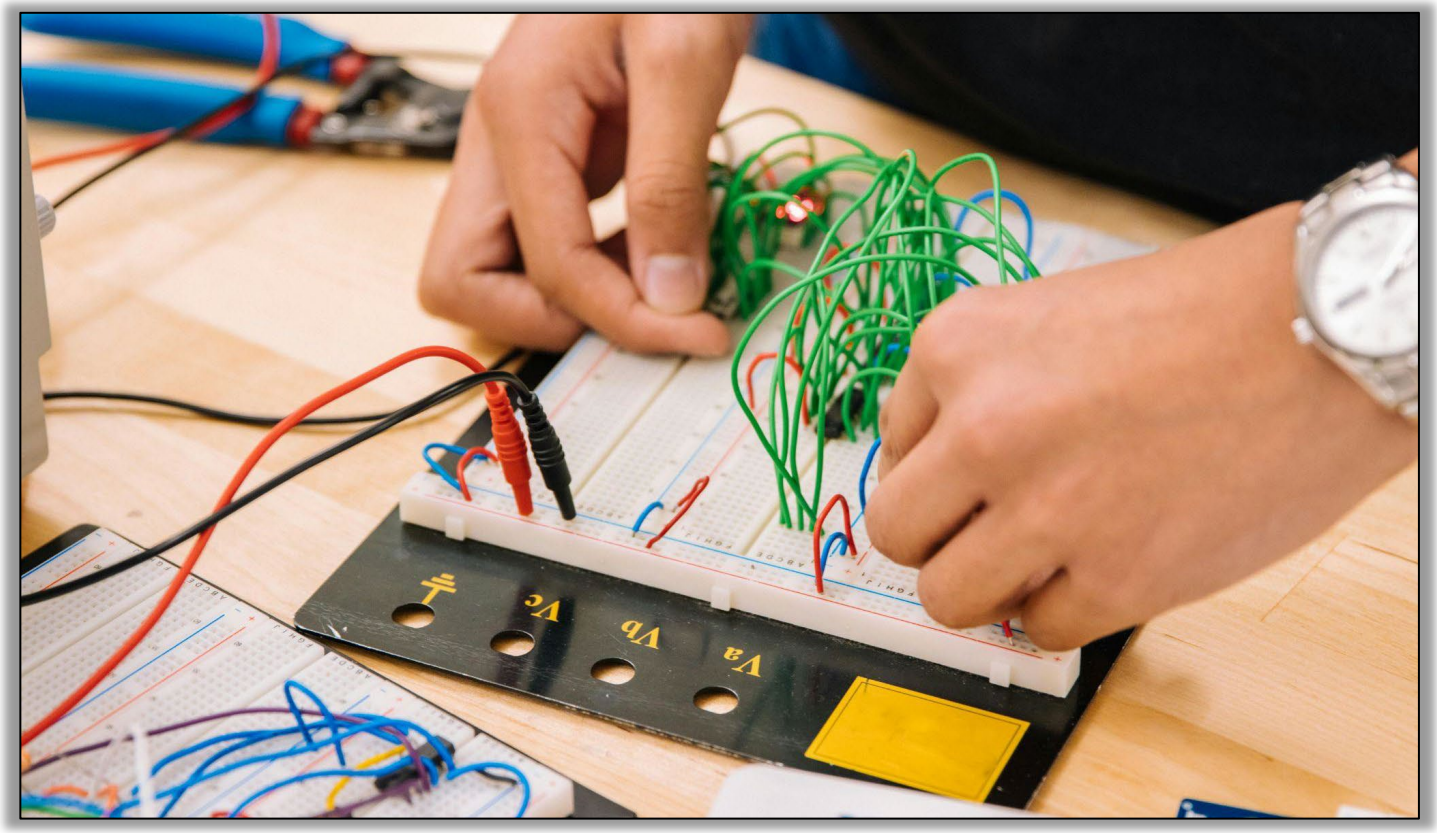
COM382 MICROPROCESSORS

Final Project Report

Smart Home Monitoring



Participants:

B2105.010052	Tariq Ahmad	(Section 4 - Wednesday)
B2205.010035	Daniah Ayad Tareq Al-Sultani	(Section 3 - Tuesday)
B2105.010055	Yousef Aboismail	(Section 3 - Tuesday)
B2105.010002	Mohammad Rauf	(Section 3 - Tuesday)

Course Instructor:

Öğr. Gör. Dr. MHD WASIM RAED

Introduction:

In the rapidly evolving landscape of smart home technology, integrating multiple sensors and devices into a cohesive and efficient system is paramount. This project, titled "Home Automation using microcontrollers," explores the development and implementation of an advanced home automation system. The primary aim is to create a seamless and responsive network that monitors environmental conditions and security parameters, thereby enhancing the convenience and safety of modern living spaces.

By leveraging the capabilities of microcontrollers and employing both local and remote monitoring through the Blynk app, this project highlights the practical applications of IoT in modern home automation. The system ensures that users are always informed about the state of their home environment, providing a comprehensive and innovative solution for enhancing home automation.

Project Description:

The system consists of two nodes, Node 1 and Node 2, which are responsible for collecting data from various sensors. Node 1 is equipped with a DHT22 temperature and humidity sensor, while Node 2 integrates a water detection sensor and a motion sensor. Both nodes are built using ESP32 LOLIN S2 mini microcontrollers.

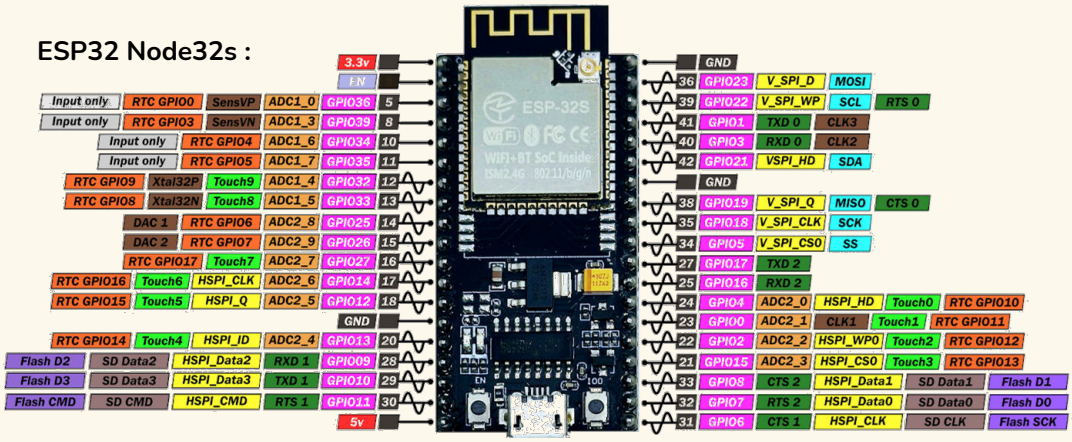
The sensor data from Node 1 and Node 2 is transmitted wirelessly to a central hub, the ESP32 Node32S, using the ESP-NOW protocol. This hub acts as a gateway, receiving data from the nodes and forwarding it to an ESP8266 LolinMCU v3 microcontroller using UART serial communication.
The ESP8266 LolinMCU v3 is responsible for interfacing with the Blynk cloud platform, which provides a user-friendly interface for monitoring and controlling the home automation system. The collected sensor data is sent to the Blynk platform, where it can be visualized and analyzed in real-time.

