COURSE NAME	**:** BSC (HONS) INFORMATION TECHNOLOGY (COMPUTER NETWORKING & SECURITY)
CODE	**:** NET3054
SUBJECT NAME	**:** IOT NETWORKING & SECURITY
SEMESTER	**:** 4
PROJECT TITLE	**:** INTELLIGENT LIGHTING AND TEMPERATURE CONTROL SYSTEM

GROUP NUMBER: 2
MEMBERS:

NAME	**STUDENT ID**
ELISHA NG WAN LING	23074180
NG ZHENG YU	22005102
PATT JUN-XI	23004633
SHASMEEN ALEESA BINTI SAMSURI	23092646
TAN XIAO SHI	23060304

CONTENTS

ABSTRACT

This project presents the design and implementation of a smart home automation system based on Internet of Things (IoT) technology for real-time monitoring and automated control. The system employs an Arduino Uno microcontroller integrated with a PIR motion sensor and an LM35 temperature sensor to detect human presence and ambient temperature conditions. Lighting is activated upon motion detection, while a DC motor fan is automatically controlled through a relay module when the temperature exceeds a predefined threshold. Wireless connectivity is provided by an Arduino Wi-Fi Shield, enabling periodic transmission of sensor data to the ThingSpeak cloud platform for visualization and remote monitoring. The system software is developed using a modular programming approach to manage sensor data acquisition, actuator control, and cloud communication. The implemented prototype demonstrates reliable sensor-based automation, real-time cloud monitoring, and improved energy efficiency in a smart home environment.

INTRODUCTION

With the increasing demand for energy efficiency, comfort, and safety in modern living environments, traditional home and building systems are no longer sufficient to meet current user expectations. Conventional lighting and temperature control systems often rely on manual operation, which can result in energy wastage, inconsistent environmental conditions, and reduced user convenience. In addition, the lack of real-time monitoring makes it difficult for users to track environmental conditions or device usage remotely.

The main objective of this project is to design and implement an intelligent lighting and temperature control system using Internet of Things technologies. The system's goals are to automatically modify lighting based on motion detection and temperature using sensor-based thresholds. Another objective is to enable real-time data monitoring through cloud integration, allowing users to observe system behavior remotely. The expected outcome is a functional prototype that improves comfort, increases energy efficiency, and demonstrates how IoT concepts can be applied in actual smart homes and buildings.

SYSTEM DESIGN AND ARCHITECTURE

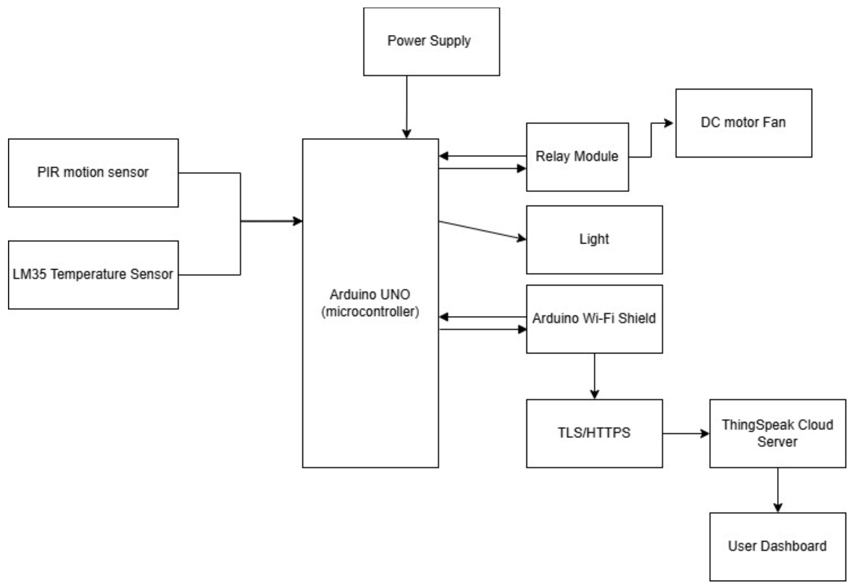


Figure 1

Hardware Description

1. Arduino UNO R3
   - It is a **central microcontroller**.
   - Reads sensor inputs, processes the data, and controls actuators (fan and lights).
   - Handles communication with the cloud server (ThingSpeak) over Wi-Fi.
2. PIR Motion Sensor
   - Detects the movement in the range.
   - Outputs a digital HIGH signal when motion is detected, LOW when no motion.
   - Help to wake up lights and record the motion events to the cloud.
3. LM35 Temperature Sensor
   - Measures ambient temperature
   - Connected to the analog input of the microcontroller.
   - DC fan ON or OFF is controlled by temperature readings based on predefined temperature limits.
4. Relay Module
   - Electronic switch to control high-voltage or current devices using low-voltage signals from the microcontroller.
   - Used only to control the DC motor fan in this system.
   - Protects the microcontroller from high current drawn by the fan.
5. DC Motor Fan
   - Controlled through the relay module.
6. LED Indicators
   - Visual feedback for motion detection and system condition.
   - Lights up when motion is detected to indicate the PIR sensor has been

7. Power Supply
    - Supplies regulated voltage to the microcontroller, sensors, and actuators.
    - Arduino UNO, 5V and 3.3V via USB.
8. Wi-Fi Module (Embedded in Shield for Arduino)
    - Connect the microcontroller (Arduino UNO) to the internet.
    - Sends sensor data to **ThingSpeak** for storage, visualization, and remote monitoring.

Communication Protocol

The Wi-Fi connection signals and HTTP requests data are transferred between the Arduino UNO and Cytron ESP8266 WiFi Shield through UART serial communication using TX and RX pins. The ESP8266 is connected to the internet through a Wi-Fi router based on the IEEE 802.11 b/g/n standards because the ESP8266 only supports the 2.4 Ghz, which is secured by WPA/WPA2 encryption. The ThingSpeak dashboard through HTTPS is secured by the cloud server TLS, the data from sensors is transferred from the ESP8266 side to the ThingSpeak cloud server through HTTP/TCPIP port 80 authenticated by the API key.

Data Flow

Step 1: Data Acquisition (Perception Layer)

- The **PIR motion sensor** detects movement and outputs a digital signal (HIGH or LOW), LED ON or OFF.
- The **LM35 temperature sensor** measures ambient temperature and outputs an analog voltage proportional to temperature.
- **Arduino microcontroller** continuously reads:
    - Digital input from the PIR sensor, Analog input from the LM35 sensor.

Step 2: Local Processing and Control

- The Arduino processes the sensor data:
    - Motion status is determined (0 = no motion, 1 = motion detected).
    - LM35 voltage output is used to measure the temperature.
- Based on predefined thresholds:
    - **Relay module** is activated to turn the **fan ON or OFF**
    - **LED indicators** are switched to reflect motion detection
- The processed values are prepared for cloud transmission.

Step 3: Data Transmission to Wi-Fi Module

- Arduino sends processed sensor data to the **Cytron ESP8266 WiFi Shield** via **UART serial communication** implemented through the **SoftwareSerial library**.

- Cytron ESP8266 WiFi shield handles:
  - Wi-Fi communication
  - HTTP request generation
- Two Arduino digital pins are configured as:
  - **TX (Transmit)** – sends data from Arduino to ESP8266
  - **RX (Receive)** – receives responses from ESP8266
- Using serial (TX/RX) communication, the Arduino UNO sends sensor and control data to the Cytron ESP8266 WiFi Shield, which then processes it to control internet and Wi-Fi access.

Step 4: Cloud Communication (Network Layer)

- Cytron ESP8266 connects to the local Wi-Fi router using **WPA2 security**.
- Sensor data is uploaded to **ThingSpeak cloud server** using:
  - **HTTP POST method**
  - ThingSpeak **Write API Key** for authentication
- Data is sent at a fixed interval (every 15 seconds).

Step 5: Cloud Processing and Storage (Application Layer)

- ThingSpeak receives the data and:
  - Authenticates the request using the API key
  - Stores data into corresponding fields:
    - Field 1 → PIR Motion
    - Field 2 → Temperature
    - Field 3 → Fan state
- ThingSpeak cloud platform processes and timestamps the data.

Step 6: Data Visualization and User Access

- Users access ThingSpeak dashboard through a web browser.
- Data is visualized as:
  - Motion detection graph
  - Temperature change graph
  - Fan ON/OFF status
- Dashboard access is protected using **HTTPS** and user authentication.

HARDWARE AND SOFTWARE IMPLEMENTATION DETAILS

The system is designed to operate utilizing an Arduino Uno, functioning as the central controller, which is connected to a laptop using a USB cable for power supply. For wireless communication, an Arduino Wi-Fi Shield (ESP8266) module is utilized, where the Wi-Fi shield connects to the Arduino using a serial connection, with the necessary pins appropriately connected based on the Wi-Fi shield layout.

An analog temperature sensor is connected to the Arduino board's analog input pin for temperature measurement. Additionally, a PIR motion sensor is connected to a digital input pin to detect any kind of motion. The relay module, LED, and DC motor fan output devices are connected to the digital output pins of the board using jumper wires and breadboards for a reliable connection.

The system firmware is developed with the Arduino IDE. Several libraries are utilized to support wireless communication and cloud interaction, including cytronWifiShield.h, cytronWifiClient.h, SoftwareSerial.h, and ThingSpeak.h.

