IoT-Based Thermal Data Logging And User Identification

Piyush Pattanayak

Institute of Mathematics and Applications
Andharua, Bhubaneswar- 751029
Odisha

IoT-Based Thermal Data Logging And User Identification

A Project Report submitted to the
Institute of Mathematics and Applications, Bhubaneswar
in partial fulfilment of the requirements
of the degree of

B. Sc. (Hons.) in Mathematics and Computing

by

Piyush Pattanayak
(7473U216013)

under the supervision of

Rashmirani Panda
Supervisor

and

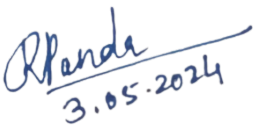
Arijit Ghosh
Co-Supervisor

Institute of Mathematics and Applications
Andharua, Bhubaneswar -751029
Odisha

Institute of Mathematics and Applications
Andharua, Bhubaneswar- 751029, Odisha.

Supervisor's Certificate

This is to certify that the work presented in this project entitled *"IoT-Based Thermal Data Logging and User Identification*" by *Piyush Pattanayak* (7473*U*216013) is a record of original research/ review work carried out by him under our supervision and guidance in partial fulfilment of the requirements for the degree of B. Sc. (Hons.) in Mathematics and Computing. Neither this project nor any part of it has been submitted for any degree or diploma to any institute or university in India or abroad.



Rashmirani Panda
(*Supervisor*)

Arijit Ghosh
(*Co-Supervisor*)

Dedicated to

My Mother

Declaration

I, *Piyush Pattanayak*, 7473*U*216013 hereby declare that this project entitled *"IoT based Thermal Data logging and User Identification"* represents my original work carried out as a B. Sc. (Hons.) student of Institute of Mathematics and Applications, Bhubaneswar and to the best of my knowledge, it is not a complete copy of previously published or written by another person, nor any material presented for award of any other degree or diploma of IMA, Bhubaneswar or any other institution. Any contribution made to this research by others, with whom I have worked at IMA, Bhubaneswar or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the section Bibliography.

Piyush Pattanayak

Acknowledgment

I would like to warmly acknowledge and express my deep sense of gratitude and indebtedness to my supervisor Mrs. Rashmirani Panda, Institute of Mathematics and Applications, Bhubaneswar, for her guidance, encouragement and suggestions while preparing this dissertation.

I would also like to thank my co-supervisor Mr. Arijit Ghosh of IMA, Bhubaneswar for his research input, constant support and help in completing this project in time.

I would like to thank my supervisor and her research collaborators Mr. Chinmayananda Padhy and Dr. Sudhakar Sahoo for introducing me the problem and given me the opportunity to work and explore new things in this area of research. I would also like to thank the other faculty members of IMA, Bhubaneswar for their constant motivation during my study.

My heartfelt thanks to all our friends, especially Nirmalya Chakraborti and Subhojit Roy, for their cooperation and inspiration during my dissertation work. I would not have completed this dissertation without their help and support.

My higher education was a dream of my respective parents. My B.Sc. program would not have been materialized without their love and moral support. How can I express in words how much I owe to them.

Piyush Pattanayak

University Rollno. 7473U216013

Abstract

In this project, we present an Internet of Things (IoT) based solution for real-time monitoring of RFID tags and temperature data. The system utilizes an ESP32 microcontroller interfaced with an RFID reader and a temperature sensor. The RFID reader identifies unique tags and retrieves associated names of individual, while the temperature sensor measures the individual's temperature. Data collected from the ESP32 is transmitted to a Google sheet via a Google Apps Script, providing seamless data logging and visualization capabilities.

The system architecture comprises hardware components including an ESP32 microcontroller, an RFID reader module (Adafruit PN532) and a digital temperature sensor (Adafruit MLX90614). The ESP32 is programmed to read RFID tags and temperature values periodically. Upon detection of an RFID tag, the corresponding name is retrieved from a predefined mapping stored within the Arduino IDE sketch. The collected data, including timestamp, UID, name and temperature, is transmitted to a designated Google sheet via a secure HTTP connection established by the Google Apps Script.