The project's objective is to create a reliable and efficient home automation system that allows users to monitor environmental conditions, detect water leaks, and track motion within their homes. By leveraging the capabilities of multiple microcontrollers and wireless communication technologies, the system aims to provide a comprehensive solution for home automation and security.

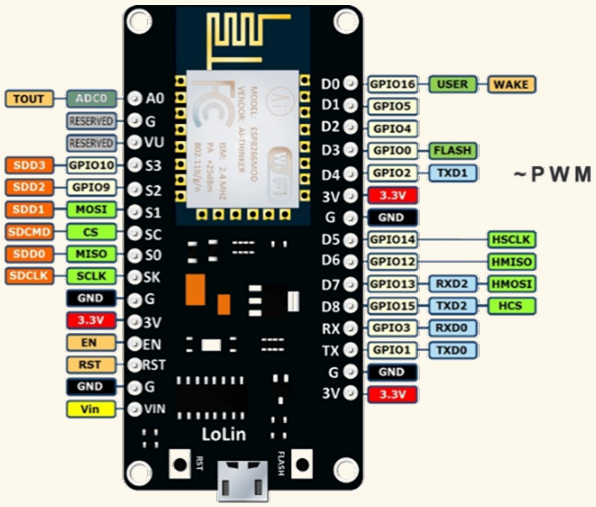
In the following sections, we will delve into the detailed implementation of each component, including the hardware setup, software configurations, and communication protocols used in this project.

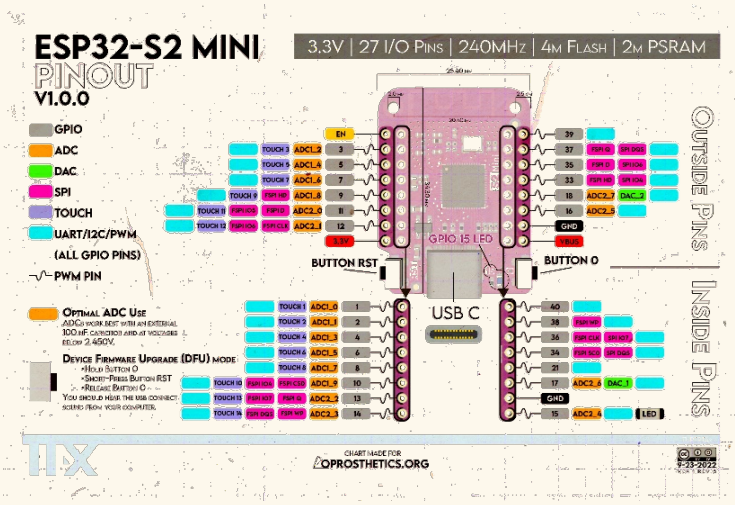
The microcontrollers used and their pins:

ESP32 Node32s :



ESP8266 LOLIN NodeMCU :

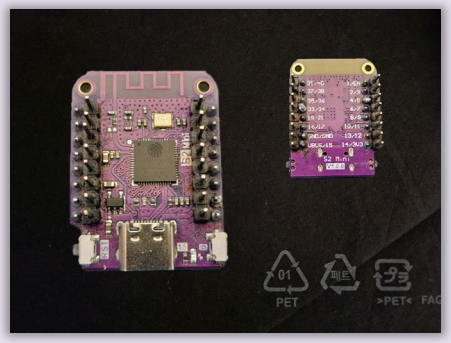




***The exact ones used followed by their actual images and their capabilities are specified in the next page.**

Microcontrollers:

ESP32 LOLIN-S2 Mini:



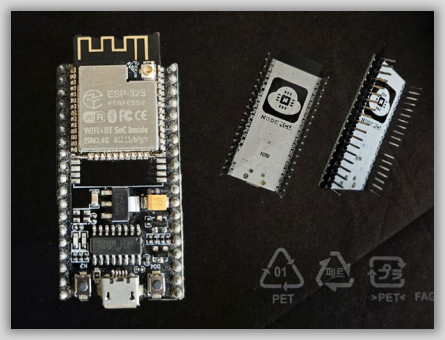
The ESP32 LOLIN S2 Mini is a compact development board based on the ESP32-S2 microcontroller from Espressif Systems, ideal for IoT and embedded projects.

Key features:

- ESP32-S2 Chip: 32-bit microprocessor with Wi-Fi and Bluetooth connectivity.
- Small Form Factor: Measures 36mm x 25mm, suitable for space-constrained projects and wearables.
- Connectivity & Interfaces: Includes USB support, capacitive touch sensing, and multiple GPIO pins for sensors and actuators.
- Programming Options: Compatible with Arduino IDE and Espressif's IoT Development Framework (ESP-IDF).
- Performance: Dual-core processor, 320KB RAM, and 4MB Flash for efficient execution of complex tasks.
- Applications: Ideal for smart home devices, environmental monitoring, wearables, and more.
- Accessibility: Affordable and easy to use for hobbyists, makers, and professionals.

With its low power consumption and versatile capabilities, the ESP32 LOLIN S2 Mini fosters innovation in the world of connected devices.

ESP32 Node32s :



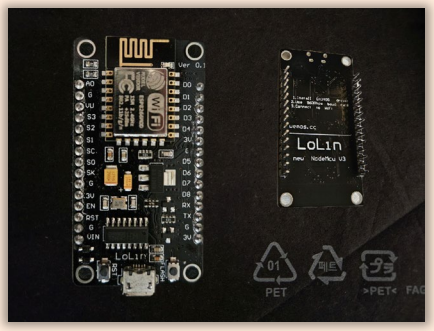
The ESP32, developed by Espressif Systems, is a powerful microcontroller popular in IoT projects.

