



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and
Engineering**

“Temperature Controlled Self-Regulating System”

*A project submitted
in partial fulfillment of the requirements
for the degree of Bachelor of Technology
in Computer Science and Engineering*

By

Braham Aggarwal (20BCT0052)

D Manjunadha Sharma (20BCT0102)

**Kakumanu L V Surya Anil
(20BCT0191)**

November -2022

UNDERTAKING

This is to declare that the project entitled “**Temperature Controlled Self-Regulation System**” is an original work done by the undersigned, in partial fulfillment of the requirements for the degree “Bachelor of Technology in Computer Science and Engineering” at the School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore.

All the analysis, design, and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or University.

Team Member Names -

Kakumanu L V Surya Anil – 20BCT0191

Braham Aggarwal - 20BCT0052

D. Manjunadha Sharma – 20BCT0102

A handwritten signature in blue ink, appearing to read 'Surya Anil' with 'KLV' written below it.

Signature of Candidate

ABSTRACT

An Implementation of an IoT Device with Ethereum Blockchain to regulate a system.

Team 5, VIT Vellore.

The rapid growth of the internet of things (IoT) in the world in recent years is due to its wide range of usability, adaptability, and smartness. Most IoT applications are performing jobs in an automatic manner without interactions with humans or physical objects. It's required that the current and upcoming devices will be smart, efficient, and able to provide the services to the users to implement such new technology in a secure manner. Thus, security issues are being explored day by day by researchers, IoT devices are typically lightweight and portable, thus it has a number of problems. Such as battery consumption, and memory, and as these devices are working open range so the most important is security. y. So, to counter these different security issues we have used Blockchain and distributed ledger technology to safeguard our data and users' privacy.

Table of Contents -

1. Introduction
2. Literature Survey
3. Proposed Methodology
4. Architecture/Design
5. Description of Various Modules
6. Implementation (Sample Code)
7. Testing (ALL Screen Snapshots)
8. Conclusion and Future Enhancements
9. References

INTRODUCTION

Temperature Controlled Self-Regulation System is an IoT Based Application that controls a fan electronically with an esp32 microcontroller. In This System, we provide a control dashboard to set the desired temperature ranges and provide manual override controls. We used a Blockchain-based authentication system for Enhanced security and access control. Implemented a feedback loop between the IoT Device and Blockchain to have the fan self-regulate to control temperature. Sending a fire warning to the customer via email in case the temperature sensed is above a certain limit.

The purpose of this project is to verify the implementation of IoT devices with Blockchain and to analyze their performance. The implementation of this project is done using an esp32 microcontroller and smart contracts deployed on the Ethereum blockchain.

IoT and blockchain together with distributed ledger technologies enable machine-to-machine trades. It makes use of a set of transactions that are entered into a common ledger distributed across all nodes, checked by many sources, and recorded in a database. IoT and blockchain work together to provide a number of potential advantages, including the ability for smart devices to operate independently without centralized authority. It can also track how devices communicate with each other.

In this application, we will be able to set the desired temperature range and our fan will automatically kick on and off to cool the area surrounding our temperature sensor to the desired temperature setting. Here we will create a dashboard that not only allows you to set the desired temperature but also allows manual override e allows you to set the desired temperature and manual override, enabling us to control the fan from a console. One additional feature it this project is the “Fire Alarm System” which used the SMTP protocol to send the user an Emergency E-mail in case the temperature is above a certain limit. All this is done using an esp32 microcontroller and Ethereum blockchain for validation and processing via the smart contract deployed.

Literature Survey

1. Secure access control to IoT devices using Blockchain

Aim: - The main aim of this paper is provide a decentralized, lightweight, and secure architecture based on blockchain

Methodology: -

- 4-tier architecture consisting of end devices as the first followed by a gateway then private or public Blockchain and finally the cloud storage.
- Whenever a device wants to send data to the cloud or talk with another device it has to use this unique address as a pass in order to carry out the task also known as a transaction.
- Every end device has unique ID.
- Transactions are used for communications between devices.

Advantages: -

- Secure Access control
- Prevents Man-in-middle and DOS attacks
- Unique Identification, Data security

Limitations: -

- Efficiency issues because of using proof-of-work algorithm.

Future Scope: -

- Using proof-of-stake for better performance.

2. Emails-and-the-drive-to-a-successful-iot-security

Aim: - The main aim of this paper is to check whether emails will remain relevant in the future in the domain of IoT or not and the various security aspects related to emails.

Methodology: -

- the number of email users has been consistently increasing by a factor of 3% every year.
- Email security has seen many advances over the years, especially in the area of multi-factor authentication and biometric access.
- email service providers and users should pay close attention to the security of their email services and accounts respectively.
- Emails re the most likely source of data breaches.
- current trends in IoT suggest that there would be an unavoidable reliance on emails to gain access to IoT Control dashboards in the not-too-far future

Limitations: -

- Scope for enhancement in security domain.

3. Fire Alarm System Based on IoT

Aim: - The main aim of this paper is to make a “Fire alarm System” in which the sensor notifies the nearest fire station and sounds an alarm.

Methodology: -

- Hardware Components – ESP8266, Infrared flame sensor, Adaptor, Buzzer, Breadboard.
- Whenever the Infrared flame sensor detects a “fire hazard” it notifies the Node MCU ESP8266 which in turn sends a notification to the nearest fire station and sets off the buzzer.

