INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Internet Of Things

TP 2: Development Devices for the Internet of Things

Group members:

Student Number	Name
a57096	Dibya Raj Khatri
a43323	Bikal Bista

Worksheet 2

Objectives :
In this course, a microcontroller device called ESP8266 or ESP32 will be used to digitize the different types of data, such as data related to weather conditions,temperature, humidity, lighting, among others.

Introduction:

The Internet of Things (IoT) enables the digitization of real-world data using microcontroller-based development boards. In this course, we use the ESP8266 and ESP32

microcontroller boards to collect and transmit sensor data such as temperature, humidity, and lighting. These devices communicate using the MQTT protocol, a lightweight messaging protocol optimized for IoT applications. This report outlines the setup process for developing applications using these boards, including the installation of the Arduino IDE, required libraries, and MQTT communication.

1. Installing the Arduino IDE

To program the ESP8266 or ESP32 development boards, the Arduino Integrated Development Environment (IDE) must be installed. The process involves the following steps:

Installation Steps

Download and Install the Arduino IDE

- Download the latest version of Arduino IDE from Arduino's official website.
- Follow the installation instructions for Windows, macOS, or Linux.

Install Board Drivers

- For ESP32: Follow the tutorial here.
- For ESP8266: Follow the tutorial here (link truncated in original worksheet).

Add ESP32 or ESP8266 Board to Arduino IDE

- Open Arduino IDE and navigate to **File** → **Preferences**.
- In the "Additional Board Manager URLs" field, add:
  - ESP8266: `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
  - ESP32: `https://dl.espressif.com/dl/package_esp32_index.json`
- Go to **Tools** → **Board** → **Boards Manager**, search for "ESP8266" or "ESP32" and install the respective package.

Install MQTT Library

- Open Arduino IDE and go to **Sketch** → **Include Library** → **Manage Libraries**.
- Search for and install the **PubSubClient** library to enable MQTT communication.

Configuring MQTT Communication

To establish MQTT communication, we will use a public broker such as:

- **broker.hivemq.com**
- **broker.emqx.io**

Connecting to an MQTT Broker

1. Connect the ESP8266/ESP32 to Wi-Fi.
2. Define MQTT server parameters and client ID in the Arduino sketch.
3. Publish and subscribe to topics for data exchange.

2.

2. Uploading Example Codes to the Development Board

To test your ESP development board, you need to upload the following example codes:

- **Blink Example:**
    - Open **Arduino IDE**
    - Navigate to **File > Examples > ESPXX > Blink**
    - Upload this code to your ESP8266 or ESP32.
    - It will turn an LED ON and OFF at regular intervals.
- **BlinkWithoutDelay Example:**
    - Open **Arduino IDE**
    - Navigate to **File > Examples > ESPXX > BlinkWithoutDelay**
    - Upload this code to your ESP8266 or ESP32.
    - It achieves the same blinking effect **without using `delay()`**, allowing other tasks to run simultaneously.

3.Difference Between Blink and BlinkWithoutDelay Example Codes

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the
voltage level)
  delay(1000);                      // wait for a second
```

```
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(1000);                      // wait for a second
}
```

```
const int ledPin = LED_BUILTIN;  // the number of the LED pin

// Variables will change:
int ledState = LOW;  // ledState used to set the LED

// Generally, you should use "unsigned long" for variables that hold
time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;  // will store last time LED was
updated

// constants won't change:
const long interval = 1000;  // interval at which to blink
(milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the
time.