Key Features:

- Wireless Communication: Integrated 2.4 GHz Wi-Fi and Bluetooth (Classic and LE).
- High Performance: Dual-core Tensilica LX6 microprocessor.
- Versatile Interfaces: Numerous GPIO pins for connecting sensors, actuators, and peripherals.
- Efficiency: Multiple power modes, ideal for battery-powered applications.
- Rich Peripherals: Includes ADCs, DACs, UART, SPI, I2C, and PWM.
- Security: Cryptographic hardware acceleration, secure boot, and flash encryption.

The ESP32 offers a comprehensive solution for modern IoT needs with its robust features and efficiency.

ESP8266 LOLIN NodeMCU :



The ESP8266 LOLIN-NodeMCU is a microcontroller board with integrated Wi-Fi, ideal for IoT projects.

Key Features:

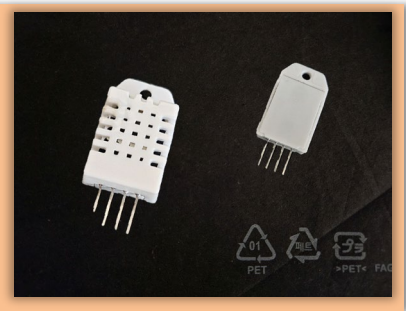
- Processor: 32-bit Tensilica L106 RISC, low power consumption, up to 160 MHz.
- Compact and Robust: Suitable for industrial environments with a wide operating temperature range.
- Pins: 30 physical pins including 13 digital GPIOs, 1 analog (A0), and 7 power pins (3 x 3.3V, 4 x GND).
- Communication: Supports UART, I2C, and SPI protocols.
- Firmware: Open-source Lua-based NodeMCU firmware with an asynchronous event-driven model, over 70 built-in C modules, and nearly 20 Lua modules.

Popular among hobbyists, hackers, and students for its ease of use, flexibility, and affordability.

Sensors:

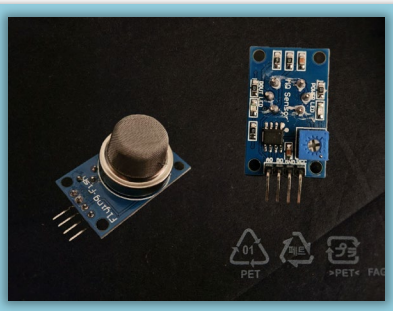
DHT22 temperature and humidity sensor :

The DHT22 is a low-cost digital sensor that measures temperature and humidity, offering high accuracy and reliability. It operates within a temperature range of -40 to 80°C with an accuracy of ±0.5°C and a humidity range of 0 to 100% RH with an accuracy of ±2-5%. Using a single-wire serial communication protocol, it transmits data digitally, eliminating the need for analog-to-digital conversion. The sensor uses a capacitive humidity sensor and a thermistor to capture environmental data, which it sends in a 40-bit format. Ideal for home automation, weather stations, HVAC systems, and agricultural monitoring, the DHT22 is valued for its ease of use and affordability, despite a slower update rate of once every two seconds.



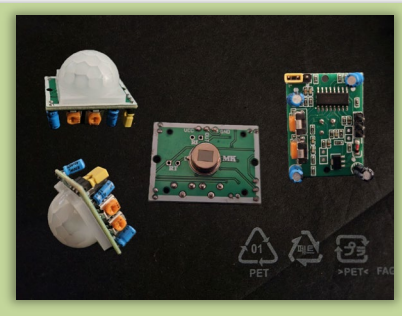
MQ-2 Gas sensor :

The MQ-2 gas sensor is a versatile and cost-effective sensor widely used for detecting a variety of gasses, including methane, butane, propane, LPG, hydrogen, alcohol, smoke, and carbon monoxide. It operates on the principle of a tin dioxide (SnO2) sensitive layer that has low conductivity in clean air and higher conductivity when exposed to combustible gasses. This change in conductivity is used to measure the concentration of the gas. The MQ-2 provides an analog output signal proportional to the gas concentration, making it easy to interface with microcontrollers for monitoring and alert systems. It operates at a voltage of 5V and requires a preheat time to stabilize the sensor for accurate readings. The MQ-2 is known for its high sensitivity, quick response time, and long lifespan. It is commonly used in safety applications such as gas leak detection, smoke alarms, and fire detection systems due to its reliability and ability to detect multiple gasses, ensuring comprehensive monitoring and safety in various environments.



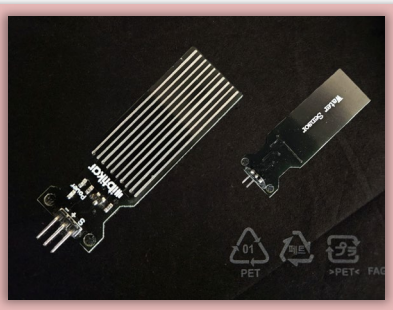
Motion sensor :

A motion sensor, also known as a motion detector, is an electronic device designed to detect and measure movement, alerting users to potential intruders or unauthorized access. It works by detecting changes in the environment, such as the presence of people or objects, and sending a signal to a control panel or monitoring center when motion is detected. There are two primary types of motion sensors: active, which emit a signal and measure the reflection or disturbance caused by an object's presence, and passive, which detect changes in the environment without emitting a signal, such as the infrared radiation emitted by the human body. Motion sensors typically consist of a sensor module, a microcontroller, and an output circuit, and may include additional components such as amplifiers, filters, and voltage regulators to enhance sensitivity and accuracy. They are commonly used in home and business security systems to detect intruders, alert users to restricted areas, and save energy by powering lights only when needed, and are also used in various applications, including automated lighting control, home control, energy efficiency, and other useful systems.



Water detector :

A water sensor is a device designed to detect the presence of water, often used to prevent water damage by alerting users to potential leaks or flooding. These sensors can be categorized into spot leak detectors, under carpet leak detectors, hydroscopic tape-based sensors, and leak detection cables, each suited for specific applications and environments. They work by detecting changes in the environment, such as the presence of water or moisture, and sending a signal to a control panel or monitoring center when water is detected. Some water sensors can detect water at a single point, while others can monitor larger areas or even entire zones via cable-type systems. The sensors can be placed in areas where water damage is likely to occur, such as near appliances or plumbing fixtures that use water and can send notifications to homeowners through smartphone apps if Wi-Fi is enabled. Proper installation by experienced professionals is crucial to ensure optimal placement and functioning of the sensors.



