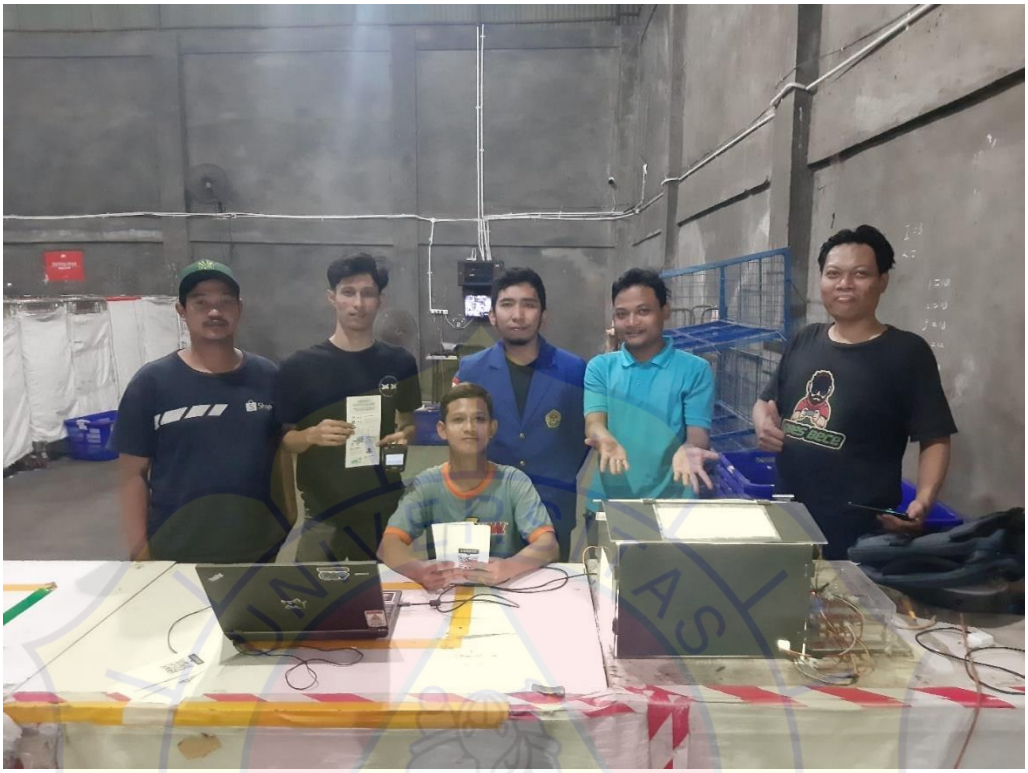
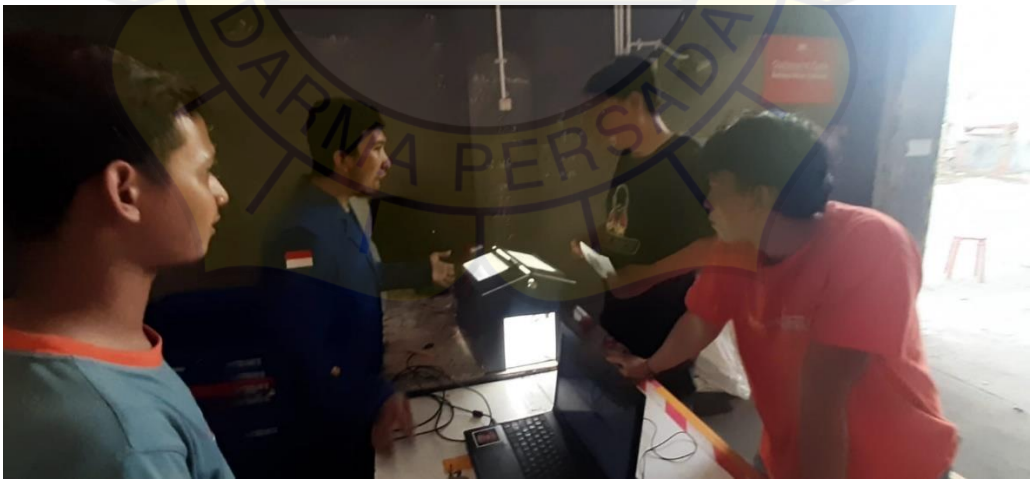
Lampiran 3. Foto Tempat Penelitian





Lampiran 4. Kode Program Arduino IDE (smartwarehouse.ino)