The Google Apps Script acts as an intermediary between the Arduino IDE and the Google Sheet. It handles incoming HTTP requests, parses the data payload and appends the data to the spreadsheet in real-time. Additionally, it performs a lookup operation to map the RFID UIDs to their associated names before appending the data. This integration enables centralized data storage, analysis and visualization, facilitating easy monitoring and tracking of RFID tag movements and environmental temperature changes over time.

Overall, the proposed IoT-based solution offers a cost-effective and scalable approach for RFID and temperature monitoring, leveraging the capabilities of Arduino, RFID technology and cloud-based data storage and processing provided by Google Sheets.

Contents

Chapter 1

Introduction

In the era of rapid technological advancement, the Internet of Things (IoT) has emerged as a transformative paradigm, revolutionizing the way we interact with the world around us by enabling seamless connectivity and intelligent interactions between physical devices and digital systems. This convergence of technology has unlocked unprecedented opportunities for innovation across various domains, from smart homes and industrial automation to healthcare and transportation. At core of IoT ecosystem, lies the ability to collect, analyze and act upon data generated by a myriad of interconnected sensors and devices, empowering individuals and organizations to make informed decisions, optimize processes and enhance quality of life.

The objective of this project is to explore the potential of IoT technology in enhancing security and automation in everyday environments through the integration of an MLX sensor and an RFID card reader. By leveraging the capabilities of these sensors and combining them with IoT principles, we aim to develop a robust and intelligent system that can monitor environmental conditions, detect intrusions and enable personalized access control, thereby enhancing safety, security and convenience in residential and commercial settings.

The MLX sensor, known for its accuracy and reliability in measuring ambient temperature and humidity, serves as a critical component of the system, providing real-time environmental data that can be leveraged for various applications, including climate control, energy management and asset monitoring. By integrating the MLX sensor into our IoT solution, we can capture valuable insights into the indoor environment, identify trends and trigger automated responses to optimize comfort and efficiency [1].

Complementing the MLX sensor, the RFID card reader adds an additional layer of security and convenience by enabling seamless authentication and access control. Through the use of RFID-enabled cards or tags, authorized users can gain entry to secured areas with ease, while unauthorized access attempts can be promptly detected and reported [2].

Our project aims to leverage IoT sensors, specifically the MLX sensor for thermal data, an RFID sensor for identification and the ESP32 microcontroller, to develop a robust system for logging user thermal data and accurately associating it with their identities.

In case of any health ailment the first changes that happens in the human body is the change in body temperature. At the earliest or initial stage if it is detected and checked, then a lot many people can be protected and the affected or infected person can be treated without any delay. The body temperature of the accessed entry person is captured with the help of MLX sensor thereby a slight difference in the temperature from normal level can be traced, taken into account and diagnosed of ailment is any.

The integration of these sensors into an IoT framework enables real-time monitoring, analysis and control of environmental conditions and access events, providing stakeholders with valuable insights and actionable intelligence to make informed decisions and respond effectively to changing conditions.

In the subsequent sections of this report, we will delve into the design, implementation and evaluation of our IoT solution, detailing the architecture, components, functionalities and performance metrics. Through rigorous experimentation and testing, we aim to demonstrate the effectiveness and practicality of our device in addressing real-world challenges and delivering tangible benefits to end-users.

In the following chapters, we excavate into the fundamentals of IoT technology, explore its applications in different sectors, analyze the challenges and opportunities it presents and propose strategies for addressing them. Chapter 2 is being provided with the various hardware devices, equipments and IDE platform required in our project work. Chapter 3 installs the Algorithm and the working procedure. Chapter 4 incorporates the Arduino IDE platform and the programming code that has been used in our project, the output has also been shown here. Our project report ends at Chapter 5, with conclusion and future work.