The program is structured into functional modules to improve readability and maintainability. These modules include sensor initialization, Wi-Fi configuration, sensor data acquisition, data transmission, and timing control. During execution, the system initializes the connected sensors, establishes a Wi-Fi connection, reads sensor data, and periodically transmits the data to the cloud platform. A delay mechanism is implemented to regulate the data update interval.

Software development and debugging are performed using Arduino IDE on a laptop. The data obtained from sensors is uploaded to the ThingSpeak platform, which provides a graphical interface in the form of charts to facilitate system analysis.

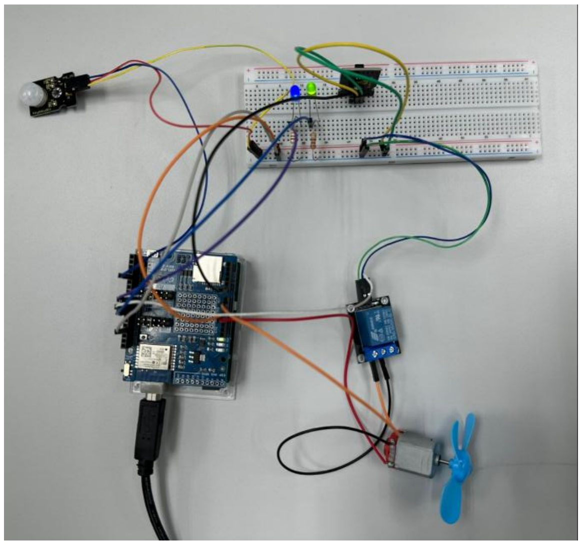


Figure 2

NETWORKING AND DATA MANAGEMENT

a. Communication Protocol (HTTP)

The method of communication between the IoT node based on Arduino and the cloud platform ThingSpeak is via Hypertext Transfer Protocol (HTTP). The main reasons for selecting this protocol are due to its simplicity, reliability and ease of implementation with the ThingSpeak cloud platform.

The Arduino acts as a client that sends readings to the ThingSpeak server by sending POST requests via HTTP (HyperText Transfer Protocol). For example, the Arduino sends a POST request for the sensor readings of temperature, light intensity, and motion status, to different fields of the ThingSpeak channel (which are defined before). The architecture of this communication model is between two machines in a client-server setup; therefore, the incoming data will be saved and processed on the cloud server.

The main communication method is from the Arduino to the server (client-to-server) as ThingSpeak does not push downwards control commands to the Arduino directly. Any control logic, such as threshold-based activation of the fan or lighting, is implemented locally on the Arduino.

b. Communication Direction and Control Logic

The protocol defined in this project for communicating between Arduino devices and the ThingSpeak cloud-based service uses both POST and GET methods but relies solely on POST requests to transfer sensor data from the Arduino device to ThingSpeak. Arduino devices can optionally use GET requests periodically to obtain configuration parameters such as threshold values from a ThingSpeak channel.

ThingSpeak does not initiate commands to any of the Arduino devices; they instead have to poll the ThingSpeak server whenever they require information, thereby creating a predicable pattern to have the devices and the server communicate, and preventing extra layers of complexity from being introduced.

Real-time decision making is done locally on the Arduino device and allows for the quickest response time possible should any delays occur in a network connection. For example, when it detects motion from the motion sensor it will turn the LED light and fan on.

c. Network Topology

The system follows the point to cloud topology using a single node. Each node consists of an Arduino sensor and actuator, and it is directly connected to the ThingSpeak cloud server using Wi-Fi and no intermediate access server or gateway is used.

This topology would be applicable in a small-scale prototype in a single room and has a few benefits:

- Lower hardware price and simplicity of the system.
- Simple installation and support.
- Direct remote access to sensor data through the cloud dashboard.

Nonetheless, this design can be optimized to offer more substantial deployments of many nodes in IoT.

d. Data Management and Graphical representation.

Sensor data is uploaded to ThingSpeak at regular time intervals of say 5-10 seconds. The data is generated in the cloud platform and stored in ordered fields relating to temperature, intensity of light, and motion status. ThingSpeak automatically creates real-time graphical visualizations, such as the line graphs that illustrate the environmental trends with time.

Local processing with the Arduino is mandatory to provide instant actuation of devices and remote monitoring, historical data analysis, and evaluation of the system performance is possible with cloud-based visualization.

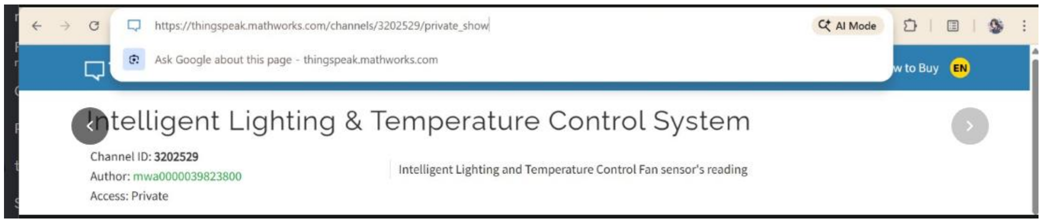
SECURITY MECHANISM

Security is a critical component of IoT systems since sensor data is sent over the internet. In this project, the Internet of Things node is an Arduino Uno. The Arduino Uno's limited processing power and memory constraints prevent it from directly handling TLS encryption and HTTPS communication. As a result, the Arduino transmits sensor data to the ThingSpeak cloud platform via HTTP.

Security is an important consideration in IoT systems due to the transmission of sensor data over the internet. In this project, the Arduino Uno is used as the IoT node. Due to its limited processing power and memory constraints, the Arduino Uno is unable to directly handle TLS encryption or HTTPS communication. As a result, sensor data is transmitted from the Arduino to the ThingSpeak cloud platform using HTTP.

Access to the ThingSpeak channel is protected using API keys, which act as a form of authentication. Only devices with the correct Write API Key are allowed to upload data to the channel, while Read API Keys control who can view the data. This prevents unauthorized users or devices from accessing or modifying the system data. To reduce the risk of credential exposure, Wi-Fi network credentials like SSID and password are not directly hard coded within the main application logic.

The use of HTTPS can be verified when accessing the ThingSpeak dashboard through a web browser, where encrypted communication is indicated by the secure connection protocol. Additionally, data uploads are accepted only when valid API keys are provided, demonstrating effective authentication and controlled access to the system.

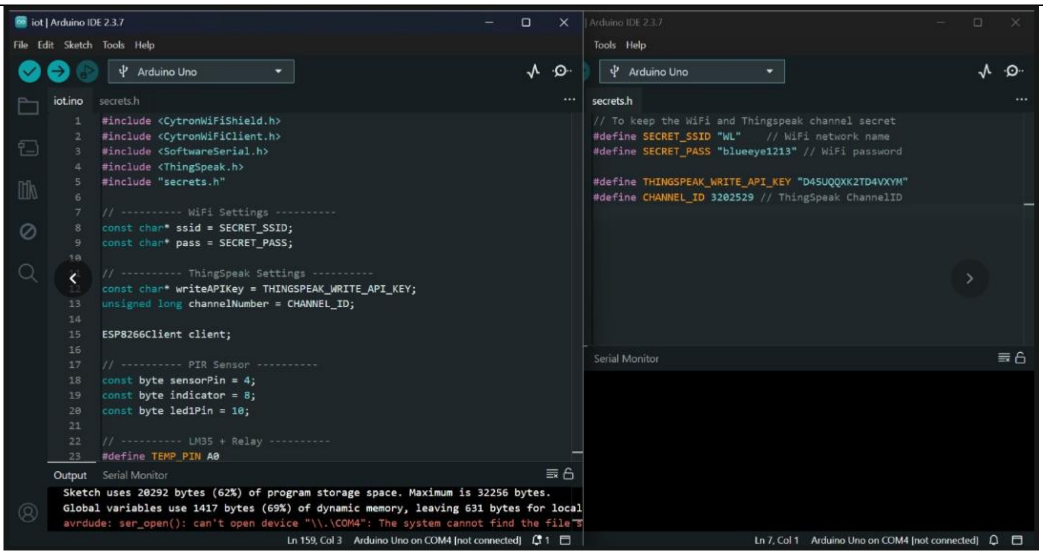

https://thingspeak.mathworks.com/channels/3202529/private_show
AI Mode
Ask Google about this page - thingspeak.mathworks.com
w to Buy
EN

Intelligent Lighting & Temperature Control System

Channel ID: 3202529
Author: mwa0000039823800
Access: Private

Intelligent Lighting and Temperature Control Fan sensor's reading

Figure 3.1



```
#include <CytronWiFiShield.h>
#include <CytronWiFiClient.h>
#include <SoftwareSerial.h>
#include <ThingSpeak.h>
#include "secrets.h"

// ---------- WiFi Settings ----------
const char* ssid = SECRET_SSID;
const char* pass = SECRET_PASS;

// ---------- ThingSpeak Settings ----------
const char* writeAPIKey = THINGSPEAK_WRITE_API_KEY;
unsigned long channelNumber = CHANNEL_ID;

ESP8266Client client;

// ---------- PIR Sensor ----------
const byte sensorPin = 4;
const byte indicator = 8;
const byte led1Pin = 10;

// ---------- LM35 + Relay ----------
#define TEMP_PIN A0
```

```
Sketch uses 20292 bytes (62%) of program storage space. Maximum is 32256 bytes.
Global variables use 1417 bytes (69%) of dynamic memory, leaving 631 bytes for local
avrdude: ser_open(): can't open device "\\.\COM4": The system cannot find the file
```

```
// To keep the WiFi and Thingspeak channel secret
#define SECRET_SSID "WL"    // WiFi network name
#define SECRET_PASS "blueeye1213" // WiFi password

#define THINGSPEAK_WRITE_API_KEY "D45UQQXK2TD4VXYM"
#define CHANNEL_ID 3202529 // ThingSpeak ChannelID
```

Figure 3.2

TESTING AND RESULTS

Testing Procedure

The smart home automation system was evaluated with the goal of verifying proper sensor execution, automated fan control, wireless data transfer, and cloud-based monitoring, the smart home automation system was assessed. The following steps made up the testing procedure:

1. **Hardware Verification**
    - Verified LM35 temperature sensor connection to the Arduino analog input.
    - Confirmed PIR motion sensor connection to the digital input pin.
    - Checked proper wiring of relay module, LED indicators, and DC motor fan to digital output pins via jumper wires and breadboards.
    - Ensured stable power supply through the USB connection to a laptop.