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecure.h>
#include <dhtnew.h>
#include <Wire.h>
#include <ArtronShop_BH1750.h>
#include <MQUnifiedsensor.h>
#define placa "ESP8266" // Define the board type
#define Voltage_Resolution 3.3
#define pin A0
#define type "MQ-135" // MQ135
#define ADC_Bit_Resolution 10
#define RatioMQ135CleanAir 3.6 // RS / R0 = 3.6 ppm
// Konfigurasi sensor MQ-135
MQUnifiedsensor MQ135(placa, Voltage_Resolution,
ADC_Bit_Resolution, pin, type);
const float CO_A = 605.18; // Value of a for CO
const float CO_B = -3.937; // Value of b for CO
#define relayPin_CO 12
#define CO_THRESHOLD 10.0 // Threshold value for CO
concentration to activate relay
// Konfigurasi WiFi
const char *ssid = "DRAZERID";         // Ganti dengan SSID WiFi
Anda
const char *password = "085710791341"; // Ganti dengan password
WiFi Anda
// Konfigurasi server
const char *host = "iot.mansyahtech.biz.id"; // Ganti dengan
alamat IP / Domain server Anda
// Konfigurasi sensor DHT21
DHTNEW mySensor(16); // Ganti dengan pin yang Anda gunakan,
misalnya: ESP 16
// Konfigurasi sensor BH1750
ArtronShop_BH1750 bh1750(0x23, &Wire); // Non Jump ADDR: 0x23,
Jump ADDR: 0x5C
const int relayPin_Light = 13;         // Pin untuk relay cahaya
float threshold_Light = 100.0;         // Ambang batas
intensitas cahaya untuk menghidupkan/mematikan lampu
// Konfigurasi sensor PIR
const int pirPin = 0;       // Pin untuk sensor PIR
const int relayPinPIR = 14; // Pin untuk relay PIR
unsigned long lastMotionTime = 0;
const unsigned long motionTimeout = 10000; // Waktu tunda (ms)
untuk mematikan kipas setelah tidak ada gerakan
bool lastPIRState = LOW;                   // Status terakhir
dari sensor PIR

// Konfigurasi sensor KY-026 dan buzzer
```

```
int flamePin = 15; // Pin yang terhubung dengan OUT sensor KY-
026
int relayPin_buzzer = 2; // Pin yang terhubung dengan buzzer
// Definisikan Token dan Chat ID Telegram
const String botToken =
"7498640822:AAFONH9MBEw0fcr52WaAvosFwsXcaMJBdSc";
const String chatID = "277716394";

WiFiClient wifiClient;

// Fungsi untuk mengirim ping ke server
void sendPing()
{
  if (WiFi.status() == WL_CONNECTED)
  {
    HTTPClient http;
    String serverPath = String("http://") + host + "/ping";

    Serial.print("Sending GET request to ");
    Serial.println(serverPath);

    http.begin(wifiClient, serverPath);
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0)
    {
      Serial.println("Ping sent to server");
      String response = http.getString();
      Serial.println(httpResponseCode);
      Serial.println(response);
    }
    else
    {
      Serial.print("Error in sending ping: ");
      Serial.println(httpResponseCode);
    }

    http.end();
  }
  else
  {
    Serial.println("WiFi not connected");
  }
}

void sendBH1750Data(float lightIntensity) {
  // Variabel statis untuk menyimpan intensitas cahaya terakhir
yang dikirim
  static float lastLightIntensity = 0;
```

```
  // Memeriksa apakah intensitas cahaya saat ini berbeda dengan
intensitas cahaya terakhir yang dikirim
  if (lightIntensity != lastLightIntensity) {
    // Memperbarui intensitas cahaya terakhir
    lastLightIntensity = lightIntensity;

    // Memeriksa status koneksi WiFi
    if (WiFi.status() == WL_CONNECTED) {
      // Membuat instance HTTPClient untuk mengirimkan
permintaan HTTP
      HTTPClient http;
      // Membentuk URL server untuk mengirim data
      String serverPath = String("http://") + host +
"/receive_bh1750_data";

      // Menampilkan URL server yang akan dikirimi permintaan
      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      // Memulai koneksi ke server dengan URL yang telah
dibentuk
      http.begin(wifiClient, serverPath);
      // Menambahkan header "Content-Type" dengan nilai
"application/json"
      http.addHeader("Content-Type", "application/json");

      // Membentuk data permintaan dalam format JSON
      String httpRequestData = "{\"light_intensity\":" +
String(lightIntensity, 1) + "}";
      // Menampilkan data permintaan yang akan dikirim
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      // Mengirimkan permintaan POST dengan data JSON
      int httpResponseCode = http.POST(httpRequestData);

      // Memeriksa kode respons dari server
      if (httpResponseCode > 0) {
        // Jika berhasil, menampilkan kode respons dan respons
dari server
        String response = http.getString();
        Serial.println(httpResponseCode);
        Serial.println(response);
      } else {
        // Jika gagal, menampilkan pesan kesalahan
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }
      // Mengakhiri koneksi HTTP
      http.end();
    } else {
```

```
      // Jika tidak terhubung ke WiFi, menampilkan pesan
kesalahan
      Serial.println("Error in WiFi connection");
    }
  }
}

void sendPIRData(bool motion_detected) {
  // Variabel statis untuk menyimpan status terakhir deteksi
gerakan
  static bool lastMotionDetected = false;

  // Memeriksa apakah gerakan terdeteksi dan status deteksi
gerakan berubah dari status terakhir
  if (motion_detected && motion_detected != lastMotionDetected)
{
    // Memperbarui status terakhir deteksi gerakan
    lastMotionDetected = motion_detected;

    // Memeriksa status koneksi WiFi
    if (WiFi.status() == WL_CONNECTED) {
      // Membuat instance HTTPClient untuk mengirimkan
permintaan HTTP
      HTTPClient http;
      // Membentuk URL server untuk mengirim data
      String serverPath = String("http://") + host +
"/receive_pir_data";

      // Menampilkan URL server yang akan dikirimi permintaan
      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      // Memulai koneksi ke server dengan URL yang telah
dibentuk
      http.begin(wifiClient, serverPath);
      // Menambahkan header "Content-Type" dengan nilai
"application/json"
      http.addHeader("Content-Type", "application/json");