Chapter 2

Devices and Modules

Prerequisites

2.1 Microcontroller

A microcontroller is a small integrated circuit that has peripherals, memory, and a central processing unit. It functions as embedded systems' brain. There are three types computer architectures: 8-bit, 16-bit, and 32-bit, each with varying processing power and capabilities. Example: ESP32, Atmel AVR, PIC, and ARM Cortex-M series microcontrollers. In our project, We use ESP32 microcontroller and it's description is given below.

2.2 ESP32

ESP32 is a low-cost, low-power microcontroller chip developed by Espressif Systems. It has integrated Wi-Fi and Bluetooth connectivity and provides many GPIO pins that facilitate connection and control of external devices and sensors.

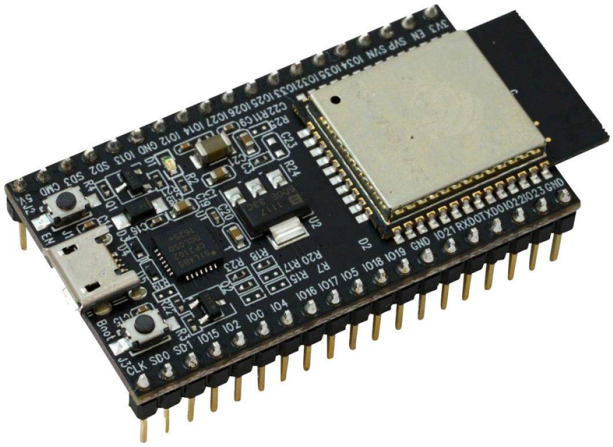


Figure 2.1: ESP32

2.2.1 Specification:

- **Processor:** Dual-core Tensilica LX6 microprocessor, clocked at up to 240 MHz.
- **Memory:**
  Internal SRAM: Up to 520 KB.
  External Flash Memory: Supports up to 16 MB of external SPI flash memory.
- **Wi-Fi:** Integrated 802.11 b/g/n Wi-Fi transceiver. Supports station, soft access point, and Wi-Fi Direct modes.
- **Security:** WPA/WPA2/WPA3.
- **Bluetooth:** Integrated Bluetooth 4.2 BR/EDR and BLE (Bluetooth Low Energy) radio. Supports Bluetooth Classic and Bluetooth Low Energy.
- **Peripherals:** GPIO (General Purpose Input/Output) pins. SPI (Serial Peripheral Interface) interface. I2C (Inter-Integrated Circuit) interface. UART (Universal Asynchronous Receiver/Transmitter) interface. ADC (Analog to Digital Converter) channels. DAC (Digital to Analog Converter) channels. Touch sensor inputs. Operating Voltage: Typically operates at 3.3V.
- **Operating Temperature:** -40°C to 85°C.
- **Power Consumption:** Varies depending on the usage scenario and power-saving features enabled, but generally low power consumption.
- **Operating System:** It can be programmed using the Arduino IDE, ESP-IDF (Espressif IoT Development Framework), MicroPython, or other development platforms.
- **Dimensions:** The ESP32 chips come in various packages including QFN, SOP, and others, with different pin configurations and sizes.

2.2.2 Applications:

- Smart Home
- Industrial automation
- Health Care
- Smart Agriculture
- Audio Devices
- Image Recognition
- Speech Recognition

2.2.3 Connection:

The USB connection between a laptop and an ESP32 entails linking the microcontroller to the laptop using a micro-USB . This connection enables tasks such as uploading code, debugging, and establishing serial communication. Development environments like Arduino IDE facilitate programming and interaction, allowing users to streamline the process of coding and testing applications for the ESP32 microcontroller.