2. **Firmware and Software Testing**
    - Verified successful initialization of the temperature sensor, motion sensor, and Wi-Fi Shield during system startup.
    - Tested temperature and motion data acquisition to ensure accurate sensor readings.
    - Confirmed periodic transmission of sensor data to the ThingSpeak platform using the programmed delay interval.

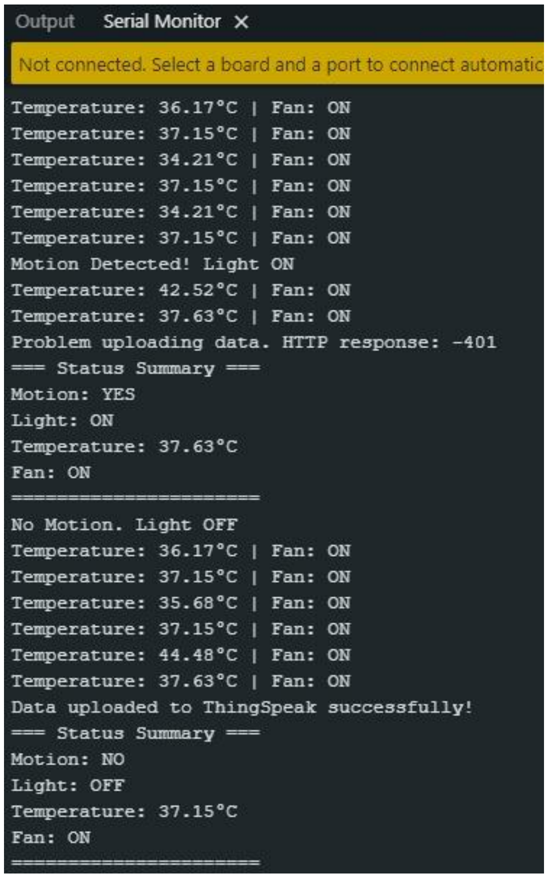
3. **Actuator Response Testing**
    - LED indicators were tested to ensure they light up upon motion detection.
    - Tested fan activation through the relay module when the temperature exceeded the predefined threshold.
    - Observed system behaviour to ensure the fan remained off when temperature values were below the threshold.

4. **Wireless Communication and Cloud Monitoring**
    - Tested Wi-Fi connectivity between the Arduino Uno and the ThingSpeak cloud platform.
    - Verified that temperature readings, motion status, and fan state were successfully uploaded and visualized in real time on the ThingSpeak dashboard.

5. **Security Verification**
    - Verified Write and Read API key authentication to ensure only authorized devices could upload or view data.
    - Confirmed HTTPS access to the ThingSpeak dashboard, ensuring encrypted communication.
    - Checked that Wi-Fi credentials were not hard coded in the firmware to prevent unauthorized access.
    - Attempted uploads and dashboard access with invalid credentials to confirm effective blocking of unauthorized actions.

6. **System Integration Testing**

- Observed the interaction between the motion sensor, temperature sensor, fan control, and cloud visualization to ensure correct system operation.
- Measured latency between sensor detection, fan activation, and data updates on the ThingSpeak platform.



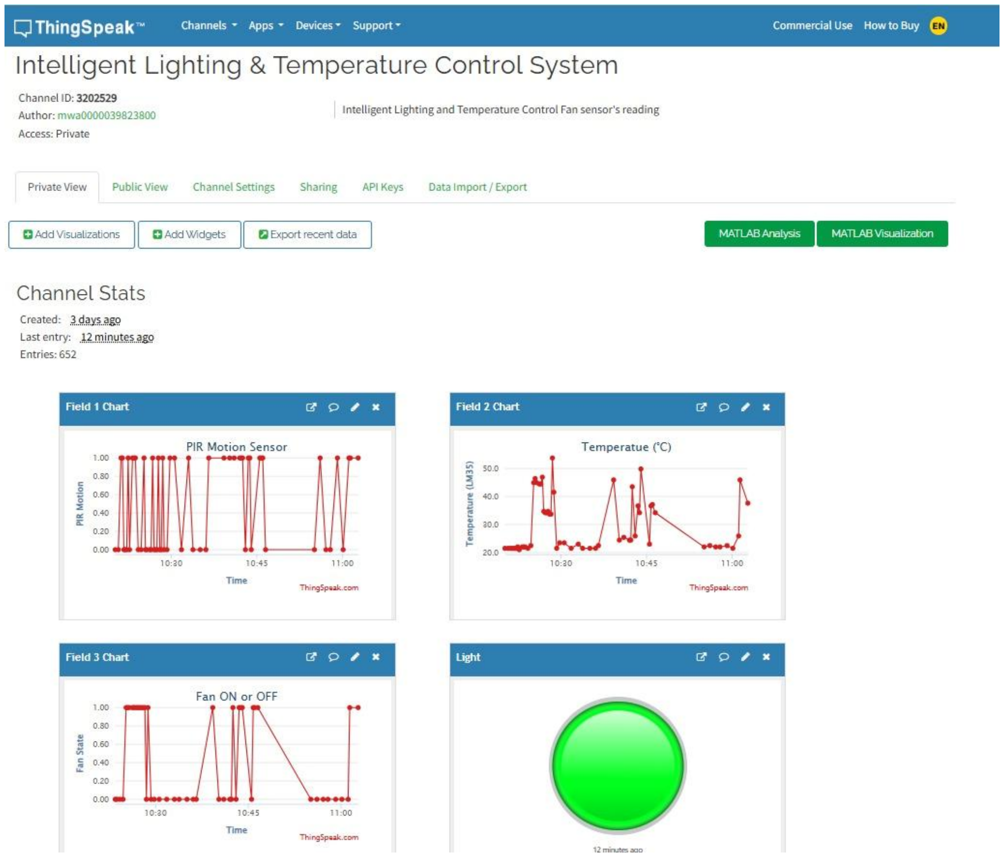
```
Not connected. Select a board and a port to connect automatic
Temperature: 36.17°C | Fan: ON
Temperature: 37.15°C | Fan: ON
Temperature: 34.21°C | Fan: ON
Temperature: 37.15°C | Fan: ON
Temperature: 34.21°C | Fan: ON
Temperature: 37.15°C | Fan: ON
Motion Detected! Light ON
Temperature: 42.52°C | Fan: ON
Temperature: 37.63°C | Fan: ON
Problem uploading data. HTTP response: -401
=== Status Summary ===
Motion: YES
Light: ON
Temperature: 37.63°C
Fan: ON
======================
No Motion. Light OFF
Temperature: 36.17°C | Fan: ON
Temperature: 37.15°C | Fan: ON
Temperature: 35.68°C | Fan: ON
Temperature: 37.15°C | Fan: ON
Temperature: 44.48°C | Fan: ON
Temperature: 37.63°C | Fan: ON
Data uploaded to ThingSpeak successfully!
=== Status Summary ===
Motion: NO
Light: OFF
Temperature: 37.15°C
Fan: ON
======================
```

Figure 4.1

Figure 4.1 shows the **Arduino IDE Serial Monitor**, which is used for debugging and verifying system operation during runtime. The serial output displays real-time temperature readings, motion detection status, and actuator states such as the fan and light. The output displayed confirms that sensor data acquisition, decision-making logic, and cloud communication are functioning as intended.

- Messages such as “Motion Detected! Light ON” and “No Motion. Light OFF” confirm correct PIR sensor functionality.
- Temperature readings and corresponding fan status indicate automated temperature control based on predefined thresholds.

- Status messages such as “Data uploaded to ThingSpeak successfully” verify successful wireless data transmission, while error messages like one that was displayed in figure 4.1 (HTTP response errors) assist in diagnosing connectivity or authentication issues.



Intelligent Lighting & Temperature Control System

Channel ID: 3202529
Author: mwa0000039823800
Access: Private

Intelligent Lighting and Temperature Control Fan sensor's reading

Channel Stats

Created: 3 days ago
Last entry: 12 minutes ago
Entries: 652

Figure 4.2

Figure 4.2 shows our **ThingSpeak cloud dashboard** used for real-time monitoring of the Intelligent Lighting and Temperature Control System. Sensor data transmitted from the Arduino Uno via the Wi-Fi Shield is visualized using multiple charts and widgets. These visualizations allow users to remotely observe environmental conditions, system activity, and actuator behavior in real time, as well as analyze historical trends.