      // Membentuk data permintaan dalam format JSON
      String httpRequestData = "{\"motion_detected\":" +
String(motion_detected) + "}";
      // Menampilkan data permintaan yang akan dikirim
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      // Mengirimkan permintaan POST dengan data JSON
      int httpResponseCode = http.POST(httpRequestData);

      // Memeriksa kode respons dari server
      if (httpResponseCode > 0) {
```

```
        // Jika berhasil, menampilkan kode respons dan respons
dari server
        String response = http.getString();
        Serial.println(httpResponseCode);
        Serial.println(response);
      } else {
        // Jika gagal, menampilkan pesan kesalahan
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }

      // Mengakhiri koneksi HTTP
      http.end();
    } else {
      // Jika tidak terhubung ke WiFi, menampilkan pesan
kesalahan
      Serial.println("Error in WiFi connection");
    }
  } else if (!motion_detected) {
    // Jika gerakan tidak terdeteksi, mengatur status terakhir
deteksi gerakan menjadi false
    lastMotionDetected = false;
  }
}

// Fungsi untuk mengirim data DHT21 ke server hanya jika terjadi
perubahan data
void sendDHTData(float temperature, float humidity) {
  // Variabel statis untuk menyimpan suhu dan kelembaban
terakhir yang dikirim
  static float lastTemperature = -1.0;
  static float lastHumidity = -1.0;

  // Memeriksa apakah suhu atau kelembaban saat ini berbeda
dengan yang terakhir dikirim
  if (temperature != lastTemperature || humidity !=
lastHumidity) {
    // Memperbarui suhu dan kelembaban terakhir
    lastTemperature = temperature;
    lastHumidity = humidity;

    // Memeriksa status koneksi WiFi
    if (WiFi.status() == WL_CONNECTED) {
      // Membuat instance HTTPClient untuk mengirimkan
permintaan HTTP
      HTTPClient http;
      // Membentuk URL server untuk mengirim data
      String serverPath = String("http://") + host +
"/receive_dht21_data";

      // Menampilkan URL server yang akan dikirimi permintaan
```

```
      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      // Memulai koneksi ke server dengan URL yang telah
dibentuk
      http.begin(wifiClient, serverPath);
      // Menambahkan header "Content-Type" dengan nilai
"application/json"
      http.addHeader("Content-Type", "application/json");

      // Membentuk data permintaan dalam format JSON
      String httpRequestData = "{\"temperature\":" +
String(temperature, 1) + ",\"humidity\":" + String(humidity, 1)
+ "}";
      // Menampilkan data permintaan yang akan dikirim
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      // Mengirimkan permintaan POST dengan data JSON
      int httpResponseCode = http.POST(httpRequestData);

      // Memeriksa kode respons dari server
      if (httpResponseCode > 0) {
        // Jika berhasil, menampilkan kode respons dan respons
dari server
        String response = http.getString();
        Serial.println(httpResponseCode);
        Serial.println(response);
      } else {
        // Jika gagal, menampilkan pesan kesalahan
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }

      // Mengakhiri koneksi HTTP
      http.end();
    } else {
      // Jika tidak terhubung ke WiFi, menampilkan pesan
kesalahan
      Serial.println("Error in WiFi connection");
    }
  }
}

void sendCOData(float co) {
  // Variabel statis untuk menyimpan nilai CO terakhir yang
dikirim
  static float lastCO = -1; // Initialize to a value that will
ensure the first reading is sent
```

```
  // Memeriksa apakah nilai CO saat ini berbeda dengan nilai CO
terakhir yang dikirim
  if (co != lastCO) {
    // Memperbarui nilai CO terakhir
    lastCO = co;

    // Memeriksa status koneksi WiFi
    if (WiFi.status() == WL_CONNECTED) {
      // Membuat instance HTTPClient untuk mengirimkan
permintaan HTTP
      HTTPClient http;
      // Membentuk URL server untuk mengirim data
      String serverPath = String("http://") + host +
"/receive_mq135_data";

      // Menampilkan URL server yang akan dikirimi permintaan
      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      // Memulai koneksi ke server dengan URL yang telah
dibentuk
      http.begin(wifiClient, serverPath);
      // Menambahkan header "Content-Type" dengan nilai
"application/json"
      http.addHeader("Content-Type", "application/json");

      // Membentuk data permintaan dalam format JSON
      String httpRequestData = "{\"co\":" + String(co, 1) + "}";
      // Menampilkan data permintaan yang akan dikirim
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      // Mengirimkan permintaan POST dengan data JSON
      int httpResponseCode = http.POST(httpRequestData);

      // Memeriksa kode respons dari server
      if (httpResponseCode > 0) {
        // Jika berhasil, menampilkan kode respons dan respons
dari server
        String response = http.getString();
        Serial.println(httpResponseCode);
        Serial.println(response);
      } else {
        // Jika gagal, menampilkan pesan kesalahan
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }

      // Mengakhiri koneksi HTTP
      http.end();
    } else {
```

```
      // Jika tidak terhubung ke WiFi, menampilkan pesan
kesalahan
      Serial.println("Error in WiFi connection");
    }
  }
}

void sendFlameData(bool flame_detected) {
  // Variabel statis untuk menyimpan status terakhir deteksi api
  static bool lastFlameDetected = false;

  // Memeriksa apakah api terdeteksi dan status deteksi api
berubah dari status terakhir
  if (flame_detected && flame_detected != lastFlameDetected) {
    // Memperbarui status terakhir deteksi api
    lastFlameDetected = flame_detected;