Advantages: -

- Faster communication during a fire at the fire station.
- Faster notification to the people nearby, hence saving lives.

Future Scope: -

- Can be used in smart buildings and offices if provided with better security.

4. Automatic Temperature Control System Using Arduino

Aim: - The main aim of this paper is to design a simple method of temperature control system automatically.

Methodology: -

- Hardware Components – Arduino UNO, Temperature sensor LM35, LCD display, Fan.
- Displaying the temperature automatically on the LCD display and automatically switching ON/OFF fans to monitor the temperatures on an automatic basis.

Advantages: -

- Presents an application of control theory using ICT and hardware-based temperature control including the design of a circuit.

Limitations: -

- No aspects regarding scalability and security are mentioned in the paper.

Future Scope: -

- Can be used in smart buildings and offices if provided with better security.

5. IoT-based Temperature Control system of Home using an Android device

Aim: - The main goal of this project is to create a voltage control device with a Node MCU board that can be operated remotely over the Internet using either the Android or iOS operating system.

Methodology: -

- Raspberry pi, ESP8266 Node MCU Wi-Fi Module is used.
- Web-based servers, Cloud technology, and Computer Networks concepts are used.
- The software requirements for this project are Arduino IDE, CCS IDE, and the Blynk app.

Advantages: -

- We can control and monitor the temperature of our home easily through a smartphone.

Limitations: -

- Safety and Security aspects are not discussed in this paper.

Future Scope: -

- Creating a Web-based online monitoring server.
- Making notification alerts for the over-temperature situation.

Proposed Methodology -

The methodology of this project includes the following steps -

- Making the circuit diagram as shown in the design section.
- Writing relevant code in the microcontroller (esp32 in this case).
- Writing smart contracts in the Ethereum Blockchain.
- Integrating the IoT device with the smart contract.

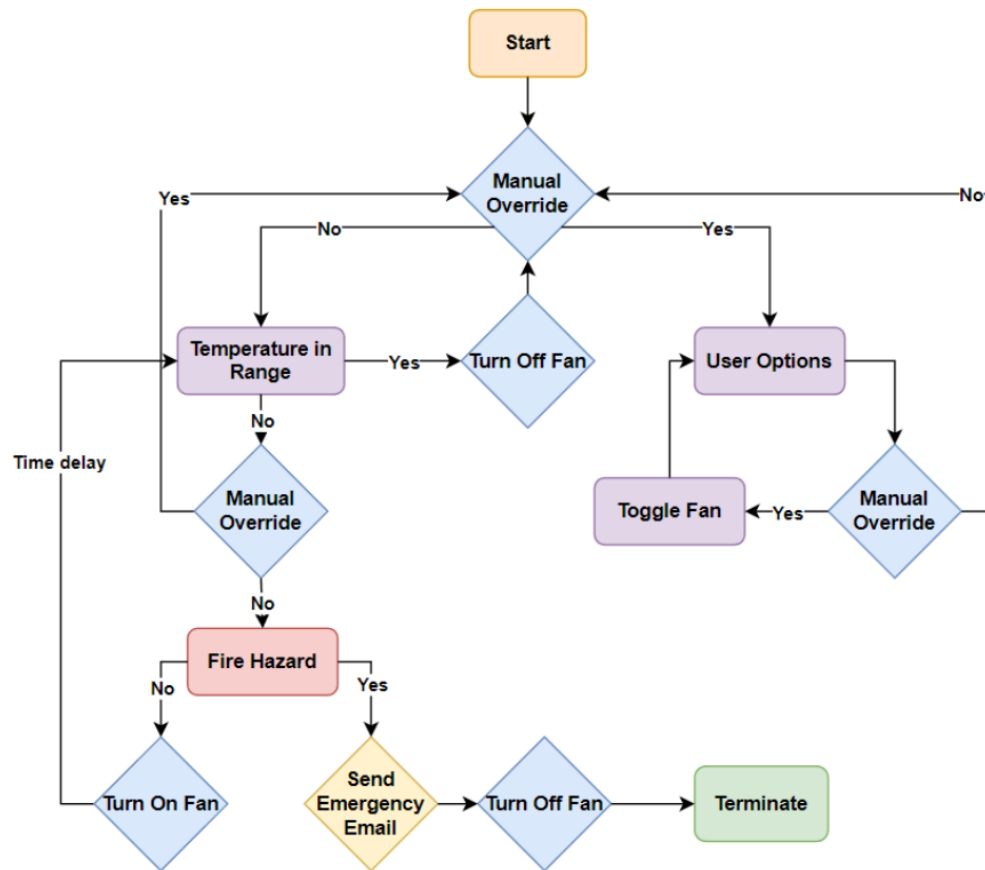
We know that IoT, when implemented in the cloud, is efficient but we can never be sure about the security aspects, to counter this problem we are using the Ethereum blockchain.

We have deployed a smart contract on the Ethereum blockchain which listens and reacts to the data sent by the esp32. The middleware used here is Infura.io as an API that connects the blockchain with the esp32, also the wallet used here is Meta Mask.

We are also using the web3-Arduino library in the Arduino IDE by which we can easily connect to the smart contract deployed.

