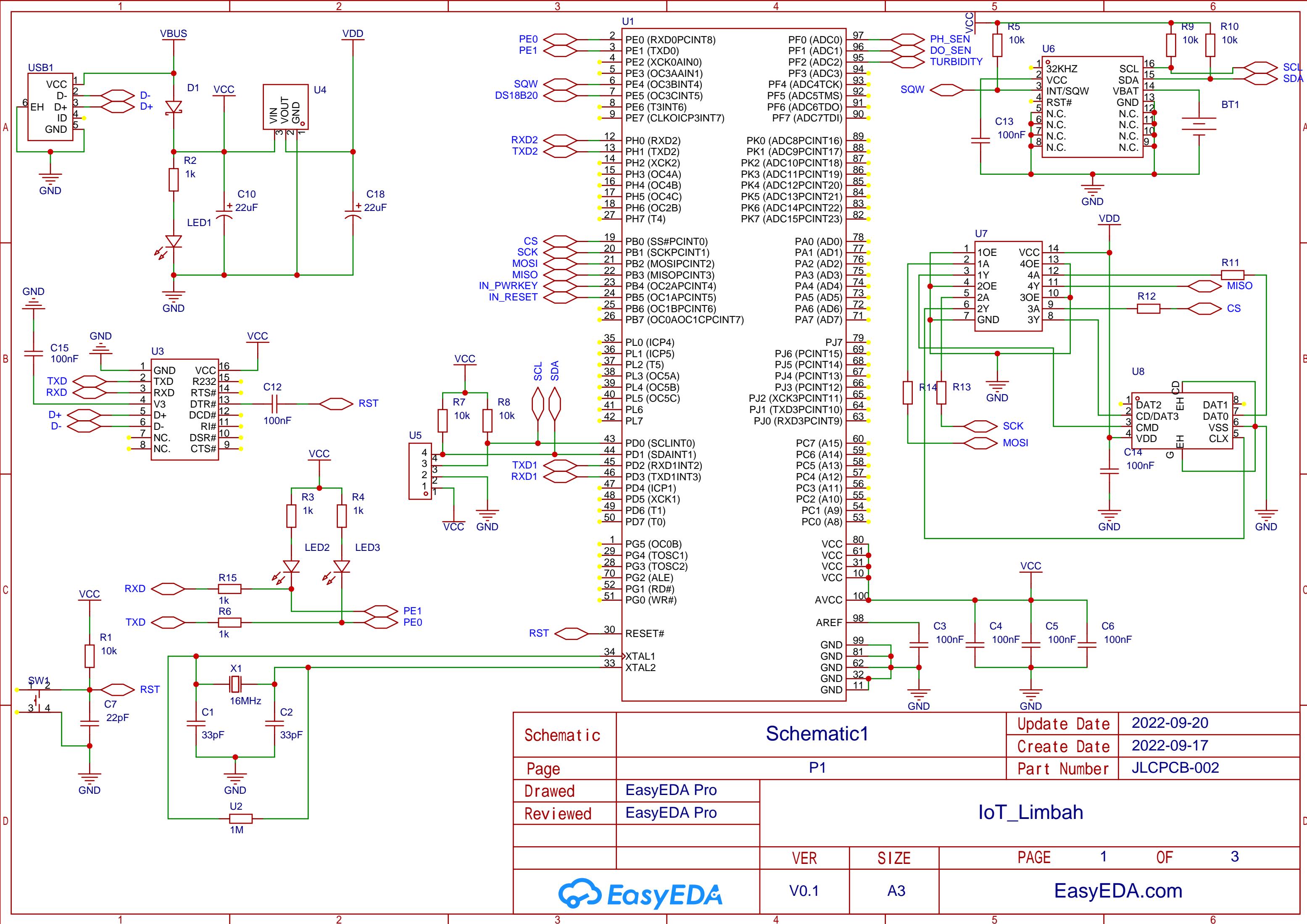
SPARING terkoneksi dengan server yang dioperasikan oleh Kementerian Lingkungan Hidup dan Kehutanan Indonesia. Koneksi ini memungkinkan pengumpulan, penyimpanan, dan analisis data limbah cair industri secara terpusat. Melalui koneksi ini, data limbah cair yang terkumpul dapat dengan mudah diakses dan digunakan oleh pemerintah untuk tujuan pengawasan, evaluasi kepatuhan, dan pengambilan keputusan berdasarkan data yang akurat dan real-time. Untuk mengintegrasikan perangkat SPARING dan Server, dibutuhkan API dan data yang telah diformat dalam bentuk Jason Web Token (JWT).