  // check to see if it's time to blink the LED; that is, if the
difference
  // between the current time and last time you blinked the LED is
bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
```

```
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
}
```

Blink Example Code

The Blink example uses the `delay()` function to control the LED state. The code turns the LED on, waits for a set duration, then turns it off and waits again. This process repeats continuously in the `loop()` function.

Code Logic:

1. The LED is turned on using `digitalWrite(LED_BUILTIN, HIGH);`
2. The program waits using `delay(1000);` (1000 milliseconds = 1 second)
3. The LED is turned off using `digitalWrite(LED_BUILTIN, LOW);`
4. Another `delay(1000);` pauses execution before repeating the cycle.

Drawbacks of Using `delay()`

- `delay()` blocks further code execution, preventing the microcontroller from performing other tasks while waiting.
- It is not suitable for applications requiring multitasking, such as sensor readings or handling network requests.

BlinkWithoutDelay Example Code

The BlinkWithoutDelay example eliminates the use of `delay()`, using the `millis()` function instead. This allows the LED to blink without blocking other code execution.

Code Logic:

1. The program records the current time using `millis()`.
2. It checks if a predefined interval has elapsed (e.g., 1000ms).
3. If the interval has passed, the LED state is toggled.
4. The recorded time is updated to keep the timing accurate.

Advantages of `millis()` Over `delay()`

- Non-blocking execution: The program can perform other tasks while managing LED blinking.
- Multitasking support: Ideal for IoT applications that require simultaneous tasks like sensor readings, network communication (MQTT), or user interactions**.**

4. Explanation of `void setup()` and `void loop()`

`void setup()` Function

- Runs **once** when the ESP starts.
- Used to **initialize settings**, such as:
    - Configuring pin modes (`pinMode(LED, OUTPUT);`)
    - Connecting to Wi-Fi (`WiFi.begin(ssid, password);`)
    - Setting up communication (`Serial.begin(115200);`)

`void loop()` Function

- Runs **continuously** after `setup()`.
- Contains the **main logic** of the program.
- In `Blink`, it toggles the LED using `delay()`.
- In `BlinkWithoutDelay`, it checks time using `millis()` and toggles the LED without blocking execution.

7.The detailed explanation of the three functions/instructions:

1. `void callback()`

This function is executed whenever the ESP receives a message from an MQTT topic it is subscribed to.

Functionality:

- It prints the topic and message received to the Serial Monitor.
- It converts the incoming payload (binary data) into a `String` for easier processing.
- It attempts to deserialize the JSON data from the received message.
- If deserialization fails, it prints an error message and exits the function.
- If the received topic is `"IPB/IoT/Lab/Presence"`, it checks the `"Detection"` field in the JSON data:
    - If `"Detection"` is `1`, it **turns ON the LED**.
    - Otherwise, it **turns OFF the LED**.

Key Role:

The `callback()` function ensures that the ESP reacts to incoming MQTT messages appropriately, such as controlling the LED based on the presence detection message.

---

2. `void reconnect()`

This function ensures that the ESP device stays connected to the MQTT broker.

Functionality:

- It continuously attempts to reconnect to the MQTT broker if the connection is lost.
- It generates a **random client ID** to ensure unique identification when reconnecting.
- If the connection is successful:
    - It prints `"connected to the Broker"`.
    - It subscribes to the topic `"IPB/IoT/Lab/Presence"`, allowing the ESP to receive messages.
- If the connection fails:
    - It prints the error code.
    - It waits `5 seconds` before trying again.

Key Role:

The `reconnect()` function ensures the ESP maintains a stable connection to the MQTT broker, automatically reconnecting when disconnected.

---

3. `if (now - lastTime > 2000)`

This condition ensures that a specific action (publishing a message) is executed every **2000 milliseconds (2 seconds)**.

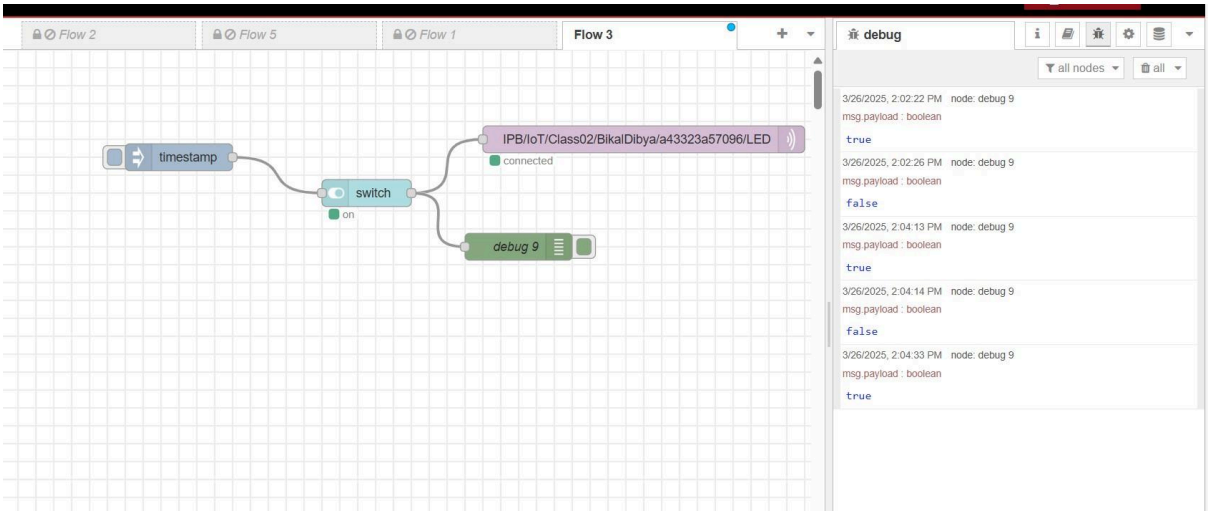
Functionality:

- `millis()` returns the number of milliseconds since the ESP started running.
- `lastTime` stores the last time the message was published.
- The condition `now - lastTime > 2000` ensures that at least **2 seconds** have passed before executing the next action.
- If true:
    - It updates `lastTime` to the current time.
    - It increases `value`.
    - It formats a message (`hello world #value`) and publishes it to `"IPB/IoT/Aula02/AlunoX"`.

Key Role:

This condition acts as a **timer** to prevent excessive MQTT message publishing, ensuring that messages are sent at controlled intervals (every 2 seconds).

8.Create a new dashboard in Node-RED containing a “switch” button that will send commands to turn on/off the ESP LED. For this purpose, use the MQTT protocol where Node-RED will publish a topic.




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

#define LED 33

const char* mqtt_server = "broker.emqx.io";
const char* ssid = "agents";
const char* password =
"QgC9O8VucAByqvVu5Rruv1zdpqM66cd23KG4ElV7vZiJND580bzYvaHqz5k07G2";

WiFiClient espClient;
PubSubClient client(espClient);

long lastTime = 0;
const int Msg_Size = 50;
char msg[Msg_Size];
int value = 0;

void setup_wifi() {
  Serial.println("Starting Setup Wifi");
  delay(10);
  Serial.println();
  Serial.print("Connecting to Wifi ");
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  randomSeed(micros());
  Serial.println("");
  Serial.println("WiFi connected");
```

```
}

void callback(String topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  String strPayload = "";
  for (int i = 0; i < length; i++) {
    strPayload = strPayload + (char)payload[i];
  }

  Serial.println(strPayload);

  // Allocate the JSON document
  JsonDocument doc;
  DeserializationError error = deserializeJson(doc, strPayload);

  if (error) {
    Serial.print("JSON deserialization failed: ");
    Serial.println(error.f_str());
    return;
  }

  // Check the correct topic for LED control
  if (topic == "IPB/IoT/Class02/BikalDibya/a43323a57096/LED") {
    if (doc["LED"] == 1) {
      digitalWrite(LED, HIGH);  // Turn on the LED
    } else {
      digitalWrite(LED, LOW);   // Turn off the LED
    }
  }
  Serial.println();
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "AulaIoT-";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("connected to the Broker");
      client.subscribe("IPB/IoT/Class02/BikalDibya/a43323a57096/LED");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
```

```
  }
}

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  Serial.println("\n Start Setup");
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  while (WiFi.status() != WL_CONNECTED) {
    WiFi.reconnect();
  }
  if (!client.connected()) {
    Serial.println("Node disconnected from Broker. Trying to connect.. ");
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastTime > 2000) {
    lastTime = now;
    ++value;
    snprintf(msg, Msg_Size, "hello world #%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("IPB/IoT/Aula02/AlunoX", msg);
  }
}
```