- **Field 1** represents motion or lighting status, showing when motion is detected and lights are activated.
- **Field 2** displays temperature readings over time captured by the LM35 temperature sensor.
- **Field 3** indicates the fan state (ON/OFF) based on temperature thresholds.
- The **light indicator widget** provides a visual representation of the current lighting status.

CONCLUSION AND FUTURE WORK

We were able to design, develop and test an Intelligent Lighting and Temperature Control System based on Internet of Things (IoT) technologies. The system combines sensors, actuators, wireless networking, and cloud-based data management to show how intelligent automation can positively impact energy efficiency, user comfort, and performance convenience in interior environments. The Arduino Uno as the central controller allowed the stable acquisition of real-time data.

The project emphasizes the significance of local processing coupled with cloud-based processing. Localization of decision making on the Arduino guaranteed a real time and undivided system response even when there was a delay in the network or even a temporary loss of connection. Meanwhile, cloud integration via ThingSpeak gave centralized storage, real-time visualization, and analysis of the data so that users can view the behavior of the system remotely and assess how environment trends change with time.

The networking design that was defined using an HTTP client/server architecture was simple, dependable, effective for smaller IoT solutions, and eliminated much of the complexity from the system. While this was done to improve overall performance of the system and ensure compatibility with cloud-based services, it also provided a means of ensuring that encrypted communications were used as part of the network and supported an API-based authentication model. These aspects significantly enhanced data confidentiality and prevented unauthorized access to the system.

The current implementation is functional and has room for significant further development. For example, to support additional room or IoT nodes, the system could be further developed to incorporate a mobile or online user interface to allow for increased user interactivity and command of the system. Additional safeguards for user protection and safety like through User Authentication Dashboards may also be developed.

The system could be further enhanced through the use of AI or machine learning so that it learns how users behave and uses predictive analysis based on environmental conditions. Additionally, other sensor types such as humidity sensors, and air quality sensors could be integrated into the system so as to improve its utility for smart buildings.

In conclusion, the project fulfilled all objectives and produced a functional prototype representative of real-world smart homes/smart buildings. Additionally, the results of the project have illustrated how automation using IoT-based technology can help prevent unnecessary energy consumption, increase awareness of the environment, and assist in promoting sustainable lifestyles. This project provided us with an opportunity to gain hands-on experience in various aspects of building an IoT system including design, connectivity, security, cloud integration which serves as a solid foundation for implementing more sophisticated smart environment technologies in the future.

REFERENCES

Bruno. (2023, April 29). *WiFiEsp*. GitHub. https://github.com/bportaluri/WiFiEsp

Control a DC Motor with Arduino (Lesson #16). (n.d.). Www.youtube.com. https://www.youtube.com/watch?v=XrJ_zLWFGFw

Cytron Technologies. (2019, November 5). *Send Sensor's Data To ThingSpeak Using ESP32 [BM] #iot*. YouTube. https://www.youtube.com/watch?v=IhOyfLOwtF0

CytronTechnologies. (2021, August 20). *GitHub - CytronTechnologies/CytronWiFiShield: Arduino Library for Cytron ESP8266 WiFi Shield*. GitHub. https://github.com/CytronTechnologies/CytronWiFiShield

Global Spy Hawk. (2020, June 15). *How to connect DC motor with relay || control with Arduino*. YouTube. https://www.youtube.com/watch?v=tP8kCtdWZn4

Interfacing Arduino uno with PIR motion sensor. (n.d.). Projecthub.arduino.cc. https://projecthub.arduino.cc/electronicsfan123/interfacing-arduino-uno-with-pir-motion-sensor-593b6b

Ks0022 keyestudio LM35 Linear Temperature Sensor - Keyestudio Wiki. (2021). Keyestudio.com. https://wiki.keyestudio.com/Ks0022_keyestudio_LM35_Linear_Temperature_Sensor

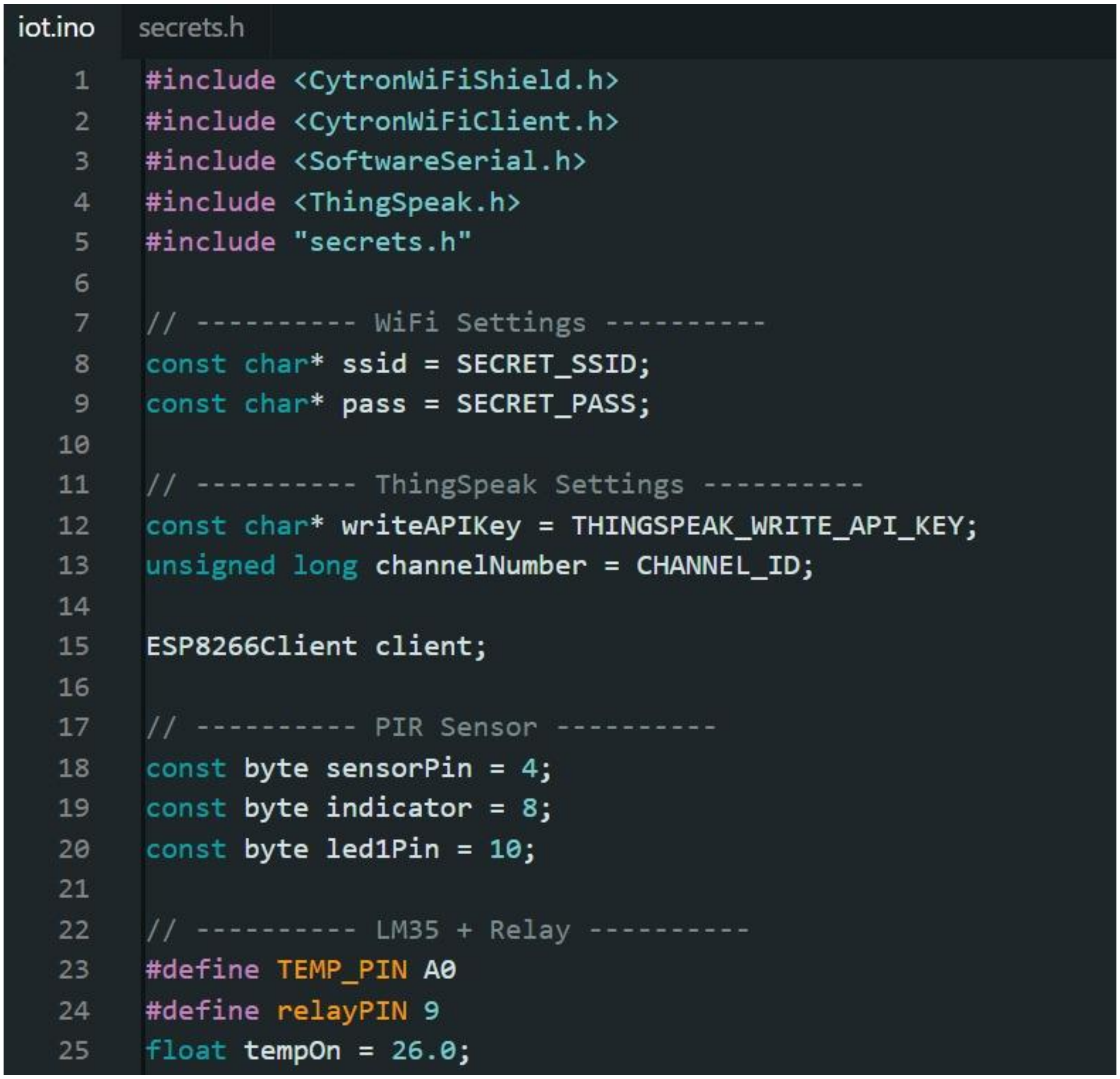
Ks0052 keyestudio PIR Motion Sensor - Keyestudio Wiki. (2021). Keyestudio.com. https://wiki.keyestudio.com/Ks0052_keyestudio_PIR_Motion_Sensor

mathworks/thingspeak-arduino. (2021, May 9). GitHub. https://github.com/mathworks/thingspeak-arduino

TechsPassion. (2024, June 27). *Using Relay Switch With Arduino*. YouTube. https://www.youtube.com/watch?v=dy9AWNs94Hs&list=PLalz_RK9TggoHwRDQjjVtwCKitKtn3r0A&index=8