Codes for the controllers:

Node 1 :

1. Library Inclusions and Definitions :

a. The code includes the necessary libraries for ESP-NOW communication, WiFi functionality, and the DHT sensor.
b. It defines the GPIO pin and type for the DHT22 sensor and sets up the pin for an onboard LED.

```
include <esp_now.h>, <WiFi.h>, <DHT.h>
define DHTPIN ← 5, DHTTYPE ← DHT22
create instance dht ← DHT(DHTPIN, DHTTYPE)
define ledPin ← 15
```

2. Global Variables and Structures :

a. An instance of the DHT sensor is created.A structure, `Node1Data`, is defined to hold the sensor data including an ID, temperature, and humidity.
b. An instance of this structure is created to hold the data that will be sent.
c. The MAC address of the central hub (which will receive the data) is defined.

```
struct Node1Data ← {id, temperature, humidity}
create instance myData ← Node1Data()
define hubAddress[] ← {0x78, 0x21, 0x84, 0xC6, 0xD2, 0xEC}
```

3. Setup Function :

a. The serial communication is initialized for debugging purposes.The LED pin is configured as an output and turned off initially.
b. The DHT sensor is initialized.
c. The WiFi is set to station mode and disconnected from any network to ensure ESP-NOW operates in a controlled manner.
d. ESP-NOW is initialized, and if it fails, an error message is printed.
e. The central hub is added as a peer in the ESP-NOW network. If this fails, an error message is printed.

```
setup():
    initialize serial communication
    pinMode(ledPin, OUTPUT)
    digitalWrite(ledPin, LOW)
    dht.begin()
    WiFi.mode(WIFI_STA), WiFi.disconnect()
    if esp_now_init() != ESP_OK:
        print "Error initializing ESP-NOW"
        return
    peer ← {hubAddress, 0, false}
    if esp_now_add_peer(&peer) != ESP_OK:
        print "Failed to add peer"
        return
```

4. Loop Function :

a. Reads the temperature and humidity values from the DHT sensor. Checks if the readings are valid (not NaN).
b. Populates the `Node1Data` structure with the sensor readings and a predefined ID.
c. Prints the sensor data to the serial monitor for debugging.
d. Sends the data to the central hub using ESP-NOW. The result of the send operation is checked, and success or failure is printed to the serial monitor.
e. Toggles the state of the onboard LED based on the success of the data transmission.
f. Waits for 500 milliseconds before repeating the process.

```
loop():
    temperature ← dht.readTemperature()
    humidity ← dht.readHumidity()
    if isnan(temperature) ∨ isnan(humidity):
        print "Failed to read from DHT sensor!"
        return
    myData.id ← 1
    myData.temperature ← temperature
    myData.humidity ← humidity
    print myData
    result ← esp_now_send(hubAddress, &myData, sizeof(myData))
    if result == ESP_OK:
        print "SUCCESSFULLY"
        ledState ← digitalRead(ledPin)
        digitalWrite(ledPin, !ledState)
    else:
        print "FAILED TO SEND"
    delay(500)
```

This code continuously reads sensor data from a DHT22, sends it to a predefined central hub using ESP-NOW, and toggles an onboard LED based on the success of the data transmission.

Node 2 :

1. Library Inclusions and Pin Definitions :

a. The code includes the necessary libraries for ESP-NOW communication and WiFi functionality.
b. Defines GPIO pins for the water sensor (`WATER_SENSOR_PIN`) and motion sensor (`MOTION_SENSOR_PIN`).
c. Defines the GPIO pin for an onboard LED (`ledPin`).

```
include <esp_now.h>, <WiFi.h>
define WATER_SENSOR_PIN ← 3, MOTION_SENSOR_PIN ← 5
define ledPin ← 15
```

2. Global Variables and Structures :

a. Defines a structure, `Node2Data`, to store sensor data, including an ID, water detection status, and motion detection status.
b. Creates an instance of this structure, `myData`, to hold the data that will be sent.
c. Specifies the MAC address of the central hub (`hubAddress`) to which the data will be sent.

```
struct Node2Data ← {id, waterDetected, motionDetected}
create instance myData ← Node2Data()
define hubAddress[] ← {0x78, 0x21, 0x84, 0xC6, 0xD2, 0xEC}
```

3. Setup Function :

a. Initializes serial communication for debugging purposes.
b. Configures the sensor pins as input and the LED pin as an output, initializing the LED to off.
c. Sets the WiFi mode to station mode and disconnects from any network to ensure ESP-NOW operates correctly.
d. Initializes ESP-NOW and checks for initialization success. If initialization fails, an error message is printed.
e. Adds the central hub as a peer in the ESP-NOW network by specifying its MAC address and setting encryption to false. If this addition fails, an error message is printed.

```
setup():
    initialize serial communication
    pinMode(WATER_SENSOR_PIN, INPUT)
    pinMode(MOTION_SENSOR_PIN, INPUT)
    pinMode(ledPin, OUTPUT)
    digitalWrite(ledPin, LOW)
    WiFi.mode(WIFI_STA), WiFi.disconnect()
    if esp_now_init() != ESP_OK:
        print "Error initializing ESP-NOW"
        return
    peer ← {hubAddress, 0, false}
    if esp_now_add_peer(&peer) != ESP_OK:
        print "Failed to add peer"
        return
```

4. Loop Function :

a. Reads the status of the water sensor and motion sensor using `digitalRead`, storing the results in `waterDetected` and `motionDetected` respectively.
b. Populates the `Node2Data` structure (`myData`) with the sensor readings and a predefined ID.
c. Prints the sensor data to the serial monitor for debugging.
d. Sends the populated data structure to the central hub using ESP-NOW. The result of the send operation is checked, and success or failure is printed to the serial monitor.
e. Toggles the state of the onboard LED based on the success of the data transmission.
f. Waits for 500 milliseconds before repeating the process.

```
loop():
    waterDetected ← digitalRead(WATER_SENSOR_PIN)
    motionDetected ← digitalRead(MOTION_SENSOR_PIN)
    myData.id ← 2
    myData.waterDetected ← waterDetected
    myData.motionDetected ← motionDetected
    print myData
    result ← esp_now_send(hubAddress, &myData, sizeof(myData))
    if result == ESP_OK:
        print "SUCCESSFULLY"
        ledState ← digitalRead(ledPin)
        digitalWrite(ledPin, !ledState)
    else:
        print "FAILED TO SEND"
    delay(500)
```

This code continuously monitors a water sensor and a motion sensor, sending their status to a central hub using ESP-NOW. The onboard LED is toggled based on the success of each data transmission, providing visual feedback.