    // Memeriksa status koneksi WiFi
    if (WiFi.status() == WL_CONNECTED) {
      // Membuat instance HTTPClient untuk mengirimkan
permintaan HTTP
      HTTPClient http;
      // Membentuk URL server untuk mengirim data
      String serverPath = String("http://") + host +
"/receive_flamesensor_data";

      // Menampilkan URL server yang akan dikirimi permintaan
      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      // Memulai koneksi ke server dengan URL yang telah
dibentuk
      http.begin(wifiClient, serverPath);
      // Menambahkan header "Content-Type" dengan nilai
"application/json"
      http.addHeader("Content-Type", "application/json");

      // Membentuk data permintaan dalam format JSON
      String httpRequestData = "{\"flame_detected\":" +
String(flame_detected ? "true" : "false") + "}";
      // Menampilkan data permintaan yang akan dikirim
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      // Mengirimkan permintaan POST dengan data JSON
      int httpResponseCode = http.POST(httpRequestData);

      // Memeriksa kode respons dari server
      if (httpResponseCode > 0) {
        // Jika berhasil, menampilkan kode respons dan respons
dari server
```

```
        String response = http.getString();
        Serial.print("HTTP Response code: ");
        Serial.println(httpResponseCode);
        Serial.print("Response: ");
        Serial.println(response);
      } else {
        // Jika gagal, menampilkan pesan kesalahan
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }

      // Mengakhiri koneksi HTTP
      http.end();
    } else {
      // Jika tidak terhubung ke WiFi, menampilkan pesan
kesalahan
      Serial.println("Error in WiFi connection");
    }
  } else if (!flame_detected) {
    // Jika api tidak terdeteksi, mengatur status terakhir
deteksi api menjadi false
    lastFlameDetected = false;
  }
}

void sendFanStatus(bool fanStatus) {
  static bool lastFanStatus = false;
  if (fanStatus != lastFanStatus) {
    lastFanStatus = fanStatus;

    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      String serverPath = String("http://") + host +
"/receive_fan_status";

      Serial.print("Sending POST request to ");
      Serial.println(serverPath);

      http.begin(wifiClient, serverPath);
      http.addHeader("Content-Type", "application/json");

      String httpRequestData = "{\"fan_status\":\"" +
String(fanStatus ? "Menyala" : "Mati") + "\"}";
      Serial.print("Request data: ");
      Serial.println(httpRequestData);

      int httpResponseCode = http.POST(httpRequestData);

      if (httpResponseCode > 0) {
        String response = http.getString();
        Serial.println(httpResponseCode);
```

```
        Serial.println(response);
      } else {
        Serial.print("Error on sending POST: ");
        Serial.println(httpResponseCode);
      }

      http.end();
    } else {
      Serial.println("Error in WiFi connection");
    }
  }
}

// Fungsi untuk mengirim pesan ke Telegram
void sendTelegramMessage(String message) {
  WiFiClientSecure client;
  client.setInsecure(); // Mengabaikan sertifikat SSL untuk
sementara

  if (!client.connect("api.telegram.org", 443)) {
    Serial.println("Connection to Telegram API failed");
    return;
  }

  String url = "/bot" + botToken + "/sendMessage?chat_id=" +
chatID + "&text=" + urlencode(message);

  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: api.telegram.org\r\n" +
               "User-Agent: ESP8266\r\n" +
               "Connection: close\r\n\r\n");

  while (client.connected()) {
    String line = client.readStringUntil('\n');
    if (line == "\r") {
      break;
    }
  }

  String line = client.readStringUntil('\n');
  if (line.startsWith("{")) {
    Serial.println("Message sent successfully");
  } else {
    Serial.println("Error sending message: " + line);
  }
}

// Fungsi untuk URL-encode pesan
String urlencode(String str) {
  String encodedString = "";
  char c;
```

```
  char code0;
  char code1;
  char code2;
  for (int i = 0; i < str.length(); i++) {
    c = str.charAt(i);
    if (c == ' ') {
      encodedString += '+';
    } else if (isalnum(c)) {
      encodedString += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) {
        code1 = (c & 0xf) - 10 + 'A';
      }
      c = (c >> 4) & 0xf;
      code0 = c + '0';
      if (c > 9) {
        code0 = c - 10 + 'A';
      }
      code2 = '\0';
      encodedString += '%';
      encodedString += code0;
      encodedString += code1;
    }
    yield();
  }
  return encodedString;
}

void setup()
{
  Serial.begin(115200);
  pinMode(flamePin, INPUT);

  // Inisialisasi sensor DHT21
  Serial.print("LIBRARY VERSION: ");
  Serial.println(DHTNEW_LIB_VERSION);
  Serial.println();

  // Inisialisasi sensor BH1750
  Wire.begin();
  if (!bh1750.begin())
  {
    Serial.println("BH1750 tidak terdeteksi. Periksa sambungan
dan alamat I2C.");
    while (1)
      ;
  }
  pinMode(relayPin_Light, OUTPUT);
  digitalWrite(relayPin_Light, LOW); // Inisialisasi relay High
Triger untuk Lampu
```

```
  // Inisialisasi Relay Kipas
  pinMode(pirPin, INPUT);
  pinMode(relayPinPIR, OUTPUT);
  digitalWrite(relayPinPIR, LOW); // Inisialisasi relay High
Triger untuk Kipas