Figure 2.2: ESP32-Laptop

2.3 MLX90614

The MLX90614 is an infrared thermometer for non-contact temperature measurements. The MLX90614 is a Contactless Infrared (IR) Digital Temperature Sensor that can be used to measure the temperature of a particular object ranging from -70° C to 382.2°C. The sensor uses IR rays to measure the temperature of the object without any physical contact and communicates to the microcontroller using the I2C protocol.

[1] [4]

Figure 2.3: MLX90614

2.3.1 Specification:

1. Operating Voltage: 3.6V to 5V (available in 3V and 5V version)
2. Supply Current: 1.5mA

3. Object Temperature Range: -70° C to 382.2°C
4. Ambient Temperature Range: -40° C to 125°C
5. Accuracy: 0.02°C
6. Field of View: 80°
7. Distance between object and sensor: 2cm-5cm (approx.)

2.3.2 Applications of MLX90614:

1. Temperature Measurement of moving objects
2. Industrial Thermal Gun
3. Human Body Temperature Measurement
4. Home/Office Temperature Control
5. Livestock Monitoring
6. Movement Detection

2.3.3 Connection:

Pin Outs:

- SDA: The serial data pin, which communicates with the controller
- SCL: The serial clock pin, which connects to the Arduino's I2C clock line
- VCC: The power pin, which can be connected to 3V3 (3.3V) or V5 (5V) pin of ESP32.
- GND: The ground pin which is connected to the GND pin of ESP32.

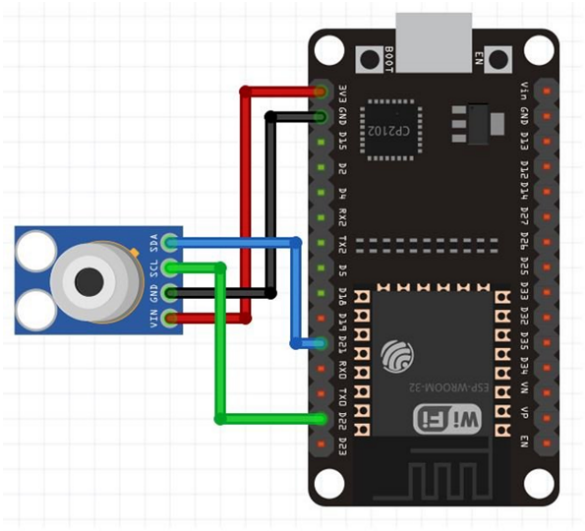


Figure 2.4: MLX90614-ESP32

To connect an MLX90614 infrared temperature sensor to an ESP32, wire the sensor's SDA pin to the ESP32's GPIO pin configured for I2C data (typically GPIO21), connect the SCL pin to the ESP32's GPIO pin configured for I2C clock (usually GPIO22), and power the sensor using the ESP32's 3.3V pin.

2.4 RFID-PN532

The PN532 is a highly integrated NFC (Near Field Communication) controller from NXP Semiconductors. RFID (Radio Frequency Identification) is a technology that allows for the wireless identification of physical objects using radio waves. The PN532 supports various RFID protocols, including ISO/IEC 14443 Type A/B, FeliCa, and ISO/IEC 18092 (NFC). It is commonly used in applications such as access control systems, contactless payment systems, and electronic ticketing. [8]

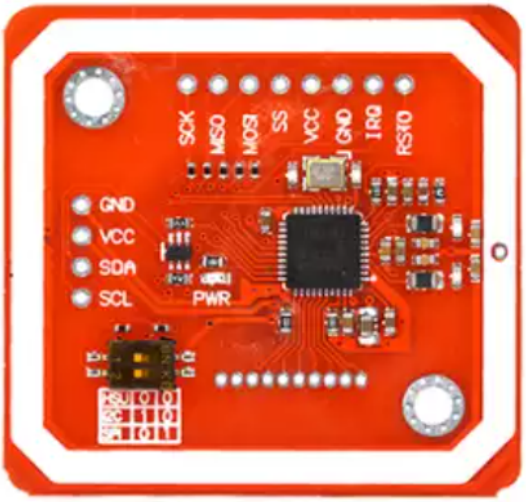


Figure 2.5: RFID PN532

Before understanding PN532 as a whole, let's learn what is NFC technology and how does it work.