Main Receiver :

1. Library Inclusions and Pin Definitions :

a. The code includes the necessary libraries for WiFi, ESP-NOW communication, and UART communication.
b. Defines a `HardwareSerial` instance for UART communication.
c. Defines GPIO pins for the gas sensor (`gasSensorPin`) and an onboard LED (`ledPin`).

```
include <WiFi.h>, <esp_now.h>, <HardwareSerial.h>
create instance mySerial ← HardwareSerial(2)
define gasSensorPin ← 39, ledPin ← 2
```

2. Global Variables and Structures :

a. Declares variables to store sensor data: `gazz` (gas), `temp` (temperature), `hum` (humidity), `mot` (motion), and `wat` (water).
b. Defines two structures: `Node1Data` to store temperature and humidity data from Node 1, and `Node2Data` to store water detection and motion detection data from Node 2.
c. Creates instances of these structures: `node1Data` and `node2Data`.

```
initialize gazz ← 0, temp ← 0, hum ← 0, mot ← 0, wat ← 0
struct Node1Data ← {id, temperature, humidity}
struct Node2Data ← {id, waterDetected, motionDetected}
create instances node1Data ← Node1Data(), node2Data ← Node2Data()
```

3. Callback Function :

a. `onDataRecv` is a callback function that handles incoming data received via ESP-NOW.
b. Checks the source of the data (Node 1 or Node 2) based on the first byte of the incoming data.
c. Copies the received data into the appropriate structure (`node1Data` or `node2Data`) using `memcpy`.
d. Updates the global variables (`temp`, `hum`, `mot`, `wat`) with the received data.

```
onDataRecv(mac, incomingData, len):
    if incomingData[0] == 1:
        memcpy(&node1Data, incomingData, sizeof(node1Data))
        temp ← (int)node1Data.temperature
        hum ← (int)node1Data.humidity
    else if incomingData[0] == 2:
        memcpy(&node2Data, incomingData, sizeof(node2Data))
        mot ← node2Data.motionDetected
        wat ← node2Data.waterDetected
```

4. Setup Function :

a. Initializes serial communication for debugging.
b. Initializes UART serial communication for the second serial interface (`mySerial`) with specified parameters.
c. Configures the gas sensor pin as an input and the LED pin as an output, initializing the LED to off.
d. Sets the WiFi mode to station mode to use ESP-NOW.
e. Initializes ESP-NOW and checks for initialization success. If initialization fails, an error message is printed.
f. Registers the callback function `onDataRecv` to handle incoming data.
g. Prints a debug message indicating the completion of the setup.

```
setup():
   initialize serial communication
   mySerial.begin(9600, SERIAL_8N1, 16, 17)
   pinMode(gasSensorPin, INPUT)
   pinMode(ledPin, OUTPUT)
   digitalWrite(ledPin, LOW)
   WiFi.mode(WIFI_STA)
   if esp_now_init() != ESP_OK:
       print "Error initializing ESP-NOW"
       return
   esp_now_register_recv_cb(onDataRecv)
   print "Setup complete, starting loop..."
```

5. Loop Function :

a. Reads the value from the gas sensor using `analogRead` and stores it in the variable `gazz`.
b. Prints the received sensor data (temperature, humidity, gas, motion, water) to the serial monitor for debugging using `Serial.printf`.
c. Sends the sensor data to the ESP8266 board via UART using `mySerial.print` and `mySerial.println` with data separated by hyphens.
d. Toggles the state of the onboard LED by reading its current state and writing the opposite state to the pin.
e. Waits for 1 second before repeating the process.

```
loop():
   gazz ← analogRead(gasSensorPin)
   print temp, hum, gazz, mot, wat
   mySerial.print(temp, "-", hum, "-", gazz, "-", mot, "-", wat)
   ledState ← digitalRead(ledPin)
   digitalWrite(ledPin, !ledState)
   delay(1000)
```

This code continuously reads data from a gas sensor, receives data from two remote nodes (one providing temperature and humidity, the other providing water and motion detection), and sends all this data to an ESP8266 board via UART communication. The onboard LED is toggled each time the loop runs, providing a visual indicator of the loop's operation.

Blynk sender :

1. Macro Definitions :

a. Defines Blynk template ID, template name, and authentication token required to connect to Blynk.
b. Defines `BLYNK_PRINT` to use Serial for debugging purposes.

```
1: define BLYNK_TEMPLATE_ID ← "TMPL6nUxJujWn",
BLYNK_TEMPLATE_NAME ← "Home Automation",
BLYNK_AUTH_TOKEN ← "hQcHtb4RUVBz0ZXlJzAj2G5qQ2EogP7a"
2: define BLYNK_PRINT ← Serial
```

2. Library Inclusions :

a. Includes the necessary libraries for using the ESP8266 WiFi and Blynk functionalities.

```
3: include <ESP8266WiFi.h>, <BlynkSimpleEsp8266.h>
```

3. Pin Definitions and WiFi Credentials :

a. Defines the GPIO pin for the onboard LED (`ledPin`).
b. Defines the WiFi credentials (`ssid` and `pass`) needed to connect to the WiFi network.
c. Defines virtual pins for Blynk to send sensor data (`gas_vpin`, `temperature_vpin`, `humidity_vpin`, `motion_vpin`, `waterLvl_vpin`).

```
4: define ledPin ← 2
5: define ssid[] ← "ESP32", pass[] ← "6789054321"
6: define gas_vpin ← V1, temperature_vpin ← V2,
humidity_vpin ← V3, motion_vpin ← V4,
waterLvl_vpin ← V5
```

4. Setup Function :

a. Initializes the LED pin as an output.
b. Sets the initial state of the LED to HIGH (off, since it is active low).
c. Begins serial communication for communication with ESP32 and debugging.
d. Initializes Blynk with the provided authentication token and WiFi credentials.

```
7: setup():
8:     pinMode(ledPin, OUTPUT)
9:     digitalWrite(ledPin, HIGH)
10:    initialize serial communication
11:    Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass)
```