  // Inisialisasi Relay Exhoust Fan
  pinMode(relayPin_CO, OUTPUT);
  digitalWrite(relayPin_CO, LOW); // Inisialisasi relay High
Triger untuk Exhoust Fan

  // Inisialisasi Relay Buzzer
  pinMode(relayPin_buzzer, OUTPUT);
  digitalWrite(relayPin_buzzer, LOW); // Inisialisasi relay High
Triger untuk Buzzer

  // Koneksi ke WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Kalibrasi sensor MQ-135
  MQ135.setRegressionMethod(1);
  MQ135.setA(CO_A);
  MQ135.setB(CO_B);
  MQ135.init();

  Serial.print("Calibrating MQ-135, please wait.");
  float calcR0 = 0;
  for (int i = 1; i <= 10; i++)
  {
    MQ135.update();
    calcR0 += MQ135.calibrate(RatioMQ135CleanAir);
    Serial.print(".");
  }
  MQ135.setR0(calcR0 / 10);
  Serial.println(" done!");
  if (isinf(calcR0))
  {
    Serial.println("Warning: Connection issue found, R0 is
infinite (Open circuit detected), please check your wiring and
supply");
    while (1)
      ;
```

```
  }
  if (calcR0 == 0)
  {
    Serial.println("Warning: Connection issue found, R0 is zero
(Analog pin with short circuit to ground), please check your
wiring and supply");
    while (1)
      ;
  }

}

void loop()
{
  static unsigned long lastPingTime = 0;
  static unsigned long lastReadTime = 0;
  unsigned long currentTime = millis();

  if (currentTime - lastReadTime >= 1000){
    lastReadTime = currentTime;

    // Baca Data Sensor DHT21
    mySensor.read();
    float temperature = mySensor.getTemperature();
    float humidity = mySensor.getHumidity();
    sendDHTData(temperature, humidity);

    //Baca Data Sensor BH1750
     float lightIntensity = bh1750.light();

    if (lightIntensity < threshold_Light)
    {
      digitalWrite(relayPin_Light, HIGH); // Hidupkan relay
(lampu menyala)
    }
    else
    {
      digitalWrite(relayPin_Light, LOW); // Matikan relay (lampu
mati)
    }
    sendBH1750Data(lightIntensity);

    //Baca Data Sensor PIR
    bool motion_detected = digitalRead(pirPin) == HIGH;
    if (motion_detected)
    {
      lastMotionTime = currentTime;
      digitalWrite(relayPinPIR, HIGH);
    }
    else if(currentTime - lastMotionTime >= motionTimeout)
    {
```

```
      digitalWrite(relayPinPIR, LOW);

    }
    sendPIRData(motion_detected);
    sendFanStatus(digitalRead(relayPinPIR) == HIGH);

    //Baca Data Flame Sensor
    bool flame_detected = digitalRead(flamePin) == HIGH;
    if (flame_detected)
    {
      sendTelegramMessage("!!!! Bahaya, Terdapat Api Yang
Terdeteksi");
      digitalWrite(relayPin_buzzer, HIGH);
      delay(5000);
      digitalWrite(relayPin_buzzer, LOW);
    }
    else
    {
      digitalWrite(relayPin_buzzer, LOW);
    }
    sendFlameData(flame_detected);

    MQ135.update();
    float co = MQ135.readSensor();

    if (co > CO_THRESHOLD)
    {
      digitalWrite(relayPin_CO, HIGH); // Aktifkan relay (HIGH-
triggered)
    }
    else
    {
      digitalWrite(relayPin_CO, LOW); // Nonaktifkan relay
(HIGH-triggered)
    }
    sendCOData(co);

  }

}
```

Lampiran 5. Kode Program Web Monitoring *Smart Warehouse*

```
from flask import Flask, jsonify, render_template, redirect,
url_for, request, flash, make_response
from io import BytesIO
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from flask_sqlalchemy import SQLAlchemy
```

```
from reportlab.lib.styles import getSampleStyleSheet,
ParagraphStyle
from reportlab.platypus import SimpleDocTemplate, Paragraph,
Spacer
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user
from datetime import datetime
import pytz
from flask_migrate import Migrate
from datetime import datetime, timedelta

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://adie:password@localhost/smartwarehouse'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
# Pengaturan zona waktu untuk aplikasi Flask
JAKARTA_TZ = pytz.timezone('Asia/Jakarta')

db = SQLAlchemy(app)
migrate = Migrate(app, db)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and user.password == password:
            login_user(user)
            return redirect(url_for('dashboard'))
        else:
            flash('Login Unsuccessful. Please check username and
password', 'danger')
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

@app.route('/')
@login_required
def dashboard():
    return render_template('dashboard.html')

@app.route('/dht21')
@login_required
def dht21():
    return render_template('dht21.html')
```

```
@app.route('/get_dht21_data')
@login_required
def get_dht21_data():
    dht21_data =
Dht21Data.query.order_by(Dht21Data.timestamp.desc()).all()
    data = [{
        'timestamp': d.timestamp.strftime('%Y-%m-%d %H:%M:%S'),
        'temperature': d.temperature,
        'humidity': d.humidity
    } for d in dht21_data]
    return jsonify(data)