2.4.1 What is NFC?

Near Field Communication(NFC) is a technology standard based on Radio Frequency Identification (RFID), transmitting information wirelessly over short distances.

NFC technology makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. NFC is compatible with hundreds of millions of contactless cards and readers already deployed worldwide.

2.4.2 How NFC works?

NFC operates on the principle of inductive coupling, at least for short-range implementations. This essentially involves the reader device generating a magnetic field by passing an electric current through a coil. When a tag (with its own coil) is brought nearby, the field in-

duces an electric current within the tag — sans any wires or even physical contact. Then, once the initial handshake is complete, any stored data on the tag is wirelessly transmitted to the reader.

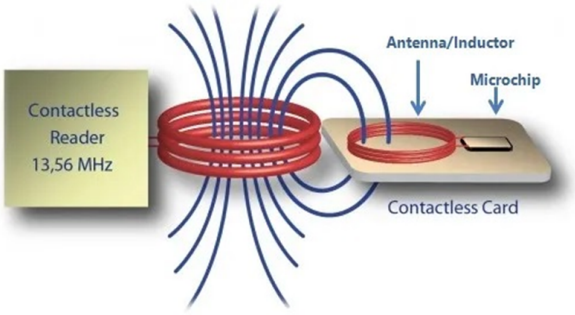


Figure 2.6: NFC

2.4.3 Main applications of NFC

- Mobile payments; Apple Pay NFC, Google Wallet
- Tap-to-Pair; Pairing Bluetooth devices just by tapping, instead of manual entering
- Embedding digital experience in physical products; NFC's small form factor allows for it to be embedded into physical products, enabling a unique experience when interacted
- Security
- Password replacement
- Product Authentication

2.4.4 PN532 NFC Module

PN532 is an NFC controller by NXP . It allows for contactless communication at 13.56 MHz. Furthermore, the support for MIFARE Classic 1K/MIFARE Classic 4K card allows for higher transfer speeds up to 424 kbit/s in both directions. It has 40 Kb ROM and 1Kb RAM. It is used to emulate ISO14443 cards.

2.4.5 Features

- 80C51 microcontroller core with 40 KB ROM and 1 KB RAM
- Highly integrated demodulator and decoder
- Integrated RF level detector
- Supports ISO/IEC 14443A/MIFARE
- Supports ISO/IEC 14443B (Reader/Writer mode only)
- Up to 50mm of operating distance in Reader/Writer mode for communication to ISO/IEC 14443A/MIFARE, ISO/IEC 14443B, or FeliCa cards
- Up to 50mm of operating distance in NFCIP-1 depending on antenna size, tuning, and power supply

- Approximately 100mm of operating distance in ISO/IEC 14443A/MIFARE or FeliCa card emulation mode
- Communication distance: 5cm to 7cm
- Low power modes
  - Hard-Power-Down mode
  - Soft-Power-Down mode
- Programmable timers
- Crystal oscillator
- 2.7 to 5.5 V power supply operating range

2.4.6 Why we choose PN532 NFC MODULE?

The Big advantage of the PN532 NFC module is that it can use different protocols to communicate with Arduino such as UART, I2C or SPI. These different protocols use specific pins and libraries of the microcontroller.

2.4.7 What are RFID Card reader and Tags?

RFID Tags: Each individual who needs to be tracked carries an RFID tag. These tags can be in the form of cards, key fobs, wristbands, or even embedded into items like ID badges or smartphones. Each tag contains a unique identifier that is associated with the individual it belongs to.