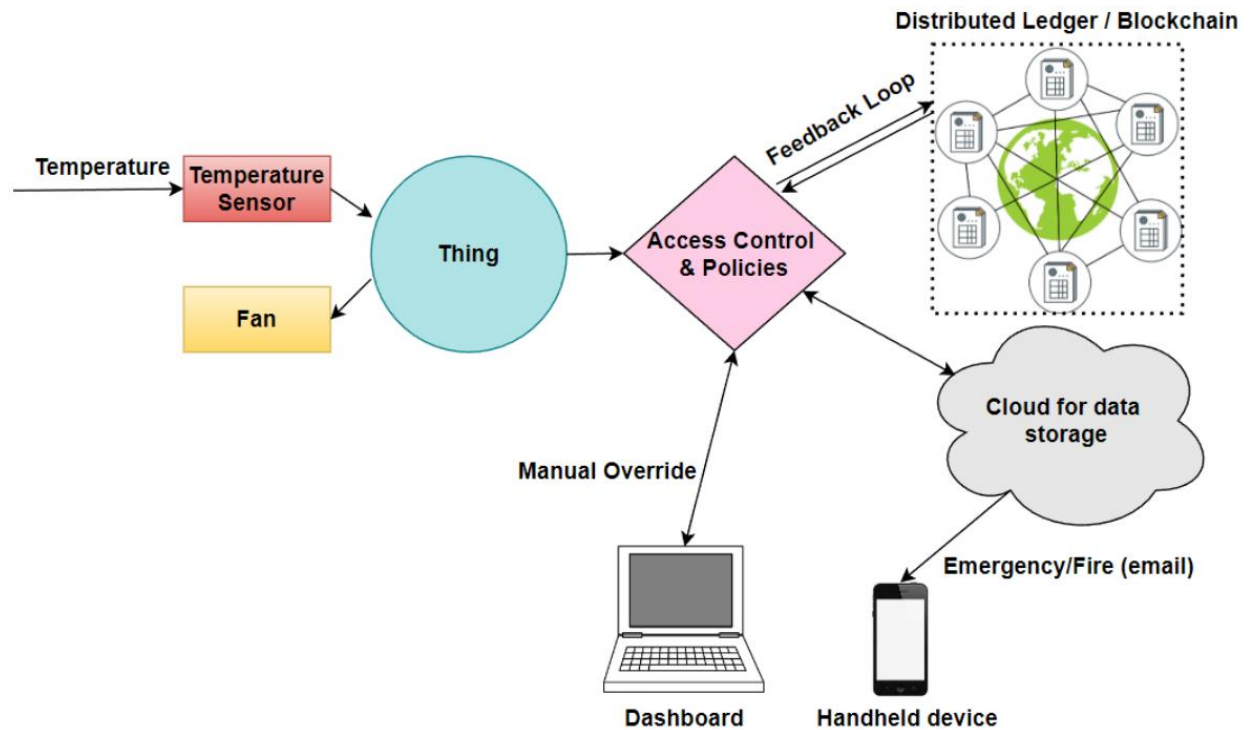
The esp32 which is now connected to the blockchain sends the temperature data periodically to the smart contract. The smart contract can then react based on that data and tell the microcontroller what to do next.