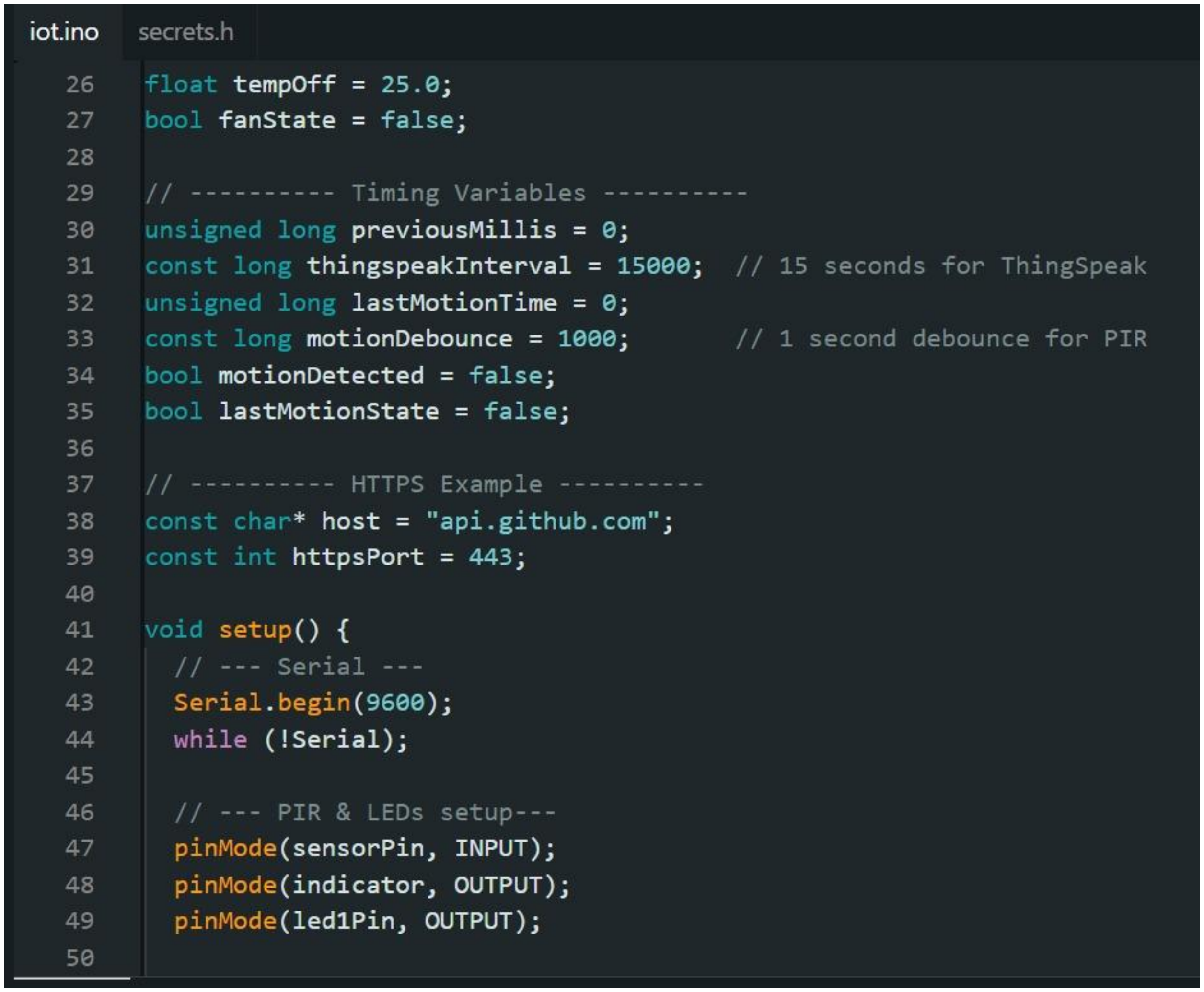
APPENDICES

Code Screenshot

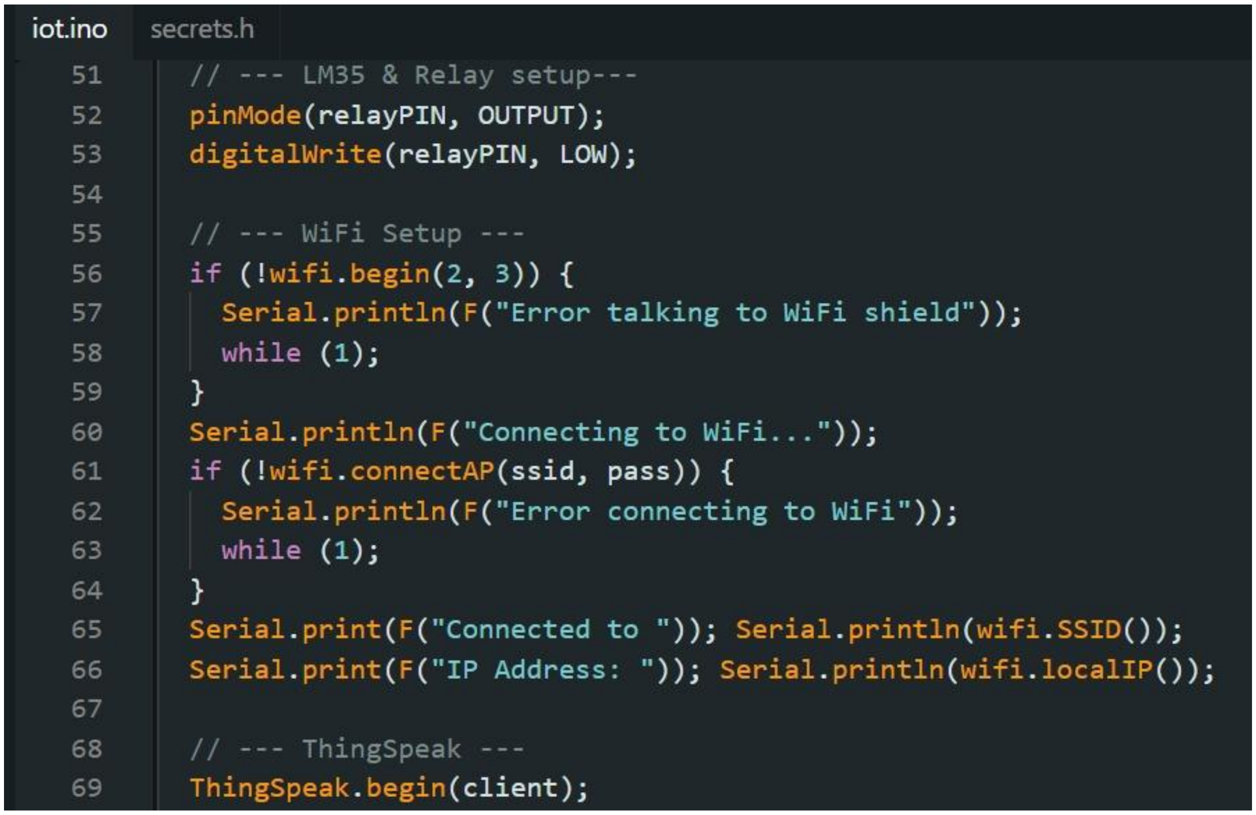


iot.ino secrets.h

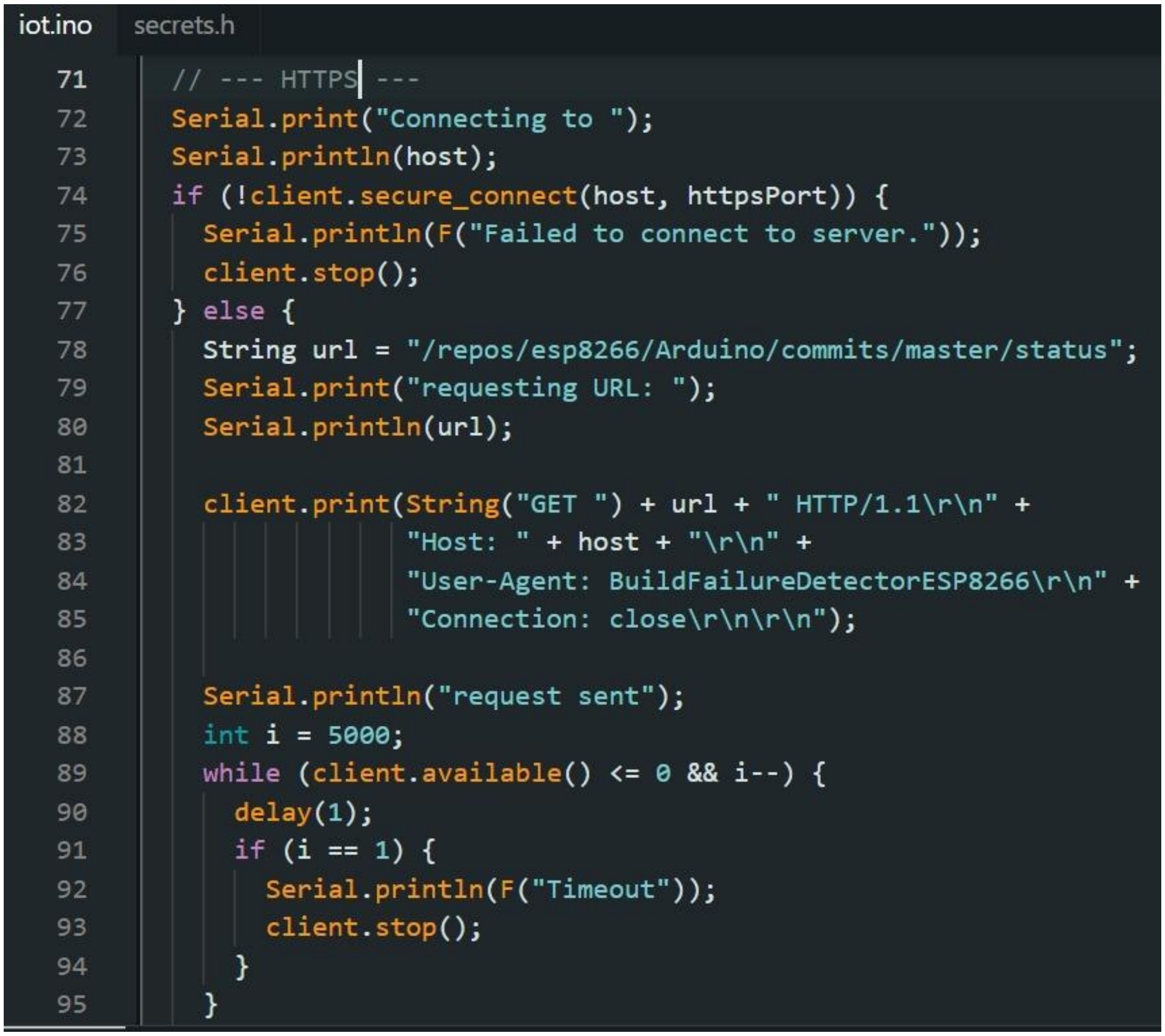
```
#include <CytronWiFiShield.h>
#include <CytronWiFiClient.h>
#include <SoftwareSerial.h>
#include <ThingSpeak.h>
#include "secrets.h"

// ---------- WiFi Settings ----------
const char* ssid = SECRET_SSID;
const char* pass = SECRET_PASS;

// ---------- ThingSpeak Settings ----------
const char* writeAPIKey = THINGSPEAK_WRITE_API_KEY;
unsigned long channelNumber = CHANNEL_ID;

ESP8266Client client;

// ---------- PIR Sensor ----------
const byte sensorPin = 4;
const byte indicator = 8;
const byte led1Pin = 10;

// ---------- LM35 + Relay ----------
#define TEMP_PIN A0
#define relayPIN 9
float tempOn = 26.0;
```