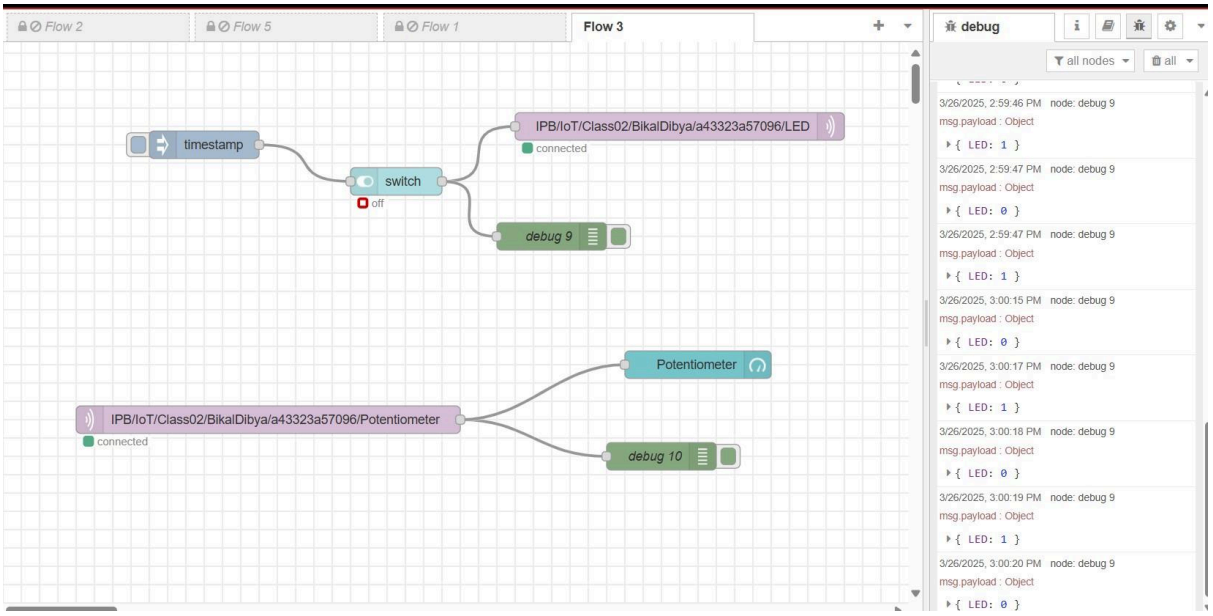
9. Keeping the functionality of the previous exercise of turning the LED on/off, connect a potentiometer to the ESP as shown in the following figure