The workflow is as follows -

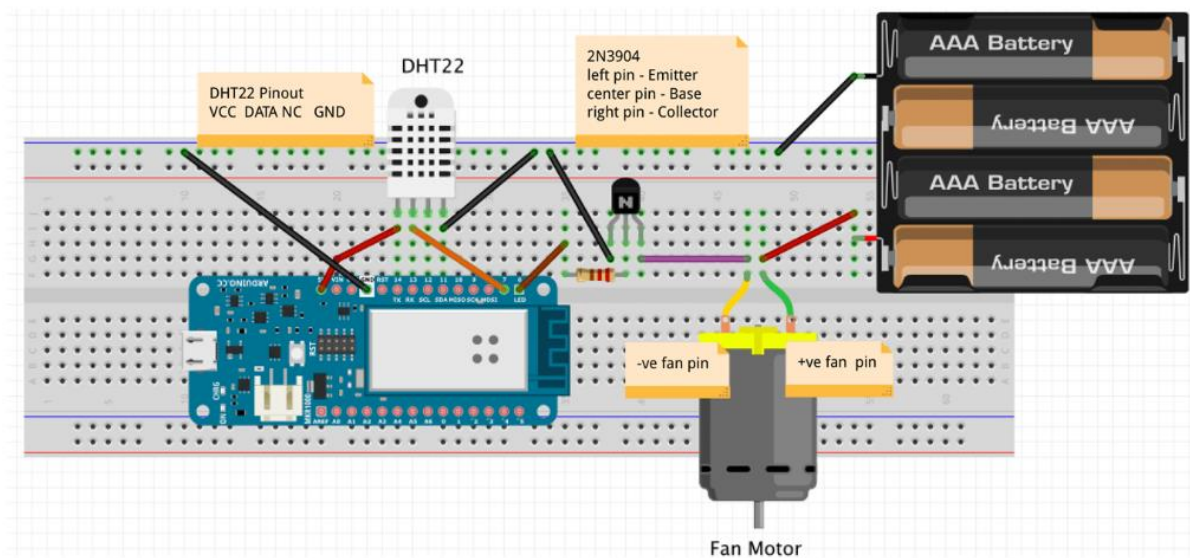


Architecture/Design -

Application Architecture -



Hardware Design -



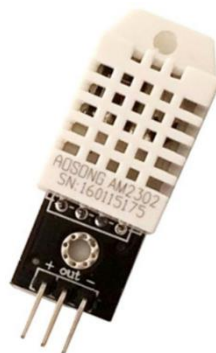
Description of Various Modules: -

Hardware Modules -

- **Esp32 microcontroller** - A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide range of applications. It is a low-cost System on Chip (SoC) Microcontroller from Espressif Systems, the developers of the famous ESP8266 SoC. It is the ESP8266 SoC's replacement and is available in single-core and comes in both single-core and 32-bit Xtensa LX6 microprocessors from Tensilica are available in dual-core versions with built-in Bluetooth and Wi-Fi.



- **DHT-22 Temperature sensor** - The DHT22 is a basic, low-cost digital temperature and humidity sensor. It measures the humidity in the air using a thermistor and a capacitive humidity sensor, and it outputs a digital signal on the data pin (no analogue input pins needed).



Software Modules/Libraries -

- **web3-arduino** - This is an Arduino (or ESP32) library to use web3 on Ethereum platform.
- **solc** – solidity compiler for node.
- **ganache-cli** - Ganache CLI uses ethereum.js to simulate full client behavior and make developing Ethereum applications faster, easier, and safer.
- **truffle** - Truffle is a world-class development environment, testing framework and asset pipeline for blockchains that make life easier for developers by utilising the Ethereum.
- **mocha** - Mocha is a widely used JavaScript test framework running on NodeJS and browsers. It supports asynchronous testing running the tests serially, allowing for more flexible and accurate reporting.

Implementation (Sample Code): -

1. To Connect to the Blynk IoT Platform -

```
#define BLYNK_TEMPLATE_ID "TMPL149d0i2I"
#define BLYNK_DEVICE_NAME "TEMPERATURE"
#define BLYNK_AUTH_TOKEN "YR8nSDJf0wtBwi2FKcsLxCzfkV6lTKF"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "braham"; // type your wifi name
char pass[] = "password"; // type your wifi password

BlynkTimer timer;

#define DHTPIN 27
#define DHTTYPE DHT11
int fanPin = 14;
DHT dht(DHTPIN, DHTTYPE);
float maxTemp = 60;
int manual = 0;
```

```

BLYNK_WRITE(V2)
{
    manual = param.asInt(); // Get value as integer
}
BLYNK_WRITE(V3)
{
    maxTemp = param.asFloat(); // Get value as integer
}
BLYNK_WRITE(V4)
{
    if (manual == 1)
    {
        int val = param.asInt(); // Get value as integer
        digitalWrite(fanPin, val);
        Blynk.virtualWrite(V4, val);
    }
}
void sendSensor()
{
    float h = dht.readHumidity();
    float t = dht.readTemperature(); // or dht.readTemperature(true) for
Fahrenheit

    if (isnan(h) || isnan(t))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    Blynk.virtualWrite(V0, t);
    Blynk.virtualWrite(V1, h);
    if (manual == 0)
    {
        if (t > maxTemp)
        {
            digitalWrite(fanPin, 1);
            Blynk.virtualWrite(V4, 1);
        }
        else
        {
            digitalWrite(fanPin, 0);
            Blynk.virtualWrite(V4, 0);
        }
    }
}

```



```

    }
}
void setup()
{
    Serial.begin(115200);
    pinMode(fanPin, OUTPUT);

    Blynk.begin(auth, ssid, pass);
    dht.begin();
    timer.setInterval(100L, sendSensor);
}

void loop()
{
    Blynk.run();
    timer.run();
}

```

2. Smart Contract written in solidity -

```

pragma solidity ^0.4.17;

contract Device {
    uint public tempLimit = 80;
    uint256 public maxTemp;
    uint256 public currTemp;
    uint256 public fan = 0;
    uint256 public panicAlarm = 0;
    constructor (uint256 temp) public {
        maxTemp = temp;
    }
    function controlLED(uint256 temp) public {
        currTemp = temp;
        if(temp > tempLimit){
            panicAlarm = 1;
            fan = 0;
            return;
        }
        if (currTemp > maxTemp) {
            fan = 1;

```

```

        } else {
            fan = 0;
        }
    }
    function setMaxTemp(uint256 temp) public {
        maxTemp = temp;
    }
}

```

3. Compiling the smart contract using JavaScript-

```

const path = require("path");
const fs = require("fs");
const solc = require("solc");

const inboxPath = path.resolve(__dirname, "contracts", "Device.sol");
const source = fs.readFileSync(inboxPath, "utf8");

module.exports = solc.compile(source, 1).contracts[":Device"];

```

4. Testing the code using mocha -

```

const assert = require("assert");
const ganache = require("ganache-cli");
const Web3 = require("web3");
const { interface, bytecode } = require("../compile.js");

const web3 = new Web3(ganache.provider());

let accounts;
let inbox;
const INITIAL_VAL = 40;
const CURR_VAL = 90;
const MAX_VAL = 70;

beforeEach(async () => {
    accounts = await web3.eth.getAccounts();

    inbox = await new web3.eth.Contract(JSON.parse(interface))