iot.ino secrets.h

```
float tempOff = 25.0;
bool fanState = false;

// ---------- Timing Variables ----------
unsigned long previousMillis = 0;
const long thingspeakInterval = 15000;  // 15 seconds for ThingSpeak
unsigned long lastMotionTime = 0;
const long motionDebounce = 1000;       // 1 second debounce for PIR
bool motionDetected = false;
bool lastMotionState = false;

// ---------- HTTPS Example ----------
const char* host = "api.github.com";
const int httpsPort = 443;

void setup() {
  // --- Serial ---
  Serial.begin(9600);
  while (!Serial);

  // --- PIR & LEDs setup---
  pinMode(sensorPin, INPUT);
  pinMode(indicator, OUTPUT);
  pinMode(led1Pin, OUTPUT);

```

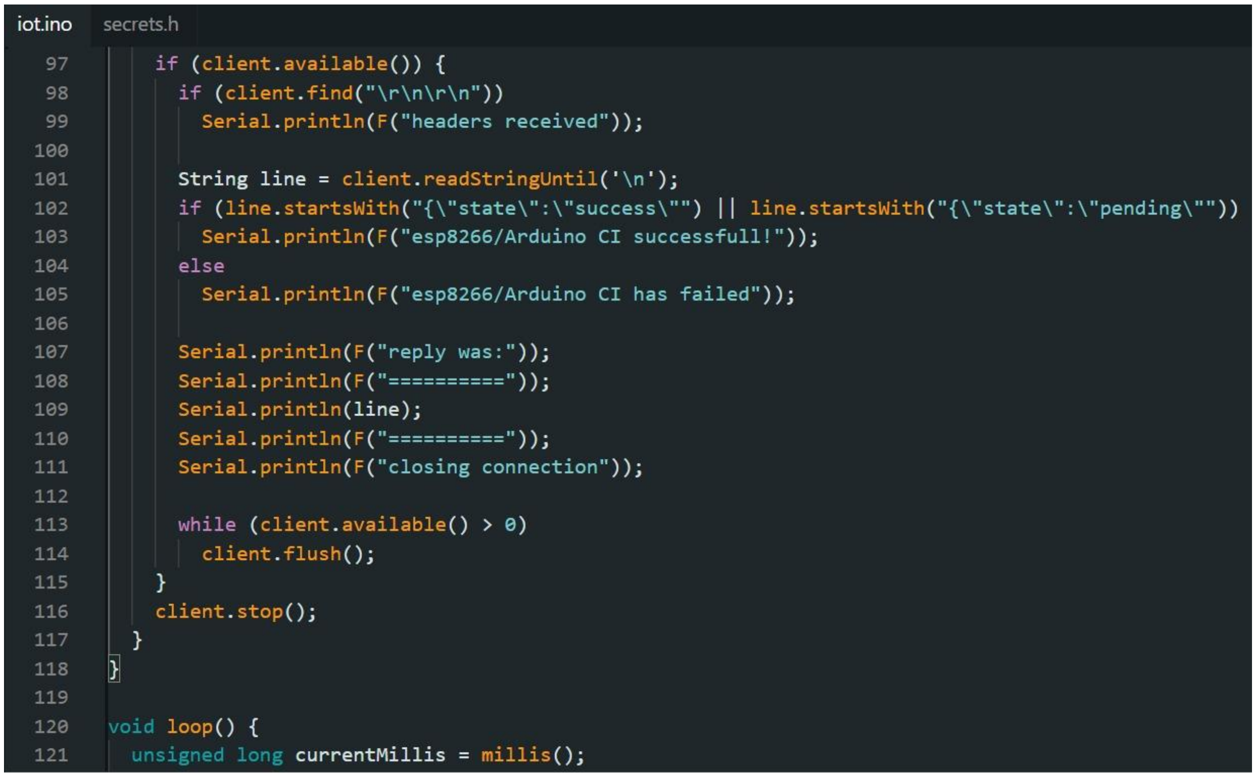

iot.ino secrets.h

```
  // --- LM35 & Relay setup---
  pinMode(relayPIN, OUTPUT);
  digitalWrite(relayPIN, LOW);

  // --- WiFi Setup ---
  if (!wifi.begin(2, 3)) {
    Serial.println(F("Error talking to WiFi shield"));
    while (1);
  }
  Serial.println(F("Connecting to WiFi..."));
  if (!wifi.connectAP(ssid, pass)) {
    Serial.println(F("Error connecting to WiFi"));
    while (1);
  }
  Serial.print(F("Connected to ")); Serial.println(wifi.SSID());
  Serial.print(F("IP Address: ")); Serial.println(wifi.localIP());

  // --- ThingSpeak ---
  ThingSpeak.begin(client);
```