@app.route('/receive_dht21_data', methods=['POST'])
def receive_dht21_data():
    if request.method == 'POST':
        data = request.get_json()
        temperature = data['temperature']
        humidity = data['humidity']
        new_data = Dht21Data(temperature=temperature,
humidity=humidity)
        db.session.add(new_data)
        db.session.commit()
        return jsonify({'message': 'Data received'}), 201

@app.route('/mq135')
@login_required
def mq135():
    return render_template('mq135.html')

@app.route('/get_mq135_data')
@login_required
def get_mq135_data():
    mq135_data =
Mq135Data.query.order_by(Mq135Data.timestamp.desc()).all()
    data = [{
        'timestamp': m.timestamp.strftime('%Y-%m-%d %H:%M:%S'),
        'co': m.co
    } for m in mq135_data]
    return jsonify(data)

@app.route('/receive_mq135_data', methods=['POST'])
def receive_mq135_data():
    if request.method == 'POST':
        data = request.get_json()
        co = data['co']
        new_data = Mq135Data(co=co)
        db.session.add(new_data)
        db.session.commit()
        return jsonify({'message': 'Data received'}), 201


@app.route('/bh1750')
@login_required
def bh1750():
    return render_template('bh1750.html')

@app.route('/get_bh1750_data')
@login_required
def get_bh1750_data():
```

```
    bh1750_data =
Bh1750Data.query.order_by(Bh1750Data.timestamp.desc()).all()
    data = [{
        'timestamp': b.timestamp.strftime('%Y-%m-%d %H:%M:%S'),
        'light_intensity': b.light_intensity
    } for b in bh1750_data]
    return jsonify(data)

@app.route('/receive_bh1750_data', methods=['POST'])
def receive_bh1750_data():
    if request.method == 'POST':
        data = request.get_json()
        light_intensity = data['light_intensity']
        new_data = Bh1750Data(light_intensity=light_intensity)
        db.session.add(new_data)
        db.session.commit()
        return jsonify({'message': 'Data received'}), 201


@app.route('/flamesensor')
@login_required
def flamesensor():
    return render_template('flamesensor.html')

@app.route('/get_flamesensor_data')
@login_required
def get_flamesensor_data():
    flame_data =
FlameSensor.query.order_by(FlameSensor.timestamp.desc()).all()
    data = [{
        'timestamp': f.timestamp.strftime('%Y-%m-%d %H:%M:%S'),
        'flame_detected': f.flame_detected
    } for f in flame_data]
    return jsonify(data)

@app.route('/receive_flamesensor_data',methods=['POST'])
def receive_flamesensor_data():
    if request.method == 'POST' :
        data = request.get_json()
        flame_detected = data['flame_detected']
        new_data = FlameSensor(flame_detected=flame_detected)
        db.session.add(new_data)
        db.session.commit()
        return jsonify({'message': 'Data received'}), 201

@app.route('/pirsensor')
@login_required
def pirsensor():
    return render_template('pirsensor.html')

@app.route('/get_pirsensor_data')
@login_required
def get_pirsensor_data():
    pir_data =
PirSensor.query.order_by(PirSensor.timestamp.desc()).all()
    data = [{
        'timestamp': p.timestamp.strftime('%Y-%m-%d %H:%M:%S'),
        'motion_detected': p.motion_detected
    } for p in pir_data]
```

```
    return jsonify(data)

@app.route('/receive_pir_data', methods=['POST'])
def receive_pir_data():
    if request.method == 'POST':
        data = request.get_json()
        motion_detected = data['motion_detected']
        new_data = PirSensor(motion_detected=motion_detected)
        db.session.add(new_data)
        db.session.commit()
        return jsonify({'message': 'Data received'}), 201

@app.route('/lampusagehistory')
@login_required
def lampusagehistory():
    return render_template('lampusagehistory.html')

@app.route('/fanusagehistory')
@login_required
def fanusagehistory():
    return render_template('fanusagehistory.html')

@app.route('/exhousthistory')
@login_required
def exhousthistory():
    return render_template('exhousthistory.html')

@app.route('/report')
@login_required
def report():
    return render_template('report.html')

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True,
nullable=False)
    password = db.Column(db.String(150), nullable=False)

with app.app_context():
    db.create_all()
    if not User.query.first():
        user = User(username='admin', password='admin')
        db.session.add(user)
        db.session.commit()


class Dht21Data(db.Model):
    __tablename__ = 'dht21_data'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))
    temperature = db.Column(db.Float, nullable=False)
    humidity = db.Column(db.Float, nullable=False)

class Mq135Data(db.Model):
    __tablename__ = 'mq135_data'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))
```

```
    co = db.Column(db.Float, nullable=False)

class FlameSensor(db.Model):
    __tablename__ = 'flamesensor'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))
    flame_detected = db.Column(db.Boolean, nullable=False)

class PirSensor(db.Model):
    __tablename__ = 'pirsensor'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))
    motion_detected = db.Column(db.Boolean, nullable=False)

class Bh1750Data(db.Model):
    __tablename__ = 'bh1750_data'
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))
    light_intensity = db.Column(db.Float, nullable=False)