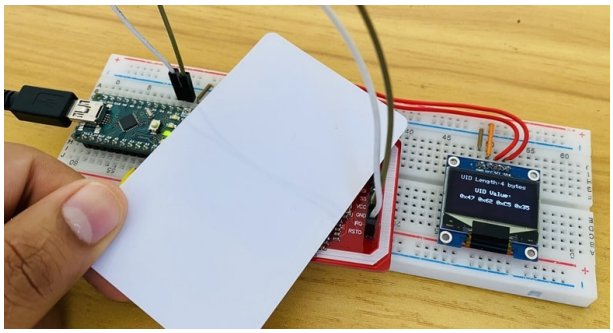


Figure 2.7: RFID Card

RFID Readers: RFID readers are deployed at strategic locations where attendance needs to be recorded, such as entry points or classrooms. These readers emit radio waves and detect RFID tags within their proximity. When a tag comes into range of a reader, it captures the unique identifier stored on the tag.

2.4.8 Connection:

To connect an RFID PN532 module with an ESP32 via SCL and SDA we have to link PN532's SCL to ESP32's SCL pin, SDA to ESP32's SDA pin, GND to GND, and VCC to ESP32's 3.3V pin. Then by enabling I2C by including the Wire library and initializing it with

Wire.begin(SDA, SCL). We utilize a library like Adafruit PN532 to interact with the module, specifying its I2C address. In setup(), initialize the PN532 module and configure it to read RFID tags. In loop(), wait for NFC tags and retrieve their UID data.

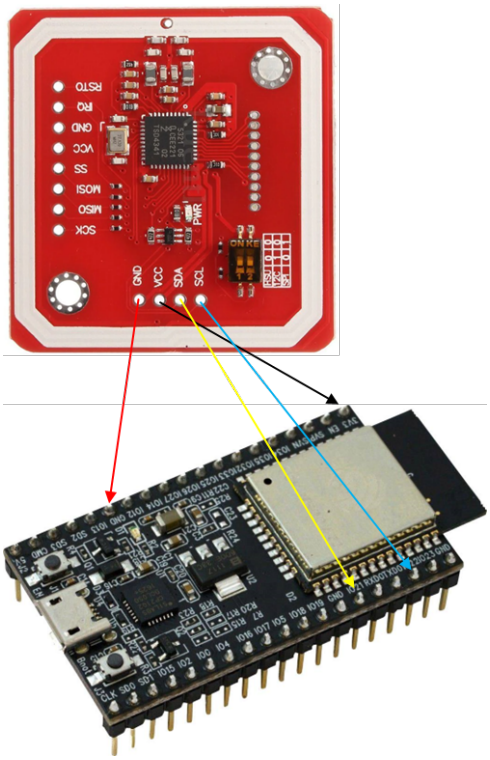


Figure 2.8: RFID-ESP32

2.5 OLED

The SSD1106 is a popular OLED (Organic Light-Emitting Diode) display controller chip manufactured by Solomon Systech. It is widely used in various electronic devices, especially in wearable devices, IoT (Internet of Things) gadgets, and DIY electronics projects.

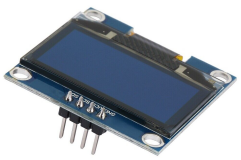


Figure 2.9: OLED(SSD1106)

2.5.1 Specifications:

- **Resolution:** Common resolutions are 128x64 or 128x32 pixels, but variants with different resolutions may also be available.

- **Interface:** Supports communication via standard interfaces like I2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface).
- **Colour Depth:** Monochrome display controller, typically supporting one-bit color depth (black and white).
- **Power Consumption:** Low power consumption due to OLED technology, especially advantageous for battery-powered devices.
- **Operating Temperature:** Wide operating temperature range, suitable for various environments.
- **Driver Support:** Various software libraries and drivers available for interfacing with the SSD1106, facilitating ease of integration into different platforms and programming languages.
- **Dimensions:** Compact size suitable for small-scale applications where space is limited.
- **Compatibility:** Compatible with a range of OLED display sizes and configurations, providing flexibility in design implementation.