```

```

        .deploy({ data: bytecode, arguments: [INITIAL_VAL] })
        .send({ from: accounts[0], gas: "1000000" });
    });

describe("Inbox", () => {
    it("Deploys a contract", () => {
        assert.ok(inbox.options.address);
    });
    it("Has a default maxTemp", async () => {
        const temp = await inbox.methods.maxTemp().call();
        assert.equal(temp, INITIAL_VAL);
    });
    it("can change the maxTemp", async () => {
        await inbox.methods.setMaxTemp(MAX_VAL).send({ from: accounts[0] });
        const message = await inbox.methods.maxTemp().call();
        assert.equal(message, MAX_VAL);
    });
    it("can change current temperature", async () => {
        await inbox.methods.controlLED(CURR_VAL).send({ from: accounts[0] });
        const message = await inbox.methods.currTemp().call();
        assert.equal(CURR_VAL, message);
        const ledStat = await inbox.methods.fan().call();
        assert.equal(ledStat, 1);
    });
    it("can change fan mode", async () => {
        await inbox.methods.controlLED(CURR_VAL).send({ from: accounts[0] });
        const ledStat = await inbox.methods.fan().call();
        assert.equal(ledStat, 1);
    });
});

```

5. Deploying the Smart Contract -

```

const HDWalletProvider = require("@truffle/hdwallet-provider");
const Web3 = require("web3");
const { interface, bytecode } = require("./compile.js");

const provider = new HDWalletProvider(
    "girl miracle address worry prize damp motor stadium head process polar volcano",
    "https://goerli.infura.io/v3/1e598425e6a34dfdab29ef4c225fe228"
);

```

```

const web3 = new Web3(provider);

const deploy = async () => {
  const accounts = await web3.eth.getAccounts();
  console.log("Deploying From: ", accounts[0]);
  const result = await new web3.eth.Contract(JSON.parse(interface))
    .deploy({ data: bytecode, arguments: [40] })
    .send({ from: accounts[0], gas: "1000000" });
  console.log("Contract deployed to: ", result.options.address);
  provider.engine.stop();
};

deploy();

```

6. Integrating IoT with Blockchain -

```

#include <WiFi.h>
#include <Web3.h>
#include <Contract.h>

#define USE_SERIAL Serial
#define ENV_SSID      "<YOUR_SSID>"
#define ENV_WIFI_KEY "<YOUR_PASSWORD>"
#define MY_ADDRESS   "0x<MY_ADDRESS>"
#define CONTRACT_ADDRESS "0x<CONTRACT_ADDRESS>"
#define INFURA_HOST  "rinkeby.infura.io"
#define INFURA_PATH  "/<YOUR_INFURA_ID>"

const char PRIVATE_KEY[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

Web3 web3(INFURA_HOST, INFURA_PATH);

void eth_send_example();

void setup() {
  USE_SERIAL.begin(115200);
  for(uint8_t t = 4; t > 0; t--) {
    USE_SERIAL.printf("[SETUP] WAIT %d...\n", t);
    USE_SERIAL.flush();
  }
}

```

```

        delay(1000);
    }

    WiFi.begin(ENV_SSID, ENV_WIFI_KEY);

    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        // wait 1 second for re-trying
        delay(1000);
    }

    USE_SERIAL.println("Connected");
    eth_send_example();
}

void loop() {

}

void eth_send_example() {
    Contract contract(&web3, CONTRACT_ADDRESS);
    contract.SetPrivateKey((uint8_t*)PRIVATE_KEY);
    uint32_t nonceVal = (uint32_t)web3.EthGetTransactionCount((char
*)MY_ADDRESS);
    uint32_t gasPriceVal = 141006540;
    uint32_t gasLimitVal = 3000000;
    uint8_t toStr[] = CONTRACT_ADDRESS;
    uint8_t valueStr[] = "0x00";
    uint8_t dataStr[100];
    memset(dataStr, 0, 100);
    string func = "set(uint256)";
    string p = contract.SetupContractData(&func, 123);
    string result = contract.SendTransaction(nonceVal , gasPriceVal, gasLimitVal,
&toStr,      &valueStr, &p);

    USE_SERIAL.println(result);
}