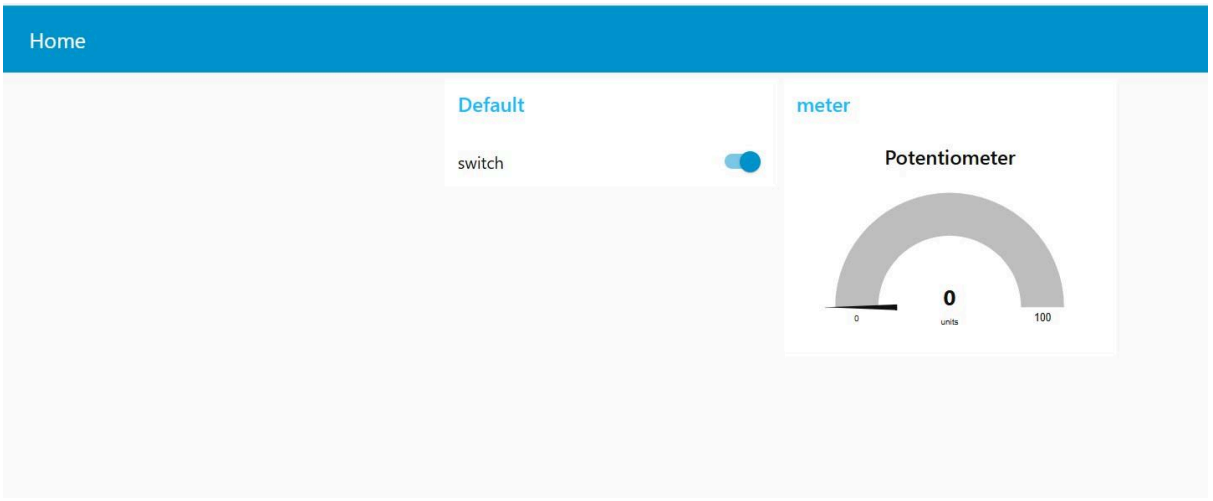
```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

#define LED 33
#define POTENTIOMETER_PIN 36  // ESP32 ADC Pin
```

```
const char* mqtt_server = "broker.emqx.io";
const char* ssid = "agents";
const char* password =
"QgC9O8VucAByqvVu5Rruv1zdpqM66cd23KG4ElV7vZiJND580bzYvaHqz5k07G2";

WiFiClient espClient;
PubSubClient client(espClient);

long lastTime = 0;
const int Msg_Size = 50;
char msg[Msg_Size];
int value = 0;

void setup_wifi() {
  Serial.println("Starting Setup Wifi");
  delay(10);
  Serial.println();
  Serial.print("Connecting to Wifi ");
  WiFi.begin(ssid, password);

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

void callback(String topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  String strPayload = "";
  for (int i = 0; i < length; i++) {
    strPayload = strPayload + (char)payload[i];
  }

  Serial.println(strPayload);

  // Allocate the JSON document
  JsonDocument doc;
  DeserializationError error = deserializeJson(doc, strPayload);

  if (error) {
    Serial.print("JSON deserialization failed: ");
    Serial.println(error.f_str());
    return;
```

```
  }

  // LED Control Topic
  if (topic == "IPB/IoT/Class02/BikalDibya/a43323a57096/LED") {
    if (doc["LED"] == 1) {
      digitalWrite(LED, HIGH);  // Turn on the LED
    } else {
      digitalWrite(LED, LOW);   // Turn off the LED
    }
  }
  Serial.println();
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "AulaIoT-";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("Connected to the Broker");
      client.subscribe("IPB/IoT/Class02/BikalDibya/a43323a57096/LED");
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  Serial.println("\nStart Setup");
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  while (WiFi.status() != WL_CONNECTED) {
    WiFi.reconnect();
  }
  if (!client.connected()) {
    Serial.println("Node disconnected from Broker. Trying to connect.. ");
    reconnect();
  }
  client.loop();
```

```
  long now = millis();
  if (now - lastTime > 2000) {
    lastTime = now;

    // Read Potentiometer value (0-4095 for ESP32 ADC)
    int potValue = analogRead(POTENTIOMETER_PIN);
    Serial.print("Potentiometer Value: ");
    Serial.println(potValue);

    // Convert value to JSON format and publish
    StaticJsonDocument<50> doc;
    doc["Potentiometer"] = potValue;
    char buffer[50];
    serializeJson(doc, buffer);

    client.publish("IPB/IoT/Class02/BikalDibya/a43323a57096/Potentiometer", buffer);
  }
}
```




Home
Default
switch
meter
Potentiometer
0
units
0
100