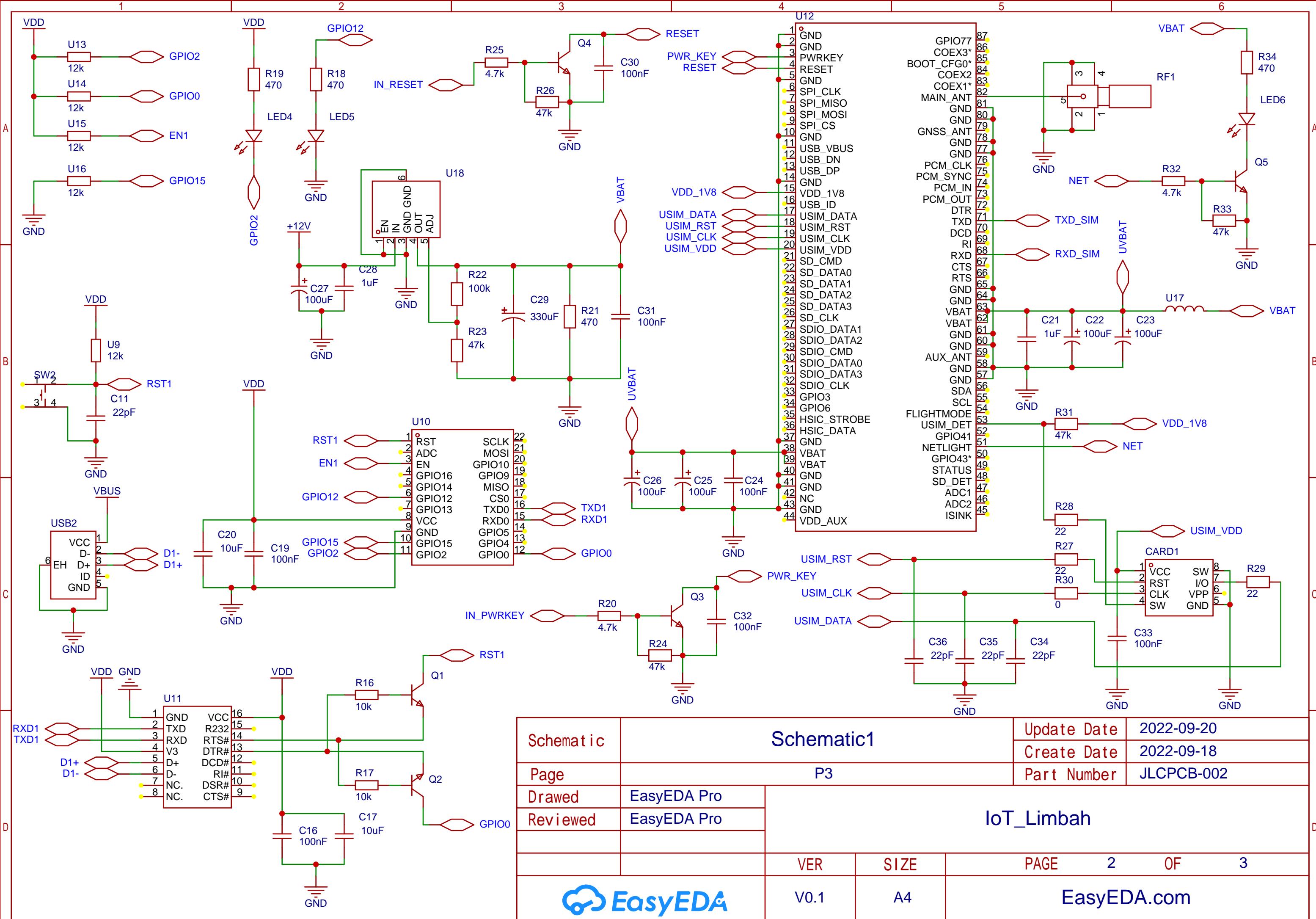
6. LANGKAH PERCOBAAN

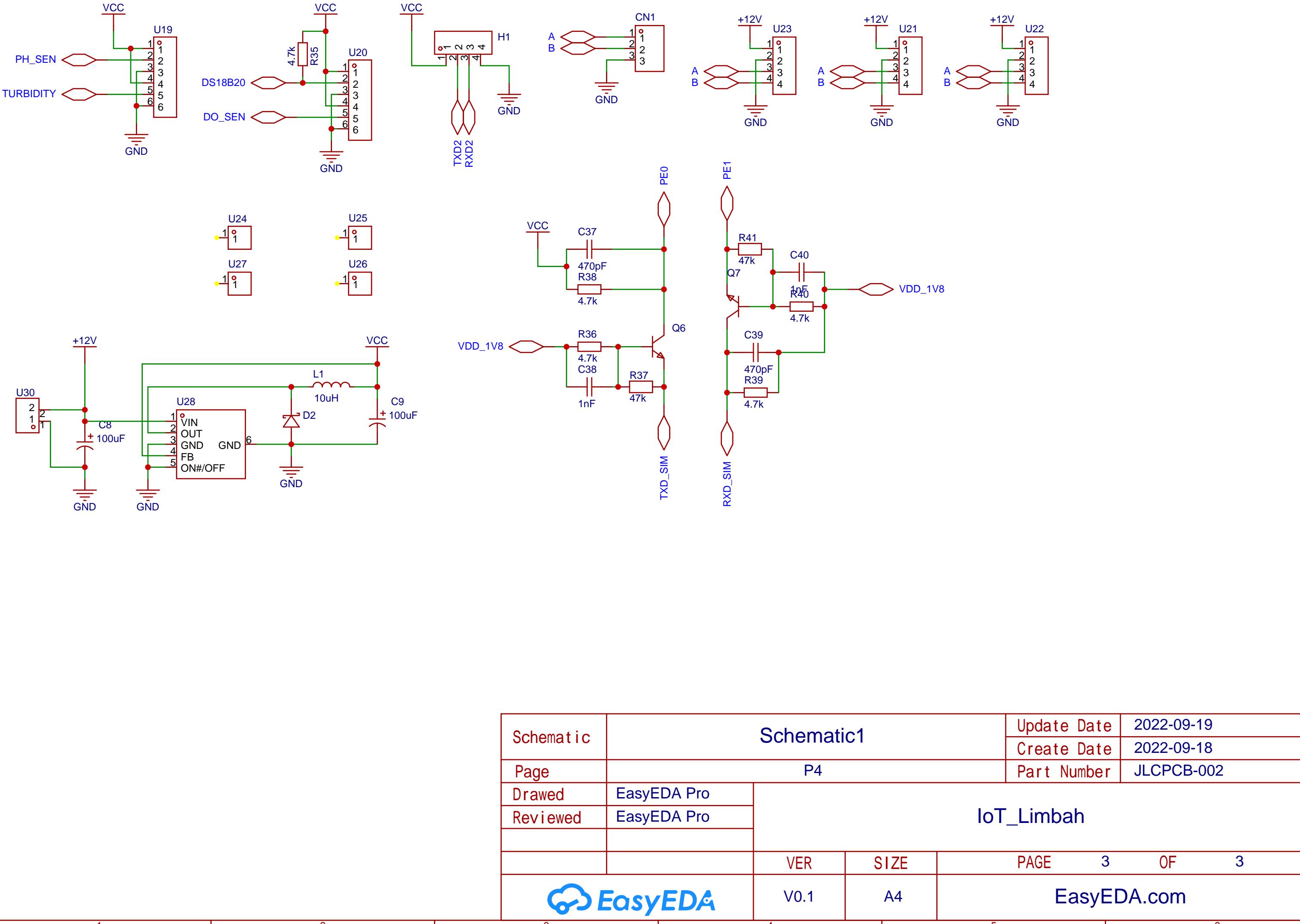
1. Buatlah desain rangkaian elektronik menggunakan perangkat lunak Easy Eda seperti yang ditunjukkan gambar di lampiran.
2. Buatlah desain PCB dan atur tata letak komponen seperti yang ditunjukkan gambar di lampiran.
3. Cetak desain PCB yang telah dibuat.
4. Lakukan kontrol kualitas pada hasil cetakan PCB.
5. Pasang semua komponen elektronik.
6. Lakukan kontrol kualitas pada hasil pemasangan komponen elektronik (test short circuit, dll).
7. Buatlah program untuk membaca sensor yang menggunakan protokol Modbus (sensor COD dan sensor NH3-N).
8. Buatlah program untuk membaca sensor pH dan TSS/Turbidity berbasis data analog.
9. Buatlah program untuk membaca sensor flow meter berbasis data digital.
10. Buatlah program timer untuk pengaturan pengiriman data sensor setiap 2 menit sekali.
11. Buatlah program untuk konversi data waktu dari RTC menjadi format epoch.
12. Buatlah program untuk konversi data sensor menjadi token JWT.
13. Buatlah program untuk pengiriman data menggunakan modul SIM7600E.
14. Pastikan semua program berjalan sesuai sengan SOP yang disyaratkan KLHK.

