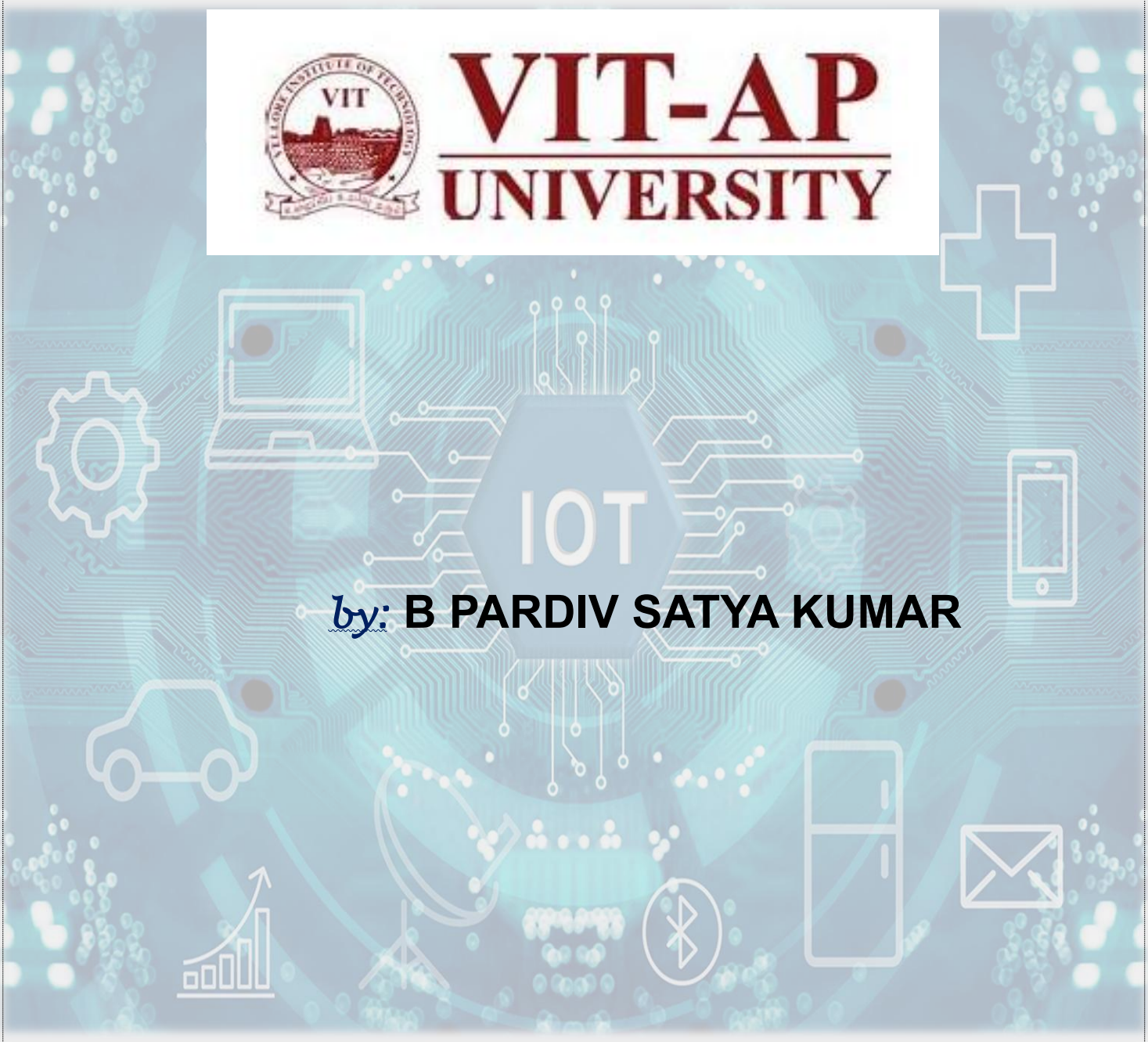
IOT - ENABLED
INTELLIGENT DOOR LOCK SYSTEM
WITH
REMOTE ACCESS



by: B PARDIV SATYA KUMAR

Abstract:

This project presents the design and development of an IoT-enabled smart door security system built using an ESP32 microcontroller. The system combines multiple hardware components—including a servo-controlled smart lock, PIR motion sensor, DHT22 temperature/humidity sensor, keypad module, LED indicators, and a 20x4 I2C LCD display—to provide secure access control and real-time environmental monitoring. MQTT communication is integrated for remote supervision, while a Node-RED dashboard enables seamless visualization and control. The solution demonstrates practical expertise in embedded systems, IoT communication, cloud integration, and real-time automation.

Project Overview:

The objective of this project is to develop a secure, intelligent, and remotely accessible door-locking system suitable for modern IoT-driven environments. The ESP32 functions as the central controller, managing user authentication through a keypad, monitoring motion via a PIR sensor, and tracking environmental conditions using a DHT22 sensor. A 20x4 I2C LCD displays real-time system status, while the servo motor operates the physical lock mechanism.

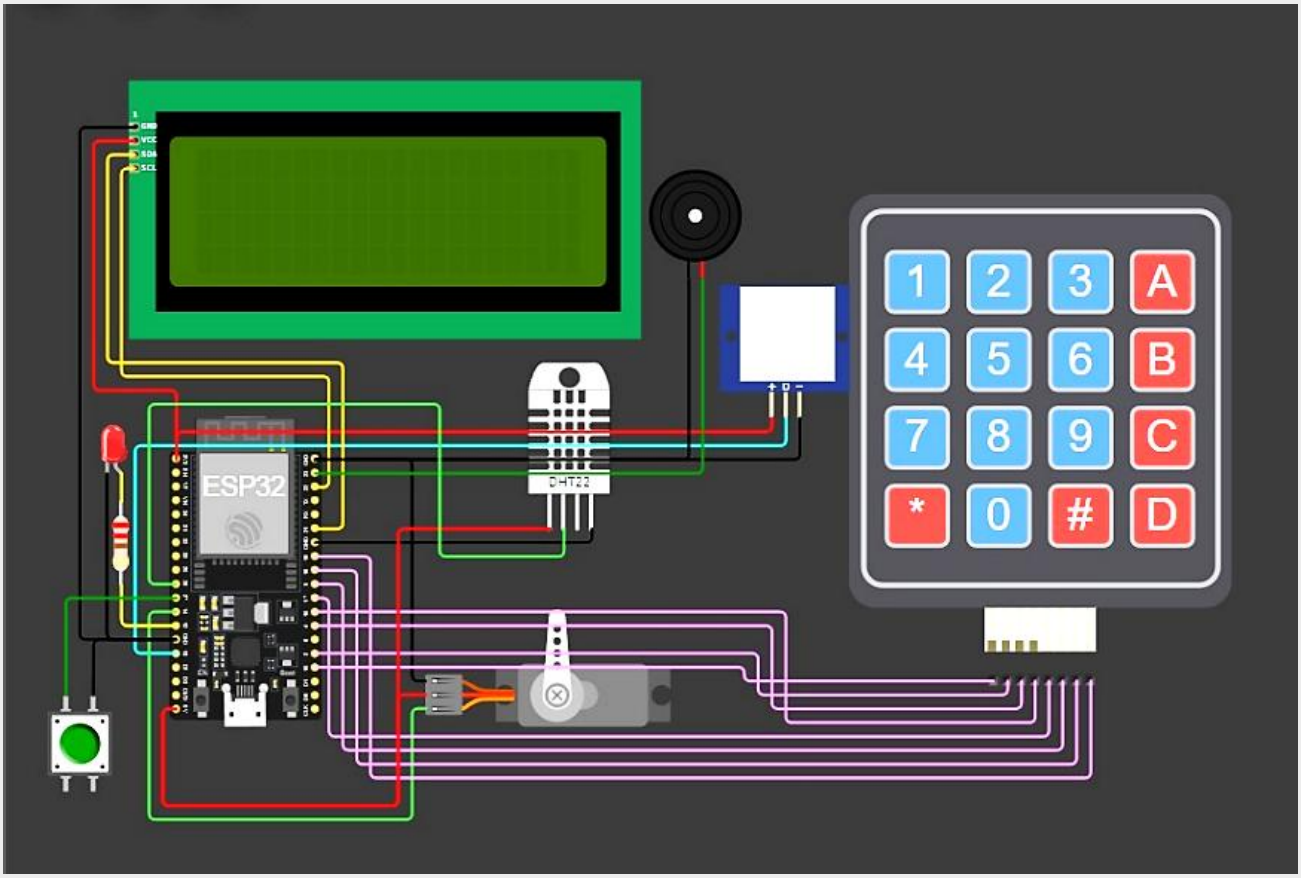
Through MQTT integration and a custom Node-RED dashboard, users can **view live sensor data, receive security alerts, and remotely lock or unlock the door** from any connected device. This project demonstrates strong competencies in embedded hardware interfacing, IoT protocols, access-control automation, and cloud-based remote monitoring.

