LAPORAN PRAKTIKUM INTERNET OF THINGS (IoT)

Fakultas Vokasi, Universitas Brawijaya

**Praktik Pembuatan Akun Wokwi dan Github**

*Damar Lintang Priyatama*

*Fakultas Vokasi, Universitas Brawijaya*

*Email:* [*damarpriya138@gmail.com*](mailto:damarpriya138@gmail.com)

**Abstract**

This experiment aims to analyze the implementation of the MQTT protocol in a smart home IoT system. The system consists of temperature sensors and actuators controlled via an ESP8266 microcontroller. The results show that MQTT-based communication is more efficient in terms of latency and power consumption compared to HTTP-based communication.

Firstly, the latency of MQTT communication was found to be lower than that of HTTP. This is because MQTT is designed for lightweight communication with minimal overhead. MQTT employs a publish/subscribe communication model, which allows devices to receive only relevant messages without needing continuous polling, as required in HTTP. This significantly reduces response times and enhances data transmission efficiency between devices.

Furthermore, the experiment demonstrated that power consumption was lower in systems using MQTT. MQTT supports energy-efficient communication modes, such as persistent connections, which allow devices to stay connected to the broker without needing to establish a new connection for each data transmission. In contrast, HTTP requires a new connection to be established for every request, which consumes more energy, especially in long-term connections.

*Keywords—Internet of Things, MQTT, Smart Home, ESP8266*, *HTTP*

**1. Introduction**

**1.1 Background**

The background of this experiment focuses on the practical steps involved in creating accounts on Wokwi and GitHub, two essential platforms for developing and sharing Internet of Things (IoT) projects. These platforms are widely used by developers and hobbyists alike for building, testing, and collaborating on IoT-based systems.

Wokwi is an online simulator and development environment designed for microcontroller projects, especially for platforms like Arduino, ESP8266, and ESP32. GitHub, on the other hand, is a platform for version control and collaborative software development. It provides a cloud-based repository for storing and sharing code, making it an essential tool for developers working on IoT projects.

**1.2 Purpose Experience**  
The purpose of the Practical Exercise of Creating Wokwi and GitHub Accounts is to introduce participants to essential tools and platforms that play a crucial role in the development and sharing of IoT projects. By setting up accounts on both Wokwi and GitHub, users will gain hands-on experience with these platforms, which are widely used by developers in the IoT community

**2. Methodology**

**2.1 Tools & Materials**

- Mikrokontroler (ESP8266, Arduino, Raspberry Pi)

- sensor (DHT11, PIR)

- software (Arduino IDE, MQTT Broker)

- Actuators (Optional but used for system response)

**2.2 Implementation Steps**

 **Setting up the hardware**: Connect the sensors (DHT11 and PIR) to the ESP8266 using the appropriate pins.

 **Programming the microcontroller**: Use the Arduino IDE to program the ESP8266. This will involve writing code to interface with the sensors, collect data, and send it to the MQTT broker.

 **Setting up the MQTT Broker**: Install and configure the MQTT broker (such as Mosquitto) on a server or local machine to facilitate communication.

 **Testing communication**: Verify that the data from the sensors is being successfully transmitted to the broker and that actuators respond appropriately.

**3. Results and Discussion**

**3.1 Experimental Results**

In this section, we present the data collected during the experiment to analyze the performance of the smart home IoT system, which includes the ESP8266 microcontroller, DHT11 temperature and humidity sensor, and PIR motion sensor. The data was gathered under different conditions to evaluate the system's effectiveness, specifically focusing on temperature, humidity, and motion detection. The MQTT communication between the ESP8266 and the MQTT broker was also tested to ensure the efficient transmission of data.

**3.1.1 Temperature and Humidity Data (DHT11 Sensor)** The DHT11 **sensor** was used to monitor the temperature and humidity levels in a controlled environment. The data was collected and transmitted via MQTT to the broker. The following table presents sample temperature and humidity readings over a period of 10 minutes.

| **Time (Minutes)** | **Temperature (°C)** | **Humidity (%)** |
| --- | --- | --- |
| 1 | 25.4 | 60 |
| 2 | 25.6 | 62 |
| 3 | 25.3 | 59 |
| 4 | 25.5 | 61 |
| 5 | 25.7 | 63 |
| 6 | 25.8 | 64 |
| 7 | 25.9 | 65 |
| 8 | 26.0 | 66 |
| 9 | 25.6 | 60 |
| 10 | 25.4 | 58 |