7. PERTANYAAN DAN TUGAS

Selesaikan pekerjaan ini dalam waktu 16 minggu perkuliahan, lakukan juga presentasi dan pelaporan update pekerjaan setiap mingguan.







Schematic	Schematic1			Update Date	2022-09-19
Page	P4			Create Date	2022-09-18
Drawn	EasyEDA Pro			Part Number	JLCPCB-002
Reviewed	EasyEDA Pro			IoT_Limbah	
			VER	SIZE	PAGE 3 OF 3
			V0.1	A4	EasyEDA.com

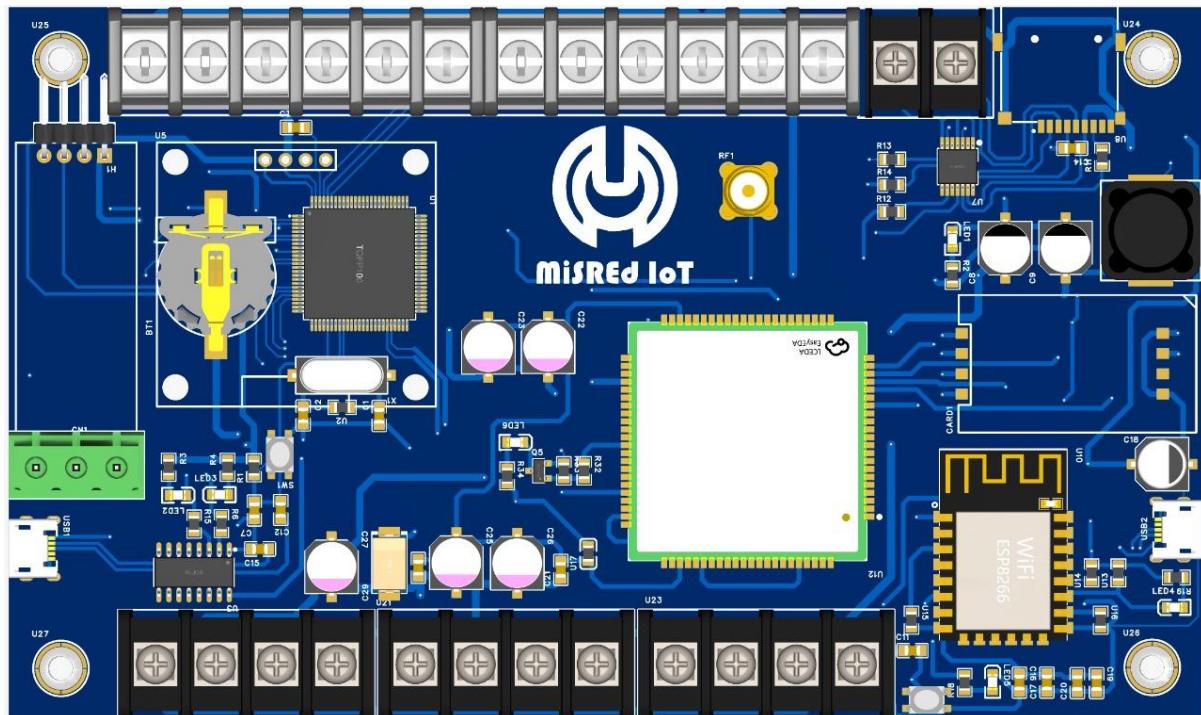


KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI SEMARANG