.

Components and Circuit Diagram:

The main components used in this project include:

- ESP32 Microcontroller
- Servo Motor (for door lock mechanism)
- PIR Sensor (motion detection)
- LED (acts as room light indicator)
- 20x4 I2C LCD Display (for status display)
- Keypad (for entering the password)
- Buzzer (for audible feedback)
- Push Button (for locking the door)
- DHT22 Sensor (for temperature and humidity)
- Wi-Fi and MQTT (for remote communication)


ESP32
DHT22
1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D

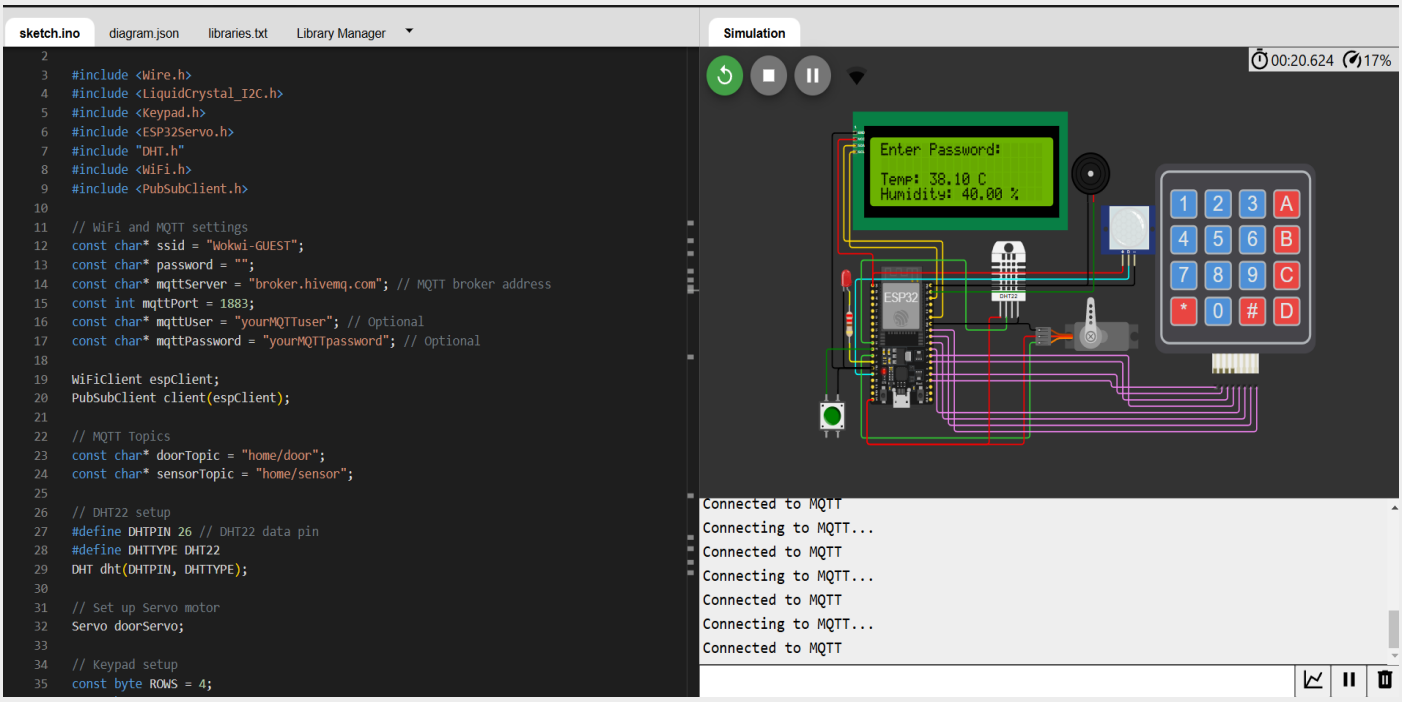
Code Explanation and Integration:

The following code implements the functionality for reading sensor data, controlling the servo motor, handling user input through the keypad, and interacting with the MQTT broker. The code also connects to the MQTT broker, allowing for communication with Node-RED.

The pseudo code for the project is as follows:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <ESP32Servo.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

// DHT22 setup
#define DHTPIN 26
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// WiFi and MQTT configuration
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
const char* mqttServer = "broker.hivemq.com";
const int mqttPort = 1883;
WiFiClient espClient;
PubSubClient client(espClient);

// Set up components
Servo doorServo;
const String password = "1234";
String inputPassword = "";

int pirPin = 13;
int ledPin = 12;
int buzzerPin = 23;
int buttonPin = 27;

LiquidCrystal_I2C lcd(0x27, 20, 4);
bool doorUnlocked = false;

void setup() {
  // Initialize components and connect to WiFi and MQTT
  // Code omitted for brevity
}
void loop() {
  // Functions to handle components and MQTT communication
  // Code omitted for brevity

}
// Additional functions: checkPasswordInput, controlPIRAndLight, updateDHT, etc.
```

Wokwi interface:




```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <ESP32Servo.h>
#include "DHT.h"
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi and MQTT settings
const char* ssid = "Wokwi-GUEST";
const char* password = "";
const char* mqttServer = "broker.hivemq.com"; // MQTT broker address
const int mqttPort = 1883;
const char* mqttUser = "yourMQTTuser"; // Optional
const char* mqttPassword = "yourMQTTpassword"; // Optional

WiFiClient espClient;
PubSubClient client(espClient);

// MQTT Topics
const char* doorTopic = "home/door";
const char* sensorTopic = "home/sensor";

// DHT22 setup
#define DHTPIN 26 // DHT22 data pin
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// Set up Servo motor
Servo doorServo;