2.5.2 Applications:

- **Wearable Devices:** SSD1106 OLED displays are used in smartwatches, fitness trackers, and other wearable gadgets due to their compact size, low power consumption, and vibrant display quality.
- **IoT (Internet of Things) Devices:** These displays are commonly integrated into IoT devices such as smart home controllers, weather stations, and environmental sensors for providing real-time data visualization.
- **Consumer Electronics:** SSD1106 OLED displays can be found in devices like digital cameras, portable media players, and remote controls, enhancing user interfaces and displaying information in a clear and readable format.
- **Medical and Healthcare Devices:** SSD1106 displays are used in medical devices such as pulse oximeters, blood glucose meters, and patient monitoring systems to display vital signs and health-related data.
- **Gaming Devices:** SSD1106 OLED displays are used in handheld gaming consoles, game controllers, and gaming accessories for displaying game graphics, scores, and interactive elements.

2.5.3 Connection:

We will use the i2C communication protocol to connect the SSD1306 OLED display to the ESP32.

Pin-outs:

VIN Pin: Connect to ESP32's 3V3/V5 pin

GND Pin: Connect to GND pin of ESP32

SCL Pin: Connect to GPIO 22 pin of ESP32

SDA Pin: Connect to GPIO 21 pin of ESP32

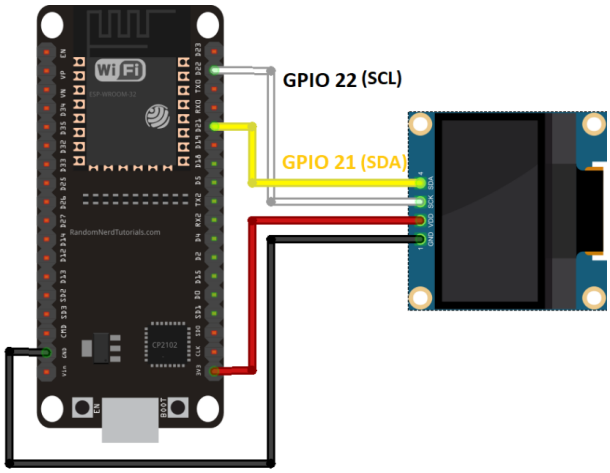


Figure 2.10: Oled-ESP32

2.6 Buzzer

A buzzer or beeper is an audio signaling device. The most common uses of buzzers and beepers include alarm devices, timers, train and confirmation of user input such as a mouse click or keystroke.

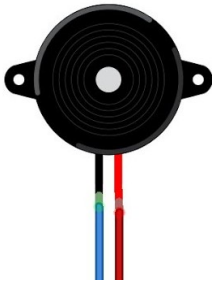


Figure 2.11: Buzzer

2.6.1 Specification:

- Rated voltage: 6V DC, but operating voltage ranges from 4–8V DC
- Rated current: less than or equals to 30mA
- Sound output at 10cm: greater than or equals to 85dB
- Resonant frequency: 2300 ±300Hz
- Operating temperature: -25°C to +80°C
- Storage temperature: -30°C to +85°C
- Frequency range: 2000–4000Hz for piezoelectric buzzers, starting at 800Hz for electromagnetic buzzers

2.6.2 Applications:

- Novelty uses
- Judging panels
- Educational purposes
- Annunciator panels
- Electronic metronomes
- Game show lock-out device
- Microwave ovens and other household appliances
- Sporting events such as basketball games
- Electrical alarms
- Joy buzzer (mechanical buzzer used for pranks)

2.6.3 Connection:

To connect a buzzer to an ESP32, we need to connect the buzzer's positive (+) terminal to one of the digital output pins on the ESP32 (GPIO pin), and the negative (-) terminal to ground (GND) pin on the ESP32. Then, we can control the buzzer by toggling the output pin HIGH and LOW using code.