JURUSAN TEKNIK ELEKTRO

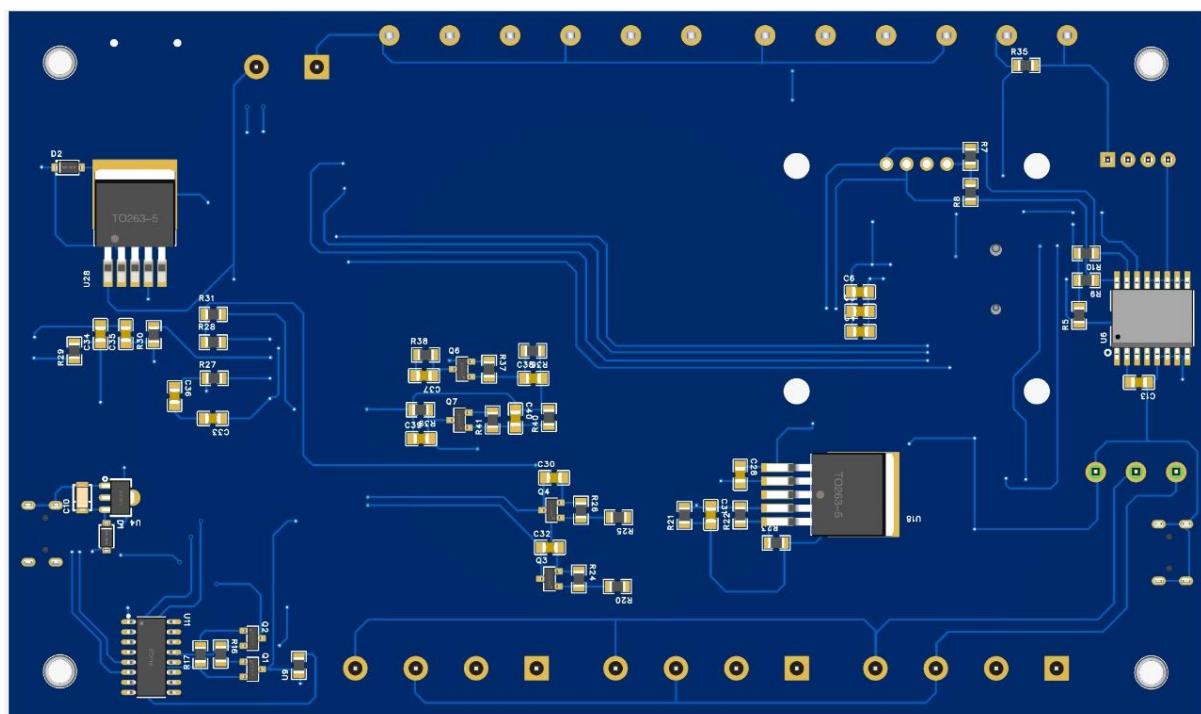
RENCANA PELAKSANAAN PROYEK

NO.:xx/TE/01/2022 SEM : 5 SKS: 3 6 JP Revisi: 00 Tanggal xx/10/ 2022

3. Konstruksi Produk



Gambar 3. Papan sirkuit tampak atas



Gambar 4. Papan sirkuit tampak bawah

TATA CARA UJI KONEKTIVITAS SPARING - SECARA ONLINE

A. Alat dan Perlengkapan yang harus disiapkan

Pihak perusahaan wajib Sparing/ penyedia barang dan jasa alat sparing wajib menyiapkan peralatan yaitu