// Keypad setup
const byte ROWS = 4;
```

Node-RED Interface Setup:

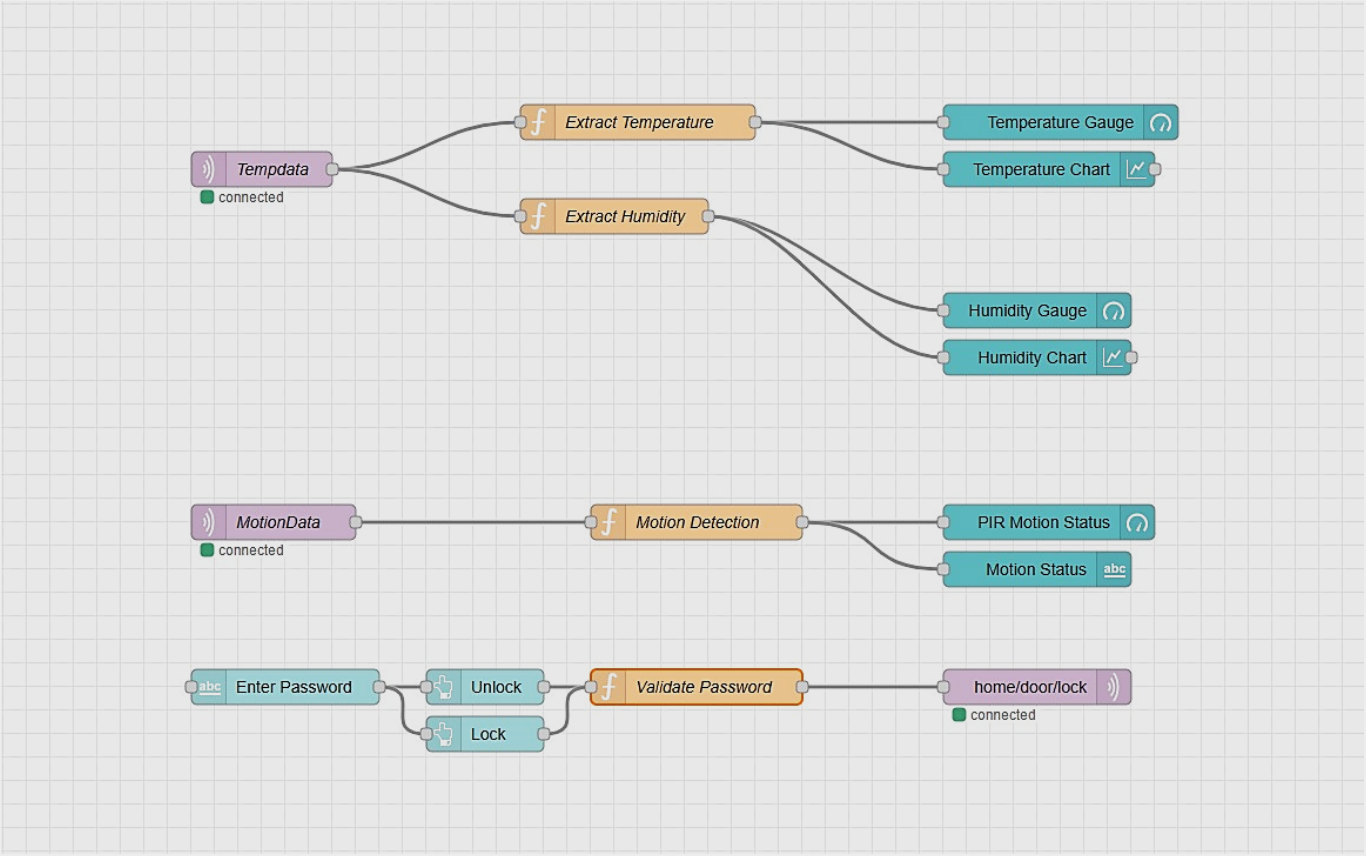
The Node-RED interface provides remote monitoring and control. MQTT topics are used for real-time data exchange between ESP32 and Node-RED. Set up MQTT nodes for each functionality as shown below.

Topics used:
- Publish: home/door/status, home/sensor/temp, home/sensor/humidity, home/sensor/motion