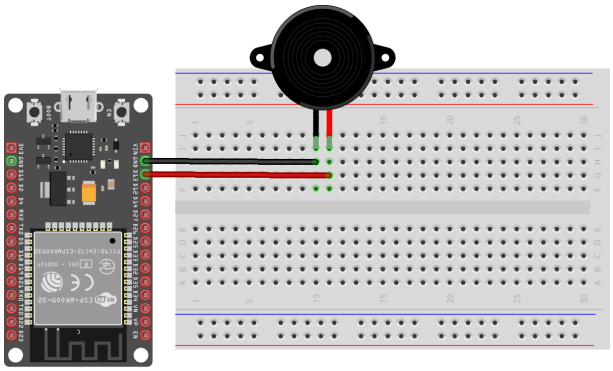


Figure 2.12: Buzzer-ESP32

2.7 Arduino IDE:

The Arduino IDE (Integrated Development Environment) is a software platform used for programming Arduino microcontrollers. It provides a user-friendly interface for writing, compiling, and uploading code to Arduino boards. With a simple and intuitive interface, it enables developers to create embedded projects quickly and efficiently. The IDE includes a text editor with syntax highlighting, a compiler, and a serial monitor for debugging. It supports a wide range of libraries and offers extensive documentation, making it accessible to beginners and advanced users alike.

The Arduino IDE code serves as the software backbone of our project, facilitating communication between hardware components, data processing, and interaction with external services like Google Sheets.

2.8 Arduino IDE Libraries

2.8.1 Wire.h

This library provides functions for I2C communication, used for interfacing with the sensors and OLED display.

2.8.2 Adafruit_SH110X.h

This library is for controlling SH110X-based OLED displays, used for displaying information.

2.8.3 Adafruit_PN532.h

This library facilitates communication with the PN532 NFC/RFID controller, enabling RFID tag detection and reading.

2.8.4 Adafruit_MLX90614.h

This library is used for interfacing with the MLX90614 infrared thermometer sensor, enabling temperature measurement.

2.8.5 HTTPClient.h>

This library enables secure communication over HTTPS, used for transmitting data to Google Sheets via a secure connection.

2.8.6 Wifi.h

This library enables Wi-Fi communication for IoT projects and network connectivity.

Chapter 3

Connection and Prototype

3.1 Connection Algorithm:

1. Connect RFID Reader (PN532) to ESP32:
   a. Connect SDA pin of RFID reader to SDA pin of ESP32 (GPIO pin 21).
   b. Connect SCL pin of RFID reader to SCL pin of ESP32 (GPIO pin 22).
   c. Connect VCC pin of RFID reader to 3.3V pin of ESP32.
   d. Connect GND pin of RFID reader to GND pin of ESP32.
2. Connect Temperature Sensor (MLX90614) to ESP32:
   a. Connect SDA pin of temperature sensor to SDA pin of ESP32 (GPIO pin 21).
   b. Connect SCL pin of temperature sensor to SCL pin of ESP32 (GPIO pin 22).
   c. Connect VCC pin of temperature sensor to 3.3V pin of ESP32.
   d. Connect GND pin of temperature sensor to GND pin of ESP32.
3. Connect OLED Display to ESP32:
   a. Connect SDA pin of OLED display to SDA pin of ESP32 (GPIO pin 21).
   b. Connect SCL pin of OLED display to SCL pin of ESP32 (GPIO pin 22).
   c. Connect VCC pin of OLED display to 3.3V pin of ESP32.
   d. Connect GND pin of OLED display to GND pin of ESP32.
4. Power Supply:
   a. Connect suitable power supply (e.g., USB cable or battery) to ESP32.
   b. Ensure power supply can provide sufficient power for all components.

The RFID , OLED and MLX90614 communicate with the ESP32 microcontroller via the I2C protocol. All the devices share a common ground (GND), and powered by connecting their VCC pin to a 3v3 pin of the ESP32. SDA (data line) and SCL (clock line) are used for data transfer. As we are connecting the SDA and SCL pins of