- a. Alat sensor SPARING (spesifikasi WAJIB sesuai dengan PermenLHK No. P.93/2018 Jo No. P.80/2019 Tentang SPARING)
- b. Data Logger
- c. Air limbah (bisa digantikan air murni dengan cairan buffer atau air kolam ikan)
- d. Wadah untuk sampel air limbah
- e. Kamera (digunakan untuk foto dan video dokumentasi uji konektivitas alat sparing)

B. Mekanisme Uji Konektivitas Alat Sensor Pendaftaran Baru

1. Pihak perusahaan wajib Sparing/ penyedia barang dan jasa alat sparing wajib menggunakan sensor dan data logger yang saling terhubung dan langsung mengirimkan data ke server Sparing KLHK (**TIDAK DIIZINKAN MENGGUNAKAN DUMMY DATA**). Jika diketahui menggunakan *dummy* data, KLHK tidak akan melaksanakan proses uji konektivitas Sparing KLHK.
2. **Kamera atau video recorder untuk melakukan proses perekaman (Video) dan foto saat aktivitas uji konektivitas.** Hasil dokumentasi dikirimkan ke email sparing.menlhk@gmail.com, sebagai pelengkap dokumen proses uji konektivitas.
3. Tahap uji koneksi:

i. Test Tahap 1

1. Pihak penyedia jasa dan alat sparing melakukan pengiriman data sensor sparing melalui alat data logger dengan interval atau jarak waktu 2 menit sekali
2. Setelah melakukan pengiriman data tersebut maka pihak KLHK akan mengkonfirmasi diterima atau tidak diterima data dari alat sparing.
3. Jika KLHK mengkonfirmasi data telah masuk dan interval waktu 2 menit sudah tepat, maka pihak penyedia melanjutkan pengiriman datanya sampai pihak KLHK memberhentikan proses tersebut dan melanjutkan proses tahap 2
4. Jika poin 3 belum berhasil maka pihak penyedia diminta memperbaiki kesalahan pengiriman data.

ii. Test Tahap 2

1. Pada test ini akan diuji kesiapan data logger sparing, pada kondisi lokasi pemasangan alat sparing memiliki sumber internet yang tidak baik.
2. Alat data logger wajib dapat menyimpan data sensor sparing sesuai dengan waktu disaat internet tidak terhubung, asumsi jika terdapat waktu selama 10 menit internet tidak terhubung, maka akan tersimpan 5 data alat sparing (@ 2 menit sekali).
3. Ketika alat data logger terhubung sumber internet maka 5 data yang hilang akan terkirim kembali ke server KLHK dengan waktu pencatatan sesuai dengan waktu yang tidak terhubung ke internet

4. Pihak penyedia akan diminta mematikan sumber internet yang terhubung ke alat data logger selama 15 menit. Dan mengkonfirmasi pihak KLHK bahwa internet sudah dimatikan
5. Setelah 15 menit maka sumber internet dihidupkan kembali, dan konfirmasi ke KLHK.
6. Pihak KLHK akan melakukan pengecekan atas data yang hilang tersebut, jika data terkonfirmasi sudah masuk maka pihak penyedia dapat melanjutkan proses tahap 3
7. Jika poin 6 belum berhasil maka pihak penyedia diminta memperbaiki kesalahan pengiriman data tersebut.

iii. Test Tahap 3

1. Pada test ini akan diuji untuk data logger mengirimkan data sensor dalam waktu 24 jam, test ini akan dilihat apakah interval data sensor yang dikirimkan data logger sesuai 2 menit sekali.
2. Setelah tahap 2 lulus maka pihak penyedia diminta menyalakan alat sparingnya selama 24 jam nonstop.
3. Pihak KLHK akan membuat jadwal waktu dengan pihak penyedia untuk pengecekan data sensor yang dikirimkan data logger
4. Jika pada poin 3 data sesuai dengan interval waktu 2 menit maka akan dinyatakan lulus. Jika masih ada kesalahan akan dilakukan test ulang untuk tahap 3 dengan jadwal yang ditentukan kembali.