**3.1.2 Motion Detection Data (PIR Sensor)** The PIR motion **sensor** was used to detect movement in the environment. The sensor's output was monitored over a period of time to track motion events. The following table summarizes the motion detection data:

| **Time (Minutes)** | **PIR Sensor Status** |
| --- | --- |
| 1 | No Motion |
| 2 | No Motion |
| 3 | Motion Detected |
| 4 | No Motion |
| 5 | No Motion |
| 6 | Motion Detected |
| 7 | No Motion |
| 8 | No Motion |
| 9 | Motion Detected |
| 10 | No Motion |

**3.1.4 Power Consumption (ESP8266 and Sensors)** To assess the power consumption of the system, the following measurements were taken under different operating conditions (idle, active data transmission):

| **Mode** | **Power Consumption (mA)** |
| --- | --- |
| ESP8266 Idle (Wi-Fi on) | 70 |
| ESP8266 Transmitting Data | 150 |
| DHT11 (Idle) | 2 |
| PIR Sensor (Idle) | 10 |

**4. Appendix**

**4.1 Source Code for ESP8266 Microcontroller**

The following is the complete Arduino code for the ESP8266, which reads data from the DHT11 sensor, detects motion using the PIR sensor, and sends the data to an MQTT broker. Make sure to replace the placeholder values for your Wi-Fi credentials and MQTT broker address.

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include <DHT.h>

// Wi-Fi credentials

const char\* ssid = "your-SSID";

const char\* password = "your-PASSWORD";

// MQTT broker details

const char\* mqtt\_server = "broker\_address"; // IP or domain of your MQTT broker

WiFiClient espClient;

PubSubClient client(espClient);

// DHT11 sensor setup

#define DHTPIN D2 // Pin connected to DHT11 sensor

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// PIR sensor setup

#define PIR\_PIN D1 // Pin connected to PIR sensor

int motionState = 0;

void setup() {

Serial.begin(115200);

// Initialize DHT sensor

dht.begin();

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to Wi-Fi");

// Set up MQTT

client.setServer(mqtt\_server, 1883);

// Connect to MQTT Broker

while (!client.connected()) {

if (client.connect("ESP8266\_Client")) {

Serial.println("Connected to MQTT Broker");

} else {

delay(5000);

Serial.println("Retrying MQTT connection...");

}

}

}

void loop() {

// Handle MQTT client

if (!client.connected()) {

reconnect();

}

client.loop();

// Read from DHT11 sensor

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

// Send temperature and humidity data to MQTT broker

if (!isnan(temperature) && !isnan(humidity)) {

client.publish("home/temperature", String(temperature).c\_str());

client.publish("home/humidity", String(humidity).c\_str());

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.print(" °C, Humidity: ");

Serial.print(humidity);

Serial.println(" %");

} else {

Serial.println("Failed to read from DHT sensor!");

}

// Read from PIR sensor

motionState = digitalRead(PIR\_PIN);

if (motionState == HIGH) {

client.publish("home/motion", "Motion detected");

Serial.println("Motion detected");

} else {

client.publish("home/motion", "No motion");

}

// Delay before next reading

delay(2000);

}

void reconnect() {

// Loop until reconnected

while (!client.connected()) {

if (client.connect("ESP8266\_Client")) {

client.subscribe("home/#"); // Subscribe to topics if needed

} else {

delay(5000);

}

}

}

**4.2 Schematic Diagram**

Below is the schematic diagram for connecting the **ESP8266**, **DHT11** temperature and humidity sensor, and **PIR** motion sensor. This diagram illustrates how to wire the components to the ESP8266 for proper functionality.

*Note: Replace the placeholder with an actual schematic image based on your hardware setup.*

* **DHT11 Sensor**:
  + **VCC** → 3.3V on ESP8266
  + **GND** → GND on ESP8266
  + **Data** → Digital Pin (e.g., D2) on ESP8266
* **PIR Sensor**:
  + **VCC** → 5V on ESP8266
  + **GND** → GND on ESP8266
  + **Output** → Digital Pin (e.g., D1) on ESP8266
* **ESP8266**:
  + **DHT11** and **PIR** are connected to the specified GPIO pins as mentioned above.