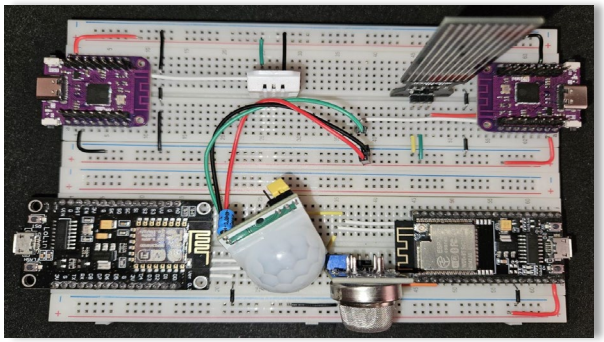
5. Loop Function :

a. Runs the Blynk process to keep the connection alive and handle communication.
b. Checks if there is data available on the serial port.
c. If data is available, reads the incoming data from the serial port until a newline character is encountered.
d. Parses the received data string using `sscanf` into individual sensor values: `temperature`, `humidity`, `gas`, `motion`, and `waterPresence`.
e. Prints the received values to the serial monitor for debugging.
f. Sends the parsed sensor values to the corresponding Blynk virtual pins using `Blynk.virtualWrite`.
g. Reads the current state of the LED pin.
h. Toggles the LED state: if the LED is currently on (LOW), it is turned off (HIGH), and vice versa.

```
12: loop():
13:    Blynk.run()
14:    if Serial.available() > 0:
15:        data ← Serial.readStringUntil('\n')
16:        declare temperature, humidity, gas, motion, waterPresence
17:        sscanf(data.c_str(), "%d-%d-%d-%d-%d", &temperature,
&humidity, &gas, &motion, &waterPresence)
18:        print "Received values:", temperature, humidity, gas,
motion, waterPresence
19:        Blynk.virtualWrite(temperature_vpin, temperature)
20:        Blynk.virtualWrite(humidity_vpin, humidity)
21:        Blynk.virtualWrite(gas_vpin, gas)
22:        Blynk.virtualWrite(motion_vpin, motion)
23:        Blynk.virtualWrite(waterLvl_vpin, waterPresence)
24:        currentState ← digitalRead(ledPin)
25:        if currentState == LOW:
26:            digitalWrite(ledPin, HIGH) // Turn LED off
27:        else:
28:            digitalWrite(ledPin, LOW) // Turn LED on
```

This code continuously reads data sent over the serial port from an ESP32, parses this data to extract sensor values, sends these values to Blynk virtual pins for monitoring, and toggles the state of an onboard LED. The Blynk platform is used for remote monitoring and control, allowing the user to view sensor data through the Blynk app.

The diagram below demonstrates the connections of the sensors to their respective controllers, all of which are placed on two breadboards combined:



Most, if not all the connections can be recognized based off the specifications mentioned within the pseudo codes provided above, the controller codes below (in C++), and the pin diagrams on page 2.

Necessary Libraries:

1. Adafruit Unified Sensor by Adafruit (v1.1.14)
   - Provides a unified sensor abstraction layer used by many Adafruit sensor libraries. This library simplifies sensor integration by providing a common interface for different types of sensors.

2. Blynk by Volodymyr Shymanskyy (v1.3.2)
   - Enables building a smartphone app for your project in minutes. Supports WiFi, Ethernet, and cellular connectivity. Works with over 400 boards including ESP8266, ESP32, Arduino, Raspberry Pi, etc. Essential for creating a user interface to interact with your project remotely.

3. BlynkESP32_BT_WF by Khoi Hoang (v1.2.2)
   - Allows inclusion of both ESP32 Blynk BT/BLE and WiFi libraries. Enables selection of one at reboot or runtime, removing the need to hardcode WiFi and Blynk credentials. Useful for flexible connectivity options for the ESP32.

4. BlynkNcpDriver by Volodymyr Shymanskyy (v0.6.3)
   - Provides a shared interface to services offered by Blynk.NCP. A low-level driver with minimal dependencies. Helps in efficient communication with the Blynk cloud service.

5. DHT kxn by Adafruit (v3.4.4)
   - A backup library for DHT11, DHT22, etc., providing temperature and humidity sensing capabilities. Ensures that you have an alternative in case the primary DHT library fails.

6. DHT sensor library by Adafruit (v1.4.6)
   - The primary library for working with DHT11, DHT22, etc., for temperature and humidity sensing. Essential for integrating DHT sensors into your project.

7. DHT22 by dvarrel (v1.0.6)
   - A specific library for the DHT22 sensor, providing temperature and humidity readings. Focuses on the DHT22 sensor, optimizing performance and reliability.

8. ESP_DoubleResetDetector by Khoi Hoang (v1.3.2)
   - Detects a double reset within a predetermined time using RTC Memory, EEPROM, LittleFS, or SPIFFS for ESP8266 and ESP32. Useful for triggering alternate startup modes based on reset behavior.

9. Makerlabvn_I2C_Motor_Driver by Makerlab.vn (v1.0.5)
   - A driver for 2-DC motors and 2-RC motors using I2C communication. Allows control of up to 5 drivers, simplifying motor control via I2C.

Codes for each controller:

Node 1 :

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

// Define the pin and sensor type for the DHT22 sensor
#define DHTPIN 5
#define DHTTYPE DHT22

// Create an instance of the DHT sensor
DHT dht(DHTPIN, DHTTYPE);

// Define the pin for the onboard LED
const int ledPin = 15;

// Define a structure to store the sensor data
struct Node1Data {
  int id;
  float temperature;
  float humidity;
};

// Create an instance of the Node1Data structure
Node1Data myData;

// Define the MAC address of the central hub
uint8_t hubAddress[] =  {0x78, 0x21, 0x84, 0xC6, 0xD2, 0xEC};

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

  // Set the LED pin as an output and initialize it to off
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Initialize the DHT sensor
  dht.begin();

  // Set the WiFi mode to station mode and disconnect
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

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

  // Add the central hub as a peer
  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, hubAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  // Read temperature and humidity from the DHT sensor
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Check if the sensor readings are valid
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Populate the data structure with sensor data
  myData.id = 1;
  myData.temperature = temperature;
  myData.humidity = humidity;

  // Print the sensor data to the serial monitor
  Serial.print("id = ");
  Serial.print(myData.id);
  Serial.print(", temperature = ");
  Serial.print(myData.temperature);
  Serial.print(", humidity = ");
  Serial.print(myData.humidity);

  // Send the sensor data to the central hub using ESP-NOW
  esp_err_t result = esp_now_send(hubAddress, (uint8_t *)&myData,
sizeof(myData));
  if (result == ESP_OK) {
    Serial.println(" SUCCESSFULLY");