C. Mekanisme Uji Konektivitas Ulang (Alat Sensor dan Logger sudah Lulus Uji Koneksi sesuai Surat Keterangan KLHK)

1. Pihak perusahaan wajib Sparing/ penyedia barang dan jasa alat sparing wajib menggunakan sensor dan data logger **yang saling terhubung dan langsung mengirimkan data** ke server Sparing KLHK (**TIDAK DIIZINKAN MENGGUNAKAN DUMMY DATA**). Jika diketahui menggunakan *dummy* data, KLHK tidak akan melaksanakan proses uji konektivitas Sparing KLHK.
2. **Kamera atau video recorder untuk melakukan proses perekaman (Video) dan foto saat aktivitas uji konektivitas/ jika alat sensor sudah dilakukan pemasangan di perusahaan, maka mengirimkan video atau foto di lokasi tersebut.** Hasil dokumentasi dikirimkan ke email sparing.menlhk@gmail.com, sebagai pelengkap dokumen proses uji konektivitas.
3. Tahap uji koneksi:
 - a. Pihak penyedia jasa dan alat sparing melakukan pengiriman data sensor sparing melalui alat data logger dengan interval atau jarak waktu 2 menit sekali
 - b. Setelah melakukan pengiriman data tersebut maka pihak KLHK akan mengkonfirmasi diterima atau tidak diterima data dari alat sparing.
 - c. Jika KLHK mengkonfirmasi data telah masuk dan interval waktu 2 menit sudah tepat, maka pihak penyedia melanjutkan pengiriman datanya sampai pihak KLHK memberhentikan proses tersebut dan melanjutkan proses tahap 2
 - d. Jika poin 3 belum berhasil maka pihak penyedia diminta memperbaiki kesalahan pengiriman data.

D. Ketentuan Umum Pengiriman Data:

- a. Koneksi menggunakan Internet port 80
- b. Konsep API lintas platform
- c. Pengiriman data menggunakan POST METHOD
- d. Data yang dikirimkan dalam format JSON (JavaScript Object Notation) yang di encode dalam bentuk JWT (JSON Web Token)
- e. Data dikirimkan 2 menit sekali
- f. Koneksi URL API menggunakan autentikasi ID
- g. Menggunakan GET METHOD untuk pengambilan JWT Secret Key (secret key tidak boleh static dikarenakan akan berubah secara berkala)

E. API Lintas Server :

- a. API Uji Konektivitas SPARING

- 1. JWT GET SECRET API ENDPOINT:

<http://203.166.207.50/api/klhk/secret-sensor>

- 2. POST API ENDPOINT:

<http://203.166.207.50/api/server-uji>

Contoh Pengiriman data :

- a. Konektivitas untuk **Uji KONEKTIVITAS SPARING**

- i. JWT GET SECRET API ENDPOINT:

<http://203.166.207.50/api/klhk/secret-sensor>

- ii. POST API ENDPOINT:

<http://203.166.207.50/api/server-uji>

- iii. JWT Payload JSON sebelum di encode sebagai berikut:

```
{  
    "uid": "1121100500014",  
    "datetime":1567756309,  
    "pH": 7,  
    "cod": 150,  
    "tss": 80,  
    "nh3n":10  
    "debit":10  
  
}
```

Tipe data payload:

uid - String (akan diberikan dari pihak penguji)

datetime - Number (Integer Timestamp)

pH - Number (Double/Float)

cod - Number (Double/Float)

tss - Number (Double/Float)

nh3n - Number (Double/Float)

- iv. JSON berisi Encoded Payload yang akan dikirimkan ke server melalui POST API ENDPOINT sebagai berikut:

```
{  
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1aWQiOjEx  
MjA4MDAzMDAwMTQsImRhdGV0aW1lIjoxNTY4NjMwMTQ5LCJwSCI6MTg2  
OTIsImNvZCI6NTUwOCwidHNzIjo1NDY2LCJuaDNuIjoxNjUzOSwiZGViaXQiOj  
E3MDA2fQ.vThMTJA2wMElKh2_uWoG45XdHj5jH3MOfUde88bSYzY"  
}
```

- v. Jika berhasil melakukan pengiriman data maka API akan memberikan respon “Data Sent Successfully!”