```

Testing: -

1. Testing the Smart Contract using mocha –

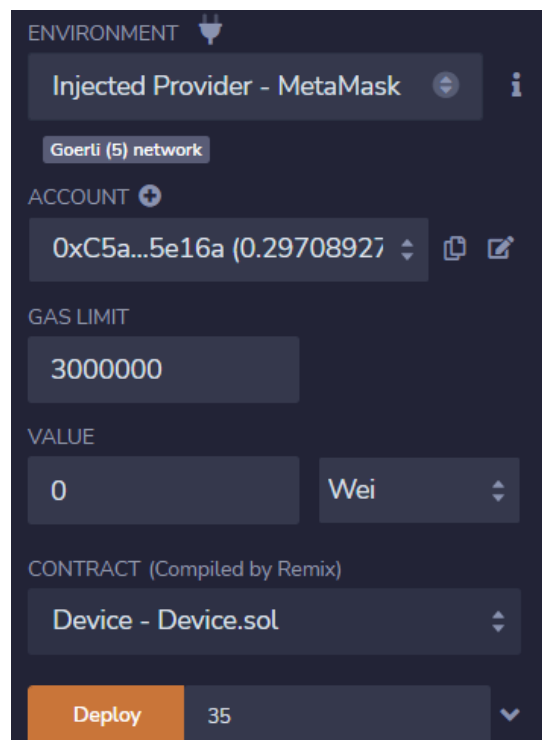
```
Dell@DESKTOP-7J8BA5J MINGW64 ~/Desktop/device
$ npx mocha
(node:3460) V8: C:\Users\Dell\Desktop\device\node_modules\solc\soljson.js:3
(Use `node --trace-warnings ...` to show where the warning was created)

Inbox
  ✓ Deploys a contract
  ✓ Has a default maxTemp (125ms)
  ✓ can change the maxTemp (300ms)
  ✓ can change current temperature (368ms)
  ✓ can change fan mode (295ms)

5 passing (3s)
```

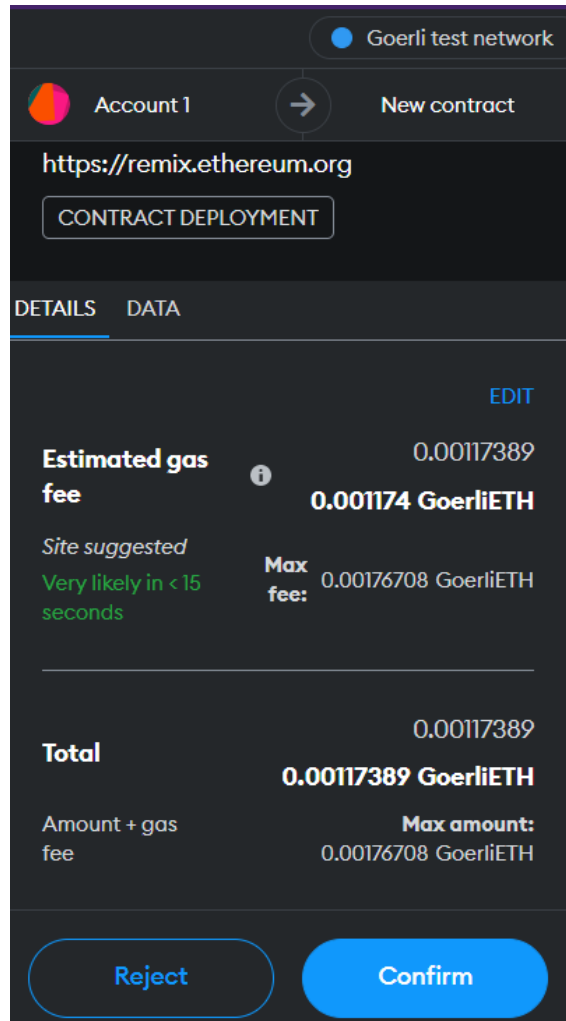
All mocha tests are passing as seen on the CLI

2. Deploying the Smart Contract using Remix –



The screenshot shows the Remix IDE's deployment panel. At the top, under 'ENVIRONMENT', 'Injected Provider - MetaMask' is selected. Below it, 'Goerli (5) network' is chosen. The 'ACCOUNT' section shows the address '0xC5a...5e16a' with a balance of '0.29708927'. The 'GAS LIMIT' is set to '3000000'. The 'VALUE' is '0' Wei. Under 'CONTRACT (Compiled by Remix)', 'Device - Device.sol' is selected. At the bottom, there is an orange 'Deploy' button and a gas price dropdown set to '35'.

Deploying The Smart Contract using MetaMask & Remix



Confirming the transaction on MetaMask

Transaction Details

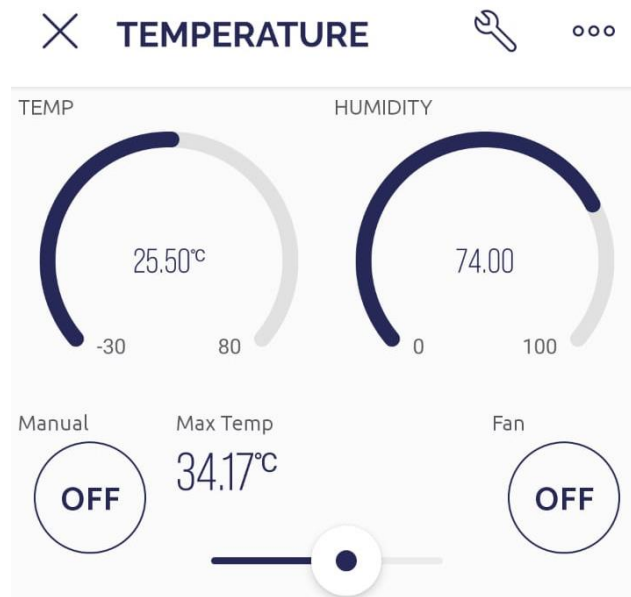
Overview State

[This is a Goerli **Testnet** transaction only]