    // Toggle the LED state
    int ledState = digitalRead(ledPin);
    digitalWrite(ledPin, !ledState);
  } else {
    Serial.println(" FAILED TO SEND");
  }

  // Wait before sending the next data
  delay(500);
}
```

Node 2 :

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


// Define pins for the sensors
#define WATER_SENSOR_PIN 3
#define MOTION_SENSOR_PIN 5


// Define the pin for the onboard LED
const int ledPin = 15;


// Define a structure to store the sensor data
struct Node2Data {
  int id;
  bool waterDetected;
  bool motionDetected;
};


// Create an instance of the Node2Data structure
Node2Data myData;


// Define the MAC address of the central hub
uint8_t hubAddress[] = { 0x78, 0x21, 0x84, 0xC6, 0xD2, 0xEC };


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

  // Initialize sensor pins
  pinMode(WATER_SENSOR_PIN, INPUT);
  pinMode(MOTION_SENSOR_PIN, INPUT);

  // Set the LED pin as an output and initialize it to off
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Set the WiFi mode to station mode and disconnect
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

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


  // Add the central hub as a peer
  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, hubAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
}


void loop() {
  // Read sensor data
  bool waterDetected = digitalRead(WATER_SENSOR_PIN);
  bool motionDetected = digitalRead(MOTION_SENSOR_PIN);

  // Populate the data structure with sensor data
  myData.id = 2;
  myData.waterDetected = waterDetected;
  myData.motionDetected = motionDetected;

  // Print the sensor data to the serial monitor
  Serial.print("id = ");
  Serial.print(myData.id);
  Serial.print(", waterDetected = ");
  Serial.print(myData.waterDetected);
  Serial.print(", motionDetected = ");
  Serial.print(myData.motionDetected);

  // Send the sensor data to the central hub using ESP-NOW
  esp_err_t result = esp_now_send(hubAddress, (uint8_t *)&myData,
sizeof(myData));

  if (result == ESP_OK) {
    Serial.println(" SUCCESSFULLY");

    // Toggle the LED state
    int ledState = digitalRead(ledPin);
    digitalWrite(ledPin, !ledState);
  } else {
    Serial.println(" FAILED TO SEND");
  }

  // Wait before sending the next data
  delay(500);
}
```

Main Receiver :

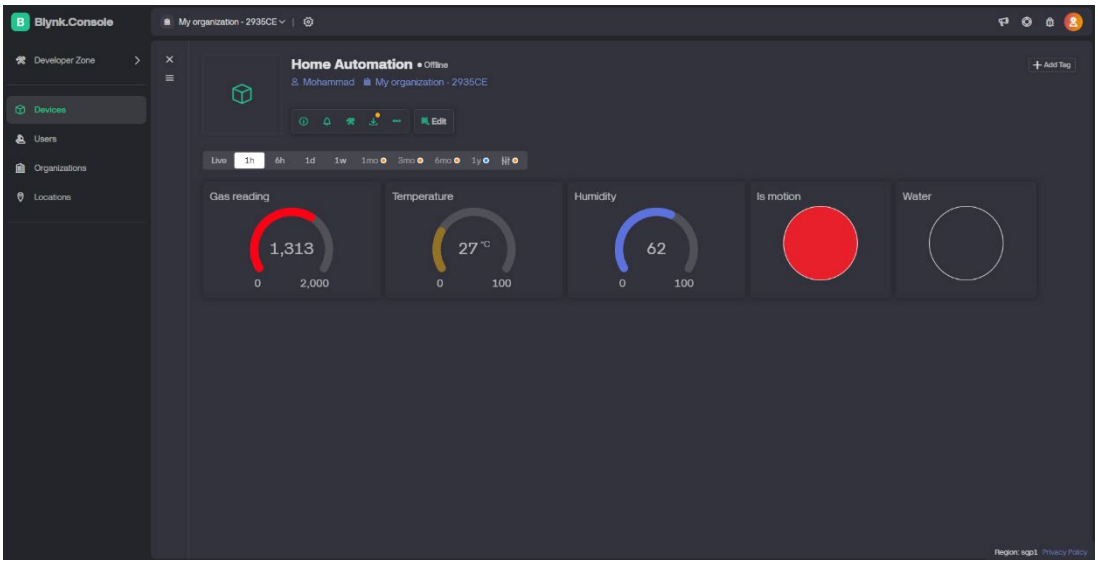
```
#include <WiFi.h>
#include <esp_now.h>
#include <HardwareSerial.h>

// Create a serial instance for UART communication
HardwareSerial mySerial(2);

// Define the pin for the gas sensor
const int gasSensorPin = 39;

// Define the pin for the onboard LED
const int ledPin = 2;

// Variables to store sensor data
int gazz = 0, temp = 0, hum = 0, mot = 0, wat = 0;

// Define structures to store data from Node 1 and Node 2
struct Node1Data {
  int id;
  float temperature;
  float humidity;
};

struct Node2Data {
  int id;
  bool waterDetected;
  bool motionDetected;
};

// Create instances of the data structures
Node1Data node1Data;
Node2Data node2Data;

// Callback function to handle incoming data from ESP-NOW
void onDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len) {
  // Check the source of the data (Node 1 or Node 2)
  if (incomingData[0] == 1) {
    // Data from Node 1 (Temperature and Humidity)
    memcpy(&node1Data, incomingData, sizeof(node1Data));
    temp = (int)node1Data.temperature;
    hum = (int)node1Data.humidity;
  } else if (incomingData[0] == 2) {
    // Data from Node 2 (Water and Motion)
    memcpy(&node2Data,incomingData, sizeof(node2Data));
    mot = node2Data.motionDetected;
    wat = node2Data.waterDetected;
  }
}

void setup() {
  Serial.begin(115200);
  // Start the UART serial communication
  mySerial.begin(9600, SERIAL_8N1, 16, 17);

  // Set the gas sensor pin as input
  pinMode(gasSensorPin, INPUT);

  // Set the LED pin as an output and initialize it to off
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Set the WiFi mode to station mode
  WiFi.mode(WIFI_STA);

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

  // Register the callback function for receiving data
  esp_now_register_recv_cb(onDataRecv);
  // Debug print to confirm setup
  Serial.println("Setup complete, starting loop...");
}

void loop() {
  // Read the gas sensor value
  gazz = analogRead(gasSensorPin);

  // Print the received sensor data
  Serial.printf("Temp: %d, Hum: %d, Gas: %d, Motion: %d, Water: %d\n",
temp, hum, gazz, mot, wat);

  // Send the sensor data to the ESP8266 board via UART
  mySerial.print(temp);
  mySerial.print("-");
  mySerial.print(hum);
  mySerial.print("-");
  mySerial.print(gazz);
  mySerial.print("-");
  mySerial.print(mot);
  mySerial.print("-");
  mySerial.println(wat);

  // Toggle the LED state
  int ledState = digitalRead(ledPin);
  digitalWrite(ledPin, !ledState);

  delay(1000);
}
```