class Report(db.Model):
    __tablename__ = 'report'
    id = db.Column(db.Integer, primary_key=True)
    reportname = db.Column(db.String(128), nullable=False)
    max_humidity = db.Column(db.Float, nullable=False,
default=0.0)
    max_temperature = db.Column(db.Float, nullable=False,
default=0.0)
    max_co = db.Column(db.Float, nullable=False, default=0.0)
    min_temperature = db.Column(db.Float, nullable=False,
default=0.0)
    min_humidity = db.Column(db.Float, nullable=False,
default=0.0)
    min_co = db.Column(db.Float, nullable=False, default=0.0)
    avg_temperature = db.Column(db.Float, nullable=False,
default=0.0)
    avg_humidity = db.Column(db.Float, nullable=False,
default=0.0)
    avg_co = db.Column(db.Float, nullable=False, default=0.0)
    flame_detected = db.Column(db.Boolean, nullable=False,
default=False)
    start_date = db.Column(db.Date, nullable=False)
    end_date = db.Column(db.Date, nullable=False)
    created_at = db.Column(db.TIMESTAMP, nullable=False,
default=lambda: datetime.now(JAKARTA_TZ))

def calculate_stats(data, attribute):
    values = [getattr(d, attribute) for d in data if getattr(d,
attribute) is not None]

    if not values:
        return {'max': 0.0, 'min': 0.0, 'avg': 0.0}

    max_value = max(values)
    min_value = min(values)
    avg_value = sum(values) / len(values) if values else 0.0
```

```
    return {'max': max_value, 'min': min_value, 'avg':
avg_value}

@app.route('/get_sensor_data')
@login_required
def get_sensor_data():
    dht21_data =
Dht21Data.query.order_by(Dht21Data.timestamp.desc()).limit(10).a
ll()
    mq135_data =
Mq135Data.query.order_by(Mq135Data.timestamp.desc()).limit(10).a
ll()
    flame_data =
FlameSensor.query.order_by(FlameSensor.timestamp.desc()).limit(1
0).all()
    bh1750_data =
Bh1750Data.query.order_by(Bh1750Data.timestamp.desc()).limit(10)
.all()
    pir_data =
PirSensor.query.order_by(PirSensor.timestamp.desc()).limit(10).a
ll()

    dht21_list = [{'timestamp': d.timestamp.timestamp(),
'temperature': d.temperature, 'humidity': d.humidity} for d in
dht21_data]
    mq135_list = [{'timestamp': m.timestamp.timestamp(), 'co':
m.co} for m in mq135_data]
    flame_list = [{'timestamp': f.timestamp.timestamp(),
'flame_detected': f.flame_detected} for f in flame_data]
    bh1750_list = [{'timestamp': b.timestamp.timestamp(),
'light_intensity': b.light_intensity} for b in bh1750_data]
    pir_list = [{'timestamp': p.timestamp.timestamp(),
'motion_detected': p.motion_detected} for p in pir_data]

    return jsonify({'dht21_data': dht21_list, 'mq135_data':
mq135_list, 'flame_data': flame_list, 'bh1750_data':
bh1750_list, 'pir_data' : pir_list}
    )

@app.route('/generate_report', methods=['POST'])
@login_required
def generate_report():
    data = request.get_json()
    start_date = data.get('start_date')
    end_date = data.get('end_date')

    if start_date and end_date:
        start_date = datetime.strptime(start_date, '%Y-%m-%d')
        end_date = datetime.strptime(end_date, '%Y-%m-%d')

        dht21_data = Dht21Data.query.filter(Dht21Data.timestamp
>= start_date, Dht21Data.timestamp <= end_date).all()
        mq135_data = Mq135Data.query.filter(Mq135Data.timestamp
>= start_date, Mq135Data.timestamp <= end_date).all()
        flame_data =
FlameSensor.query.filter(FlameSensor.timestamp >= start_date,
FlameSensor.timestamp <= end_date).all()
```

```
        pir_data = PirSensor.query.filter(PirSensor.timestamp >=
start_date, PirSensor.timestamp <= end_date).all()
        bh1750_data =
Bh1750Data.query.filter(Bh1750Data.timestamp >= start_date,
Bh1750Data.timestamp <= end_date).all()
    else:
        return jsonify([]), 400

    stats = {
        'temperature': calculate_stats(dht21_data,
'temperature'),
        'humidity': calculate_stats(dht21_data, 'humidity'),
        'co': calculate_stats(mq135_data, 'co'),
        'light_intensity': calculate_stats(bh1750_data,
'light_intensity'),
        'motion_detected': calculate_stats(pir_data,
'motion_detected'),
        'flame_detected': calculate_stats(flame_data,
'flame_detected')
    }

    start_date_str = start_date.strftime('%Y-%m-%d')
    end_date_str = end_date.strftime('%Y-%m-%d')
    reportname = f"Sensor Report {start_date_str} to
{end_date_str}"

    report = Report(
        reportname=reportname,
        max_humidity=stats['humidity']['max'],
        max_temperature=stats['temperature']['max'],
        max_co=stats['co']['max'],
        min_temperature=stats['temperature']['min'],
        min_humidity=stats['humidity']['min'],
        min_co=stats['co']['min'],
        avg_temperature=stats['temperature']['avg'],
        avg_humidity=stats['humidity']['avg'],
        avg_co=stats['co']['avg'],
        flame_detected=stats['flame_detected']['max'],
        start_date=start_date,
        end_date=end_date
    )

    db.session.add(report)
    db.session.commit()
    return jsonify(report.id)