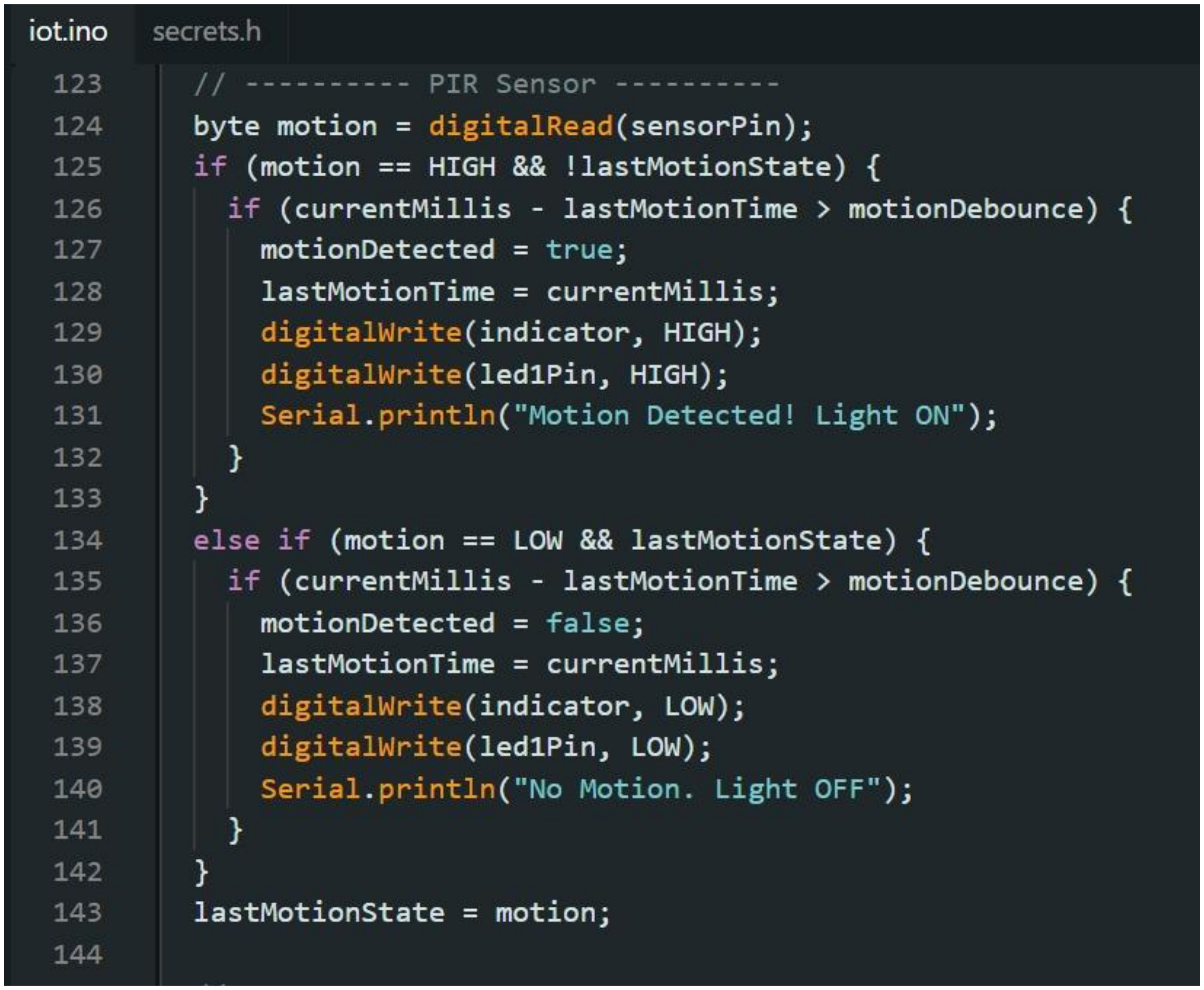


iot.ino secrets.h

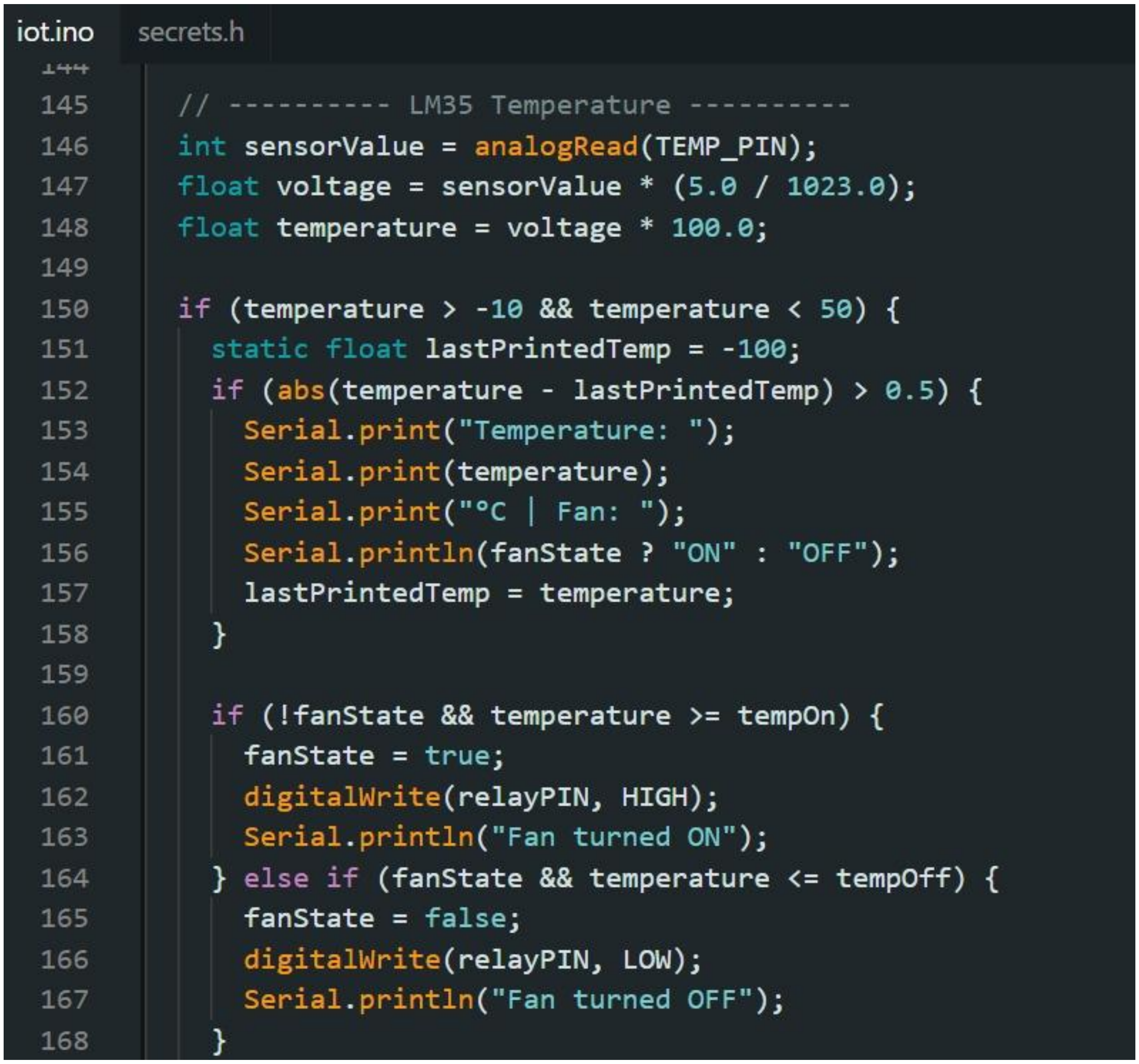
```
  // --- HTTPS ---
  Serial.print("Connecting to ");
  Serial.println(host);
  if (!client.secure_connect(host, httpsPort)) {
    Serial.println(F("Failed to connect to server."));
    client.stop();
  } else {
    String url = "/repos/esp8266/Arduino/commits/master/status";
    Serial.print("requesting URL: ");
    Serial.println(url);

    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "User-Agent: BuildFailureDetectorESP8266\r\n" +
                 "Connection: close\r\n\r\n");

    Serial.println("request sent");
    int i = 5000;
    while (client.available() <= 0 && i--) {
      delay(1);
      if (i == 1) {
        Serial.println(F("Timeout"));
        client.stop();
      }
    }
```



```
    if (client.available()) {
      if (client.find("\r\n\r\n"))
        Serial.println(F("headers received"));

      String line = client.readStringUntil('\n');
      if (line.startsWith("{\"state\":\"success\"") || line.startsWith("{\"state\":\"pending\""))
        Serial.println(F("esp8266/Arduino CI successfull!"));
      else
        Serial.println(F("esp8266/Arduino CI has failed"));

      Serial.println(F("reply was:"));
      Serial.println(F("=========="));
      Serial.println(line);
      Serial.println(F("=========="));
      Serial.println(F("closing connection"));

      while (client.available() > 0)
        client.flush();
    }
    client.stop();
  }
}

void loop() {
  unsigned long currentMillis = millis();
```

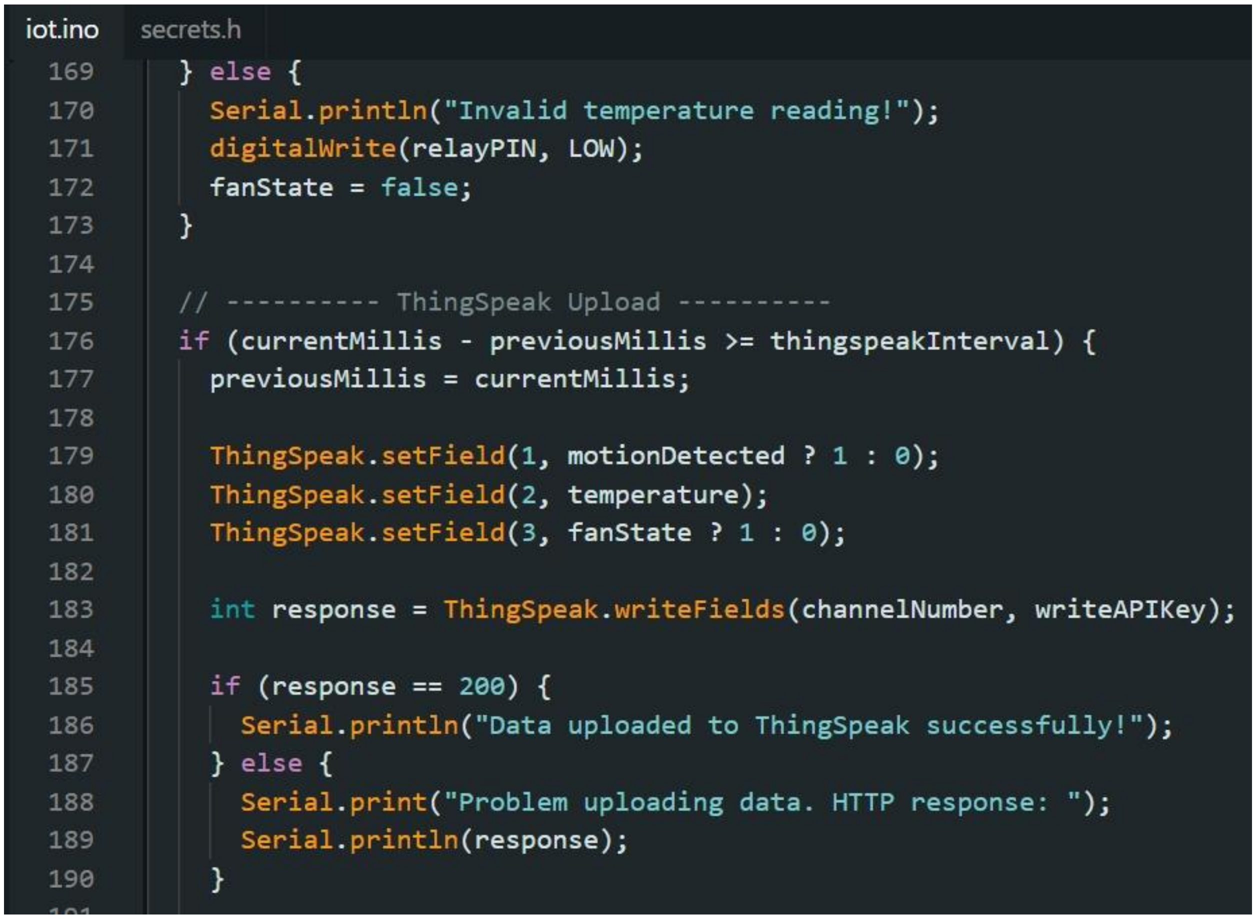


```
// ---------- PIR Sensor ----------
byte motion = digitalRead(sensorPin);
if (motion == HIGH && !lastMotionState) {
  if (currentMillis - lastMotionTime > motionDebounce) {
    motionDetected = true;
    lastMotionTime = currentMillis;
    digitalWrite(indicator, HIGH);
    digitalWrite(led1Pin, HIGH);
    Serial.println("Motion Detected! Light ON");
  }
}
else if (motion == LOW && lastMotionState) {
  if (currentMillis - lastMotionTime > motionDebounce) {
    motionDetected = false;
    lastMotionTime = currentMillis;
    digitalWrite(indicator, LOW);
    digitalWrite(led1Pin, LOW);
    Serial.println("No Motion. Light OFF");
  }
}
lastMotionState = motion;