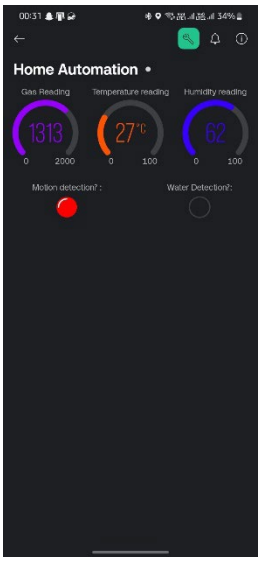
Blynk Sender :

```
// Define Blynk template ID and authentication token
#define BLYNK_TEMPLATE_ID "TMPL6nUxJujWn"
#define BLYNK_TEMPLATE_NAME "Home Automation"
#define BLYNK_AUTH_TOKEN "hQcHtb4RUVBz0ZXlJzAj2G5qQ2EogP7a"


// Define BLYNK_PRINT to use Serial for debugging
#define BLYNK_PRINT Serial


// Include necessary libraries
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>


// Define the pin for the onboard LED
const int ledPin = 2;


// WiFi credentials
char ssid[] = "ESP32";
char pass[] = "6789054321";


// Define virtual pins for Blynk
#define gas_vpin V1

#define temperature_vpin V2

#define humidity_vpin V3

#define motion_vpin V4

#define waterLvl_vpin V5


void setup() {
  // Initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);

  // Initialize the LED state to HIGH (LED off, as it is active low)
  digitalWrite(ledPin, HIGH);

  // Start Serial for communication with ESP32 and for debug
  Serial.begin(9600);

  // Initialize Blynk
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}


void loop() {
  Blynk.run();

  // Check if there is data available on the serial port
  if (Serial.available() > 0) {
    // Read the incoming data
    String data = Serial.readStringUntil('\n');

    int temperature, humidity;
    int gas;
    int motion, waterPresence;

    // Parse the received data
    sscanf(data.c_str(), "%d-%d-%d-%d-%d", &temperature, &humidity, &gas,
&motion, &waterPresence);

    // Print the received values to the Serial Monitor
    Serial.print("Received values: ");
    Serial.print(temperature);
    Serial.print(", ");
    Serial.print(humidity);
    Serial.print(", ");
    Serial.print(gas);
    Serial.print(", ");
    Serial.print(motion);
    Serial.print(", ");
    Serial.println(waterPresence);


    // Send the parsed values to Blynk virtual pins
    Blynk.virtualWrite(temperature_vpin, temperature);
    Blynk.virtualWrite(humidity_vpin, humidity);
    Blynk.virtualWrite(gas_vpin, gas);
    Blynk.virtualWrite(motion_vpin, motion);
    Blynk.virtualWrite(waterLvl_vpin, waterPresence);

    // Read the current state of the LED pin
    int currentState = digitalRead(ledPin);

    // Toggle the LED state
    if (currentState == LOW) {
      digitalWrite(ledPin, HIGH); // Turn LED off
    } else {
      digitalWrite(ledPin, LOW); // Turn LED on
    }
  }
}
```

Blynk Application Interface:

Blynk Web and Mobile Dashboard:





Conclusion:

The home automation system described in your project effectively leverages a multi-node setup to monitor and manage environmental conditions within a home. The system is structured around two primary sensor nodes and a central hub, all utilizing ESP32 and ESP8266 microcontrollers to ensure seamless data collection, transmission, and cloud integration.

Node 1 and Node 2:

- **Node 1** is equipped with a DHT22 sensor for monitoring temperature and humidity. This node continuously collects environmental data and transmits it wirelessly.
- **Node 2** incorporates a water detection sensor and a motion sensor, focusing on security and leak detection. This node similarly sends its data wirelessly.

Both nodes are built using ESP32 LOLIN S2 mini microcontrollers and utilize the ESP-NOW protocol for efficient and low-latency data transmission to the central hub.

Central Hub:

- The central hub is an ESP32 Node32S microcontroller. It receives data from Node 1 and Node 2 via ESP-NOW.
- Upon receiving the sensor data, the hub forwards it to an ESP8266 LolinMCU v3 microcontroller using UART serial communication. This step ensures that all collected data is centralized and prepared for cloud transmission.

Data Integration and Visualization:

- The ESP8266 LolinMCU v3 microcontroller serves as the interface between the collected sensor data and the Blynk cloud platform.
- Utilizing the Blynk library, the system transmits temperature, humidity, water detection, and motion data to the Blynk platform, where it can be visualized and monitored in real-time.

Project Objectives and Achievements:

- The primary goal of the project is to provide a reliable and efficient home automation solution that enhances environmental monitoring and security.
- The integration of multiple microcontrollers and wireless communication technologies ensures a robust and scalable system.
- Real-time data visualization and remote monitoring are achieved through the Blynk platform, offering users a user-friendly interface for managing their home environment.

Key Implementations:

- ESP-NOW Protocol: Ensures low-latency and efficient wireless communication between sensor nodes and the central hub.
- UART Communication: Facilitates reliable data transfer from the ESP32 hub to the ESP8266 microcontroller.
- Blynk Integration: Provides a cloud-based platform for real-time data visualization and remote monitoring, enhancing the system's usability.

This comprehensive approach to home automation combines environmental monitoring and security features, leveraging the strengths of ESP32 and ESP8266 microcontrollers, ESP-NOW protocol, and the Blynk platform.

References:

Renzo Mischianti (2024) *ESP32 nodemcu-32s esp-32s kit: High resolution pinout, datasheet, and Specs*, *Renzo Mischianti.* Available at: https://mischianti.org/esp32-nodemcu-32s-esp-32s-kit-high-resolution-pinout-datasheet-and-specs/ (Accessed: 24 May 2024).

ADC usage of ESP32-S2 Mini - let's control it. Available at: https://www.letscontrolit.com/forum/viewtopic.php?t=9759 (Accessed: 24 May 2024).

ESP32 ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems. Available at: https://www.espressif.com/en/products/socs/esp32 (Accessed: 24 May 2024).