@app.route('/get_all_report')
@login_required
def get_all_report():
    reports =
Report.query.order_by(Report.created_at.desc()).all()
    data = [{
        'id': r.id,
        'reportname': r.reportname,
        'max_humidity': r.max_humidity,
        'max_temperature': r.max_temperature,
        'max_co': r.max_co,
        'min_temperature': r.min_temperature,
        'min_humidity': r.min_humidity,
```

```
        'min_co': r.min_co,
        'avg_temperature': r.avg_temperature,
        'avg_humidity': r.avg_humidity,
        'avg_co': r.avg_co,
        'flame_detected': r.flame_detected,
        'start_date': r.start_date.strftime('%Y-%m-%d'),
        'end_date': r.end_date.strftime('%Y-%m-%d'),
        'created_at': r.created_at.strftime('%Y-%m-%d %H:%M:%S')
    } for r in reports]
    return jsonify(data)

@app.route('/report/<int:id>', methods=['GET'])
def get_report(id):
    report = Report.query.get_or_404(id)
    return jsonify({
        'reportname': report.reportname,
        'max_humidity': report.max_humidity,
        'max_temperature': report.max_temperature,
        'max_co': report.max_co,
        'min_temperature': report.min_temperature,
        'min_humidity': report.min_humidity,
        'min_co': report.min_co,
        'avg_temperature': report.avg_temperature,
        'avg_humidity': report.avg_humidity,
        'avg_co': report.avg_co,
        'flame_detected': report.flame_detected,
        'start_date': report.start_date.strftime('%Y-%m-%d
%H:%M:%S'),
        'end_date': report.end_date.strftime('%Y-%m-%d
%H:%M:%S'),
        'created_at': report.created_at.strftime('%Y-%m-%d
%H:%M:%S')
    })

@app.route('/report/<int:id>', methods=['DELETE'])
def delete_report(id):
    report = Report.query.get_or_404(id)
    db.session.delete(report)
    db.session.commit()
    return '', 204


@app.route('/download_report_pdf/<int:id>', methods=['GET'])
def download_report_pdf(id):
    report = Report.query.get_or_404(id)
    buffer = BytesIO()
    pdf = canvas.Canvas(buffer, pagesize=letter)

    styles = getSampleStyleSheet()
    title_style = styles['Title']
    heading_style = styles['Heading2']
    body_style = styles['BodyText']

    # Title
    pdf.setFont("Helvetica-Bold", 16)
    pdf.drawString(50, 750, 'Smart Warehouse Report')

    # Add data to PDF
    pdf.setFont("Helvetica", 12)
```

```
    y_position = 720

    pdf.drawString(50, y_position, 'Report Name:')
    pdf.drawString(150, y_position, report.reportname)
    y_position -= 20

    pdf.drawString(50, y_position, 'Max Humidity:')
    pdf.drawString(150, y_position, str(report.max_humidity))
    y_position -= 20

    pdf.drawString(50, y_position, 'Max Temperature:')
    pdf.drawString(150, y_position, str(report.max_temperature))
    y_position -= 20

    pdf.drawString(50, y_position, 'Max CO:')
    pdf.drawString(150, y_position, str(report.max_co))
    y_position -= 20

    pdf.drawString(50, y_position, 'Min Temperature:')
    pdf.drawString(150, y_position, str(report.min_temperature))
    y_position -= 20

    pdf.drawString(50, y_position, 'Min Humidity:')
    pdf.drawString(150, y_position, str(report.min_humidity))
    y_position -= 20

    pdf.drawString(50, y_position, 'Min CO:')
    pdf.drawString(150, y_position, str(report.min_co))
    y_position -= 20


    pdf.drawString(50, y_position, 'Avg Temperature:')
    pdf.drawString(150, y_position, str(report.avg_temperature))
    y_position -= 20

    pdf.drawString(50, y_position, 'Avg Humidity:')
    pdf.drawString(150, y_position, str(report.avg_humidity))
    y_position -= 20

    pdf.drawString(50, y_position, 'Avg CO:')
    pdf.drawString(150, y_position, str(report.avg_co))
    y_position -= 20


    pdf.drawString(50, y_position, 'Flame Detected:')
    pdf.drawString(150, y_position, 'Yes' if
report.flame_detected else 'No')
    y_position -= 20

    pdf.drawString(50, y_position, 'Start Date:')
    pdf.drawString(150, y_position,
report.start_date.strftime('%Y-%m-%d %H:%M:%S'))
    y_position -= 20

    pdf.drawString(50, y_position, 'End Date:')
    pdf.drawString(150, y_position,
report.end_date.strftime('%Y-%m-%d %H:%M:%S'))
    y_position -= 20
```

```
    pdf.drawString(50, y_position, 'Created At:')
    pdf.drawString(150, y_position,
report.created_at.strftime('%Y-%m-%d %H:%M:%S'))
    y_position -= 20
    pdf.save()
    pdf.showPage()
    buffer.seek(0)

    response = make_response(buffer.getvalue())
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = f'attachment;
filename=report_{id}.pdf'
    return response

last_ping_time = None

@app.route('/ping', methods=['GET'])
def ping():
    global last_ping_time
    last_ping_time = datetime.now()
    return "Ping received", 200

@app.route('/status', methods=['GET'])
def status():
    global last_ping_time
    status = "disconnected"
    if last_ping_time:
        if datetime.now() - last_ping_time <
timedelta(seconds=10):
            status = "connected"
    return jsonify(status=status)

if __name__ == '__main__':
    app.run(debug=True,host="0.0.0.0")
```