- Subscribe: home/door/lock

Node-RED Flow Configuration (JSON):

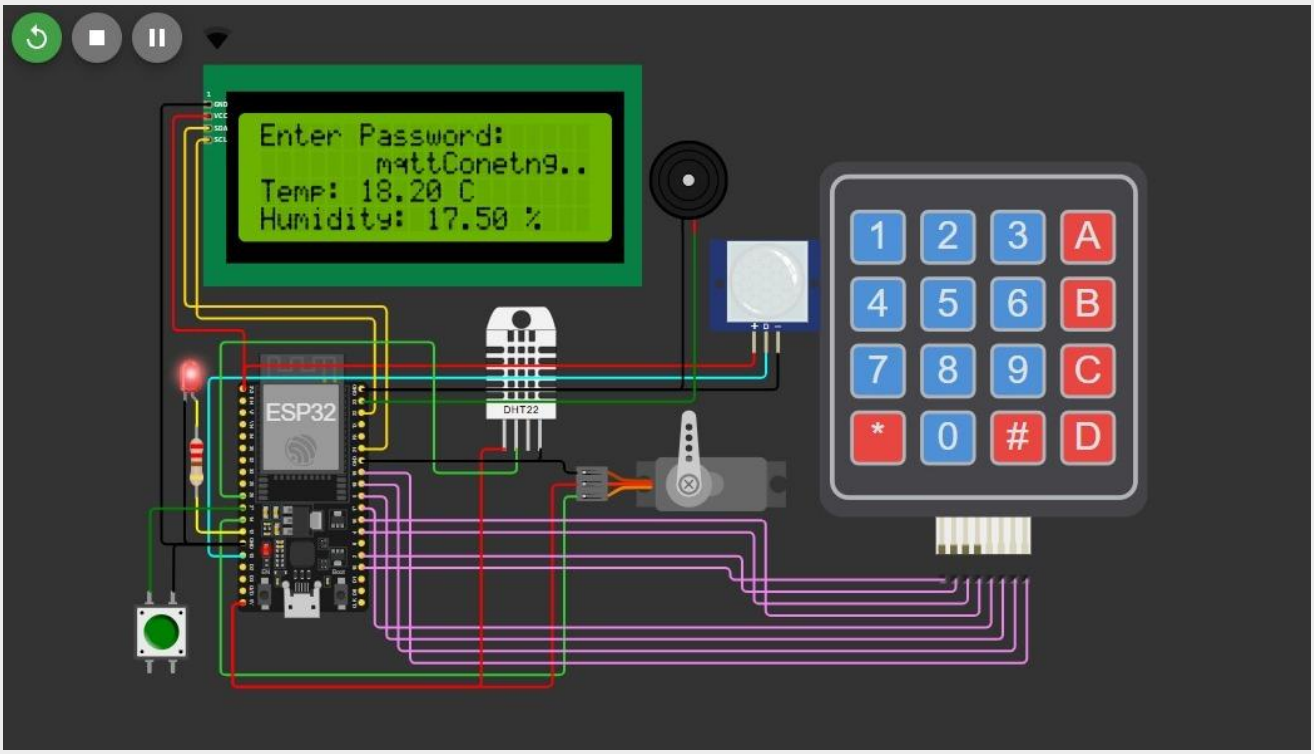
```
[
    {
        "id": "mqtt_in_temp",
        "type": "mqtt in",
        "topic": "home/sensor/temp",
        "broker": "mqtt_broker",
        "x": 120,
        "y": 80,
        "wires": [["temp_display"]]
    },
```