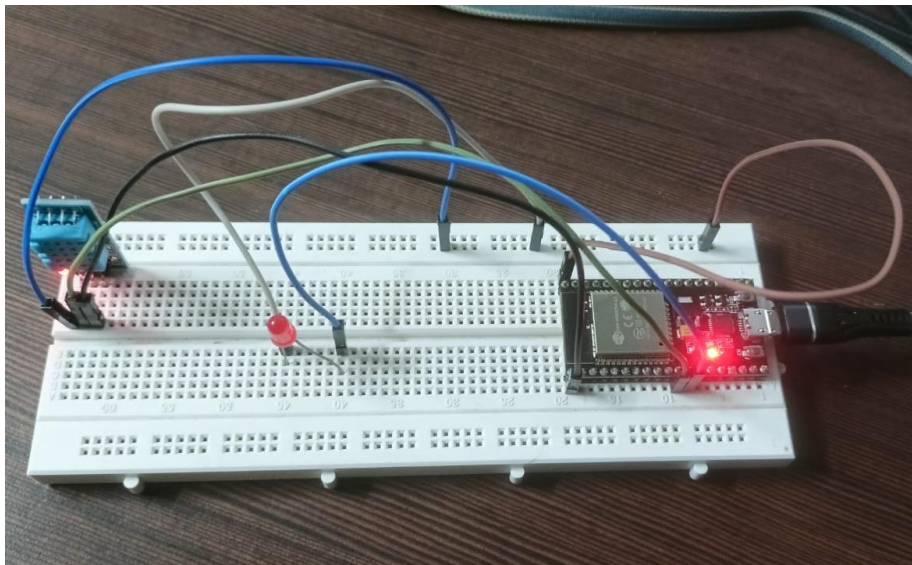
Transaction Hash:	0xa35242565e3540dcbd689d8a19554f39801a6a8101735e93e68f154a8a9261f2
Status:	Success
Block:	7953720 1 Block Confirmation
Timestamp:	9 secs ago (Nov-14-2022 08:04:00 PM +UTC)
From:	0xc5a681606b9df701a34b29c7eeb888c5cf15e16a
To:	[Contract 0xf7d8760260be054da9a079c0b1febceab50a568 Created]
Value:	0 Ether (\$0.00)
Transaction Fee:	0.001238183633327912 Ether (\$0.00)
Gas Price:	0.000000005330587928 Ether (5.330587928 Gwei)

The smart contract is deployed on the blockchain. This is verified on etherscan.

3. Testing the Application –



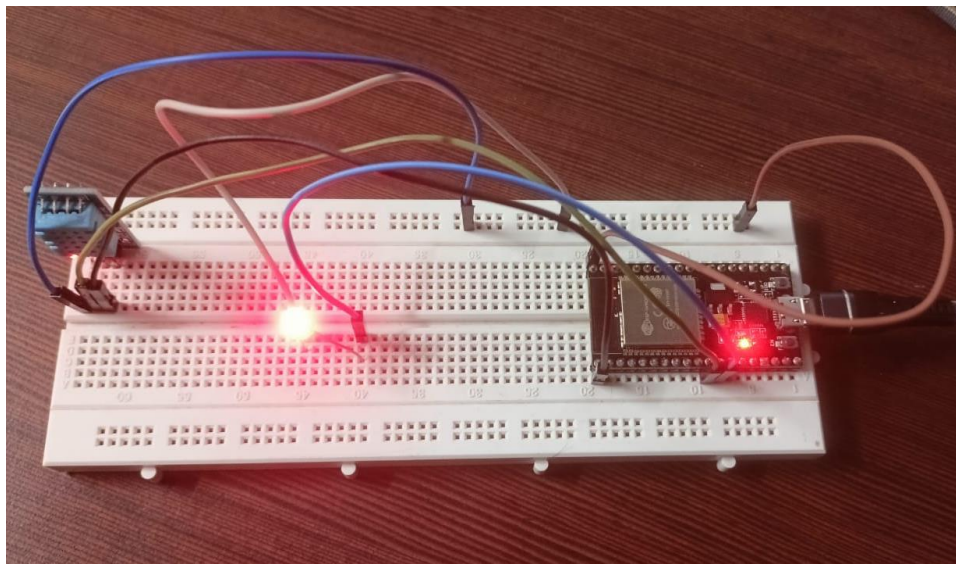
When the current temperature is lower than the max tolerable temperature.



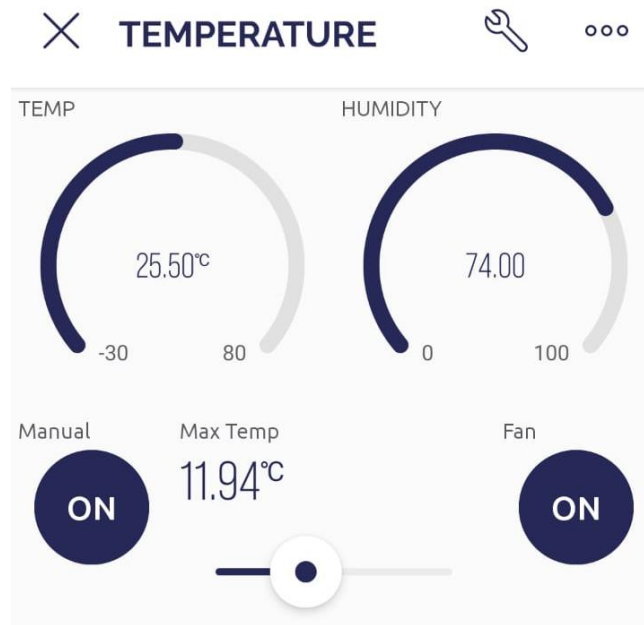
The LED is off.