```

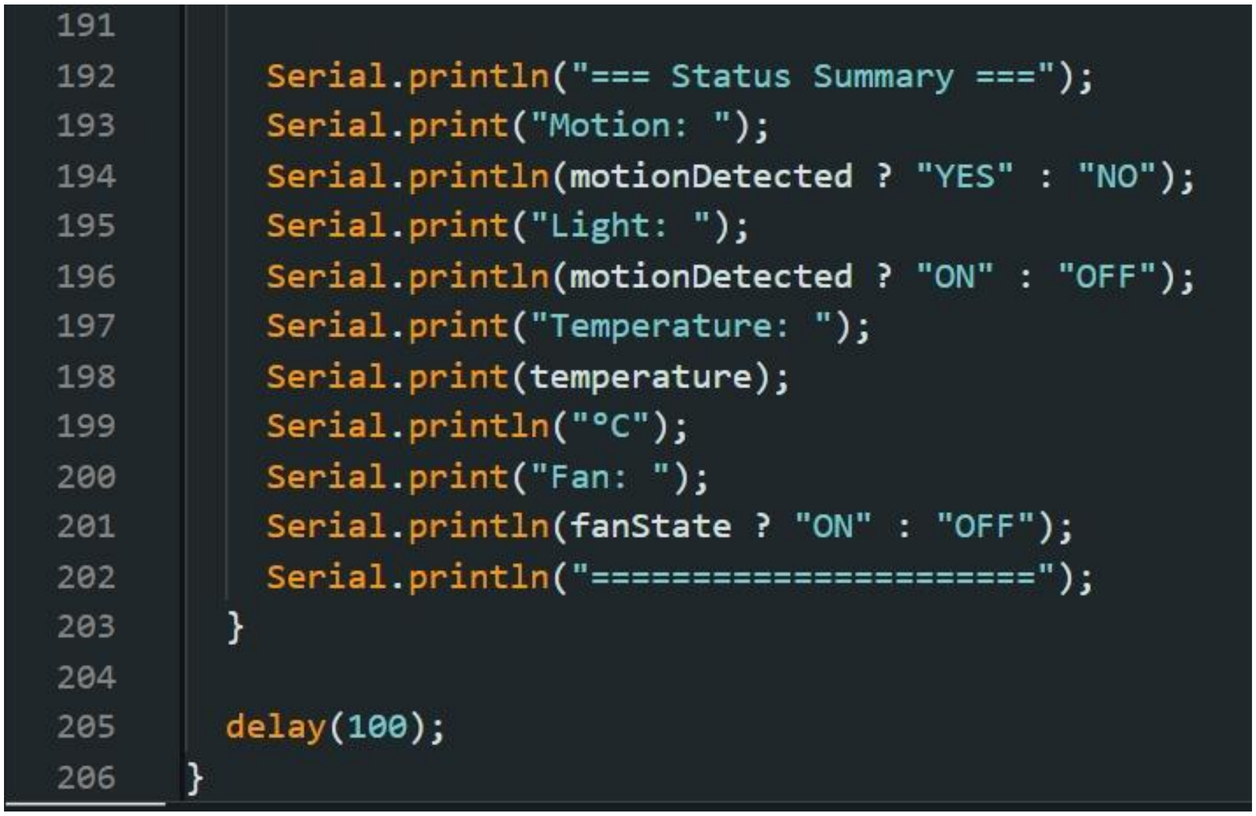


iot.ino secrets.h

```
  // ---------- LM35 Temperature ----------
  int sensorValue = analogRead(TEMP_PIN);
  float voltage = sensorValue * (5.0 / 1023.0);
  float temperature = voltage * 100.0;

  if (temperature > -10 && temperature < 50) {
    static float lastPrintedTemp = -100;
    if (abs(temperature - lastPrintedTemp) > 0.5) {
      Serial.print("Temperature: ");
      Serial.print(temperature);
      Serial.print("°C | Fan: ");
      Serial.println(fanState ? "ON" : "OFF");
      lastPrintedTemp = temperature;
    }

    if (!fanState && temperature >= tempOn) {
      fanState = true;
      digitalWrite(relayPIN, HIGH);
      Serial.println("Fan turned ON");
    } else if (fanState && temperature <= tempOff) {
      fanState = false;
      digitalWrite(relayPIN, LOW);
      Serial.println("Fan turned OFF");
    }
```

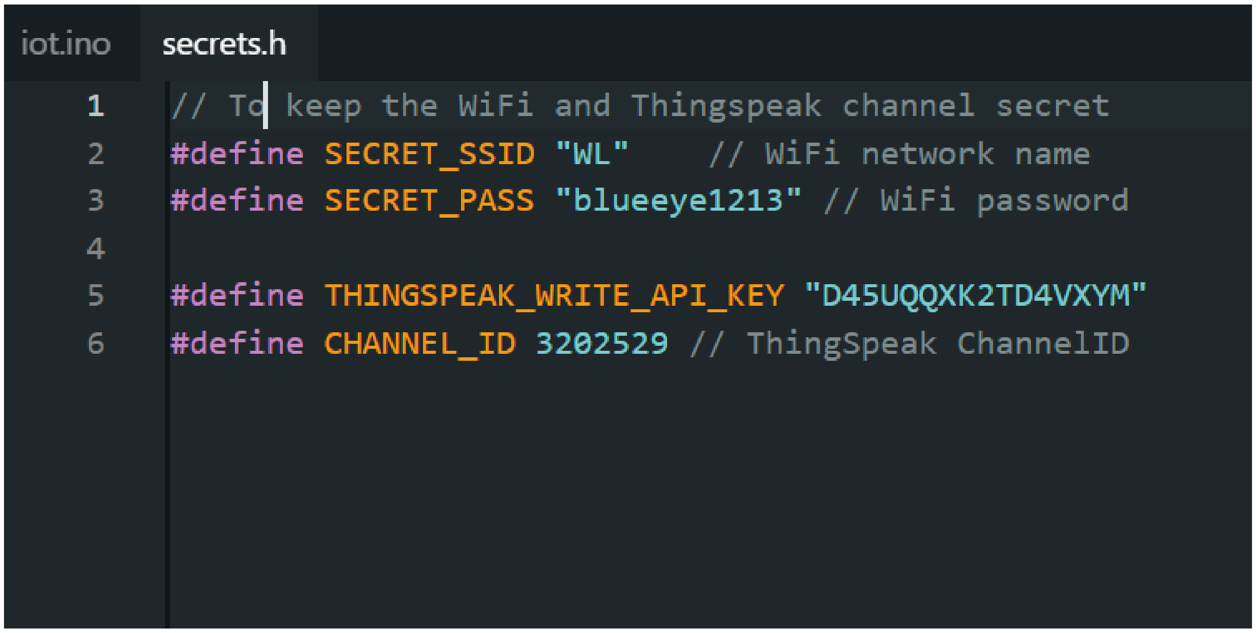


```
  } else {
    Serial.println("Invalid temperature reading!");
    digitalWrite(relayPIN, LOW);
    fanState = false;
  }

  // ---------- ThingSpeak Upload ----------
  if (currentMillis - previousMillis >= thingspeakInterval) {
    previousMillis = currentMillis;

    ThingSpeak.setField(1, motionDetected ? 1 : 0);
    ThingSpeak.setField(2, temperature);
    ThingSpeak.setField(3, fanState ? 1 : 0);

    int response = ThingSpeak.writeFields(channelNumber, writeAPIKey);

    if (response == 200) {
      Serial.println("Data uploaded to ThingSpeak successfully!");
    } else {
      Serial.print("Problem uploading data. HTTP response: ");
      Serial.println(response);
    }
```



```

    Serial.println("=== Status Summary ===");
    Serial.print("Motion: ");
    Serial.println(motionDetected ? "YES" : "NO");
    Serial.print("Light: ");
    Serial.println(motionDetected ? "ON" : "OFF");
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println("°C");
    Serial.print("Fan: ");
    Serial.println(fanState ? "ON" : "OFF");
    Serial.println("======================");
  }

  delay(100);
}
```

Configuration Files


iot.ino
secrets.h

```
// To keep the WiFi and Thingspeak channel secret
#define SECRET_SSID "WL"     // WiFi network name
#define SECRET_PASS "blueeye1213" // WiFi password

#define THINGSPEAK_WRITE_API_KEY "D45UQQXK2TD4VXYM"
#define CHANNEL_ID 3202529 // ThingSpeak ChannelID
```