```
    {
        "id": "temp_display",
        "type": "ui_text",
        "label": "Temperature (°C)",
        "x": 300,
        "y": 80
    }
    // Add nodes for humidity, motion, door status, and control button similarly
]
```

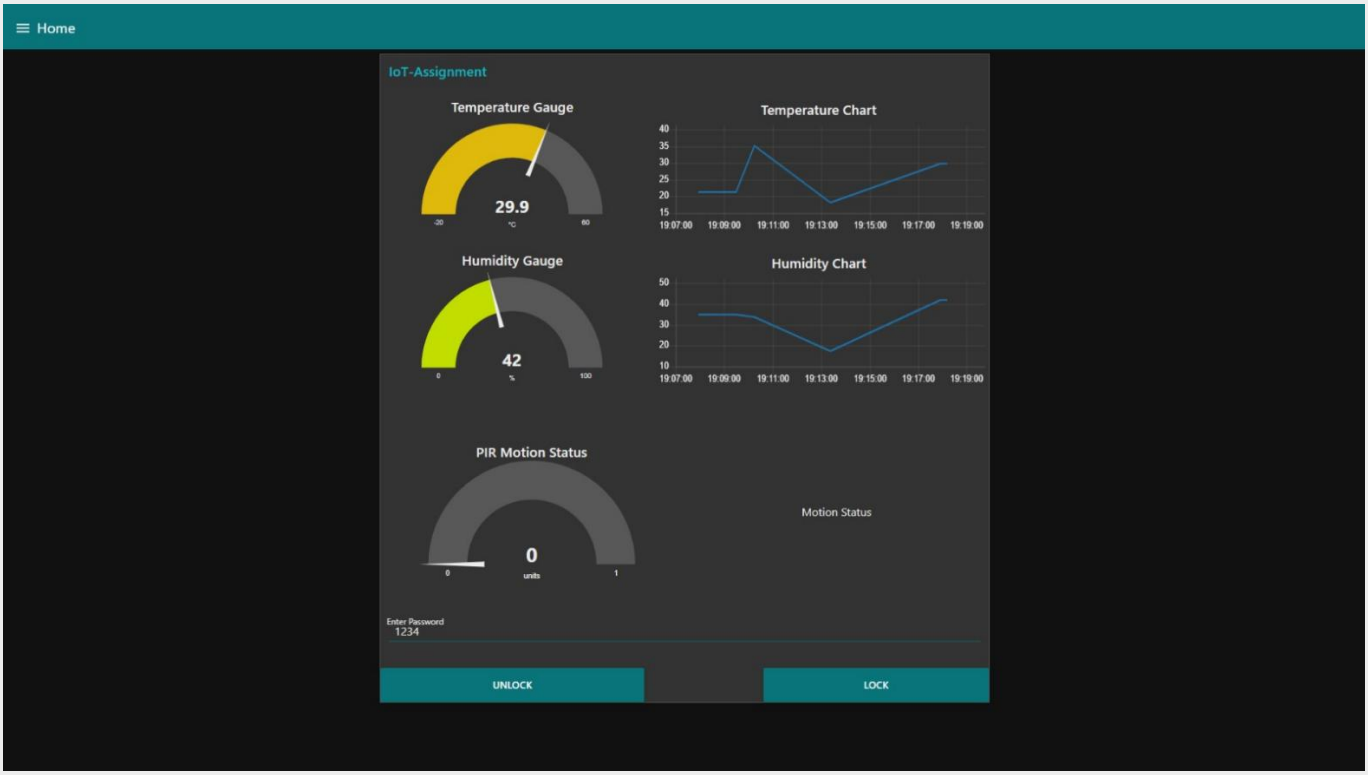


Testing and Conclusion:

Wokwi:



Node-red dashboard:



Use Case:

Residential Applications:

For homeowners, this IoT-based system provides advanced security and control that goes beyond standard smart locks. The door can automatically lock at preset times each night, ensuring the house remains secure even if someone forgets to lock the door manually. Through the Node-RED–based mobile interface, users can remotely unlock the door for guests, delivery agents, or service personnel. The system can also be extended with an **ESP32-CAM module** for real-time video streaming and a **two-way voice communication module**, allowing homeowners to visually verify visitors and speak with them before granting access.

Office & Commercial Usage:

In office environments, the system can be configured to unlock during business hours and automatically secure itself after hours. Each employee can be assigned a unique access code, allowing centralized logging and tracking of entry events. Integration with cameras and voice modules enhances security by enabling remote visitor verification and recording visual logs of entry attempts. The MQTT-based architecture makes it easy to integrate with existing company dashboards or automation workflows.

Why This System Is Superior to Existing Market Gadgets

Unlike many commercially available smart locks that operate as closed, proprietary systems, this solution is **fully customizable**, **open-source**, and **scalable**. Users can modify hardware, add sensors, integrate cameras, and expand software features without vendor restrictions. Commercial gadgets often require monthly subscription fees for cloud storage, remote access, or advanced features; however, this project uses **free, open-source tools** such as Node-RED and MQTT, making it more cost-effective. Additionally, the ESP32-based architecture supports offline operation, ensuring that the system functions even during internet outages—something most cloud-dependent smart locks cannot offer.

By combining flexibility, affordability, and enhanced security features, this system presents a more adaptable and future-ready alternative to typical smart door solutions.

* * * * *