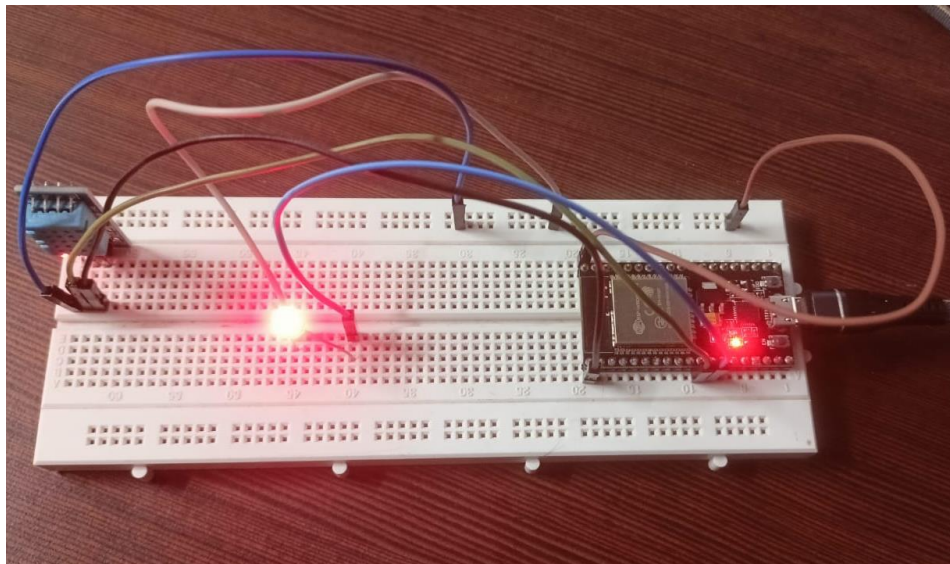
When the current temperature is higher than the max tolerable temperature.



The LED is on.



Manual mode is on, and fan is also on.



LED is on.

Conclusion:

Network security is a pending challenge for the IoT industry which is quite trending. The proposed model in this project uses Blockchain for providing secure access control to IoT devices. This application exploits the immutability feature of Blockchain by which the backend code is immutable, and no attacker can change this backend logic to toy with our application. As a whitelist is stored on Blockchain, no one can alter its contents providing better authentication and access control.

By using Blockchain instead of a traditional cloud provider we can feel secure about our data on the internet, and it makes a robust application. However, as observed even by using the Ethereum blockchain the transaction speed is quite slow and not quite adaptable for the IoT ecosystem which requires swift data transmission and policy execution, this problem certainly needs to be addressed.

Future Enhancements:

In the future, we aim to make the application more robust and efficient and try to implement a better consensus algorithm to improve efficiency, one example of doing so can be switching to a private blockchain by which transaction speed can be improved drastically.

References:

- [1]. Ghadekar, P., N. Doke, S. Kaneri, and V. Jha. "Secure access control to IoT devices using blockchain." *International Journal of Recent Technology and Engineering* 8, no. 2 (2019): 3064-3070.
- [2]. Gambo, Yaknan J., Charles I. Saidu, John A. Odey, and Joel H. Yohanna. "EMAILS AND THE DRIVE TO A SUCCESSFUL IoT SECURITY."
- [3]. Mahgoub, Asma, Nourhan Tarrad, Rana Elsherif, Abdulla Al-Ali, and Loay Ismail. "IoT-based fire alarm system." In *2019 Third World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pp. 162-166. IEEE, 2019.
- [4]. Foysal, Musfiqur Rahman, Refath Ara Hossain, Mohammad Monirul Islam, Shayla Sharmin, and Nazmun Nessa Moon. "IoT Based Temperature Control System of Home by using an Android Device." In *2021 1st International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pp. 1-8. IEEE, 2021.
- [5]. Okada, T. "Handle smart contract on Ethereum with Arduino or ESP32." (2018).
- [6]. Mohanty, Debajani. "Deploying smart contracts." In *Ethereum for Architects and Developers*, pp. 105-138. Apress, Berkeley, CA, 2018.
- [7]. Chen, Ting, Zihao Li, Hao Zhou, Jiachi Chen, Xiapu Luo, Xiaoqi Li, and Xiaosong Zhang. "Towards saving money in using smart contracts." In *2018 IEEE/ACM 40th international conference on software engineering: New ideas and emerging technologies results (ICSE-NIER)*, pp. 81-84. IEEE, 2018.
- [8]. Christidis, Konstantinos, and Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things." *Ieee Access* 4 (2016): 2292-2303.
- [9]. Zeng, Hao, Owies M. Wani, Piotr Wasylczyk, Radosław Kaczmarek, and Arri Priimagi. "Self-regulating iris based on light-actuated liquid crystal elastomer." *Advanced materials* 29, no. 30 (2017): 1701814.
- [10]. Nicol, J. Fergus, and Michael A. Humphreys. "Thermal comfort as part of a self-regulating system." (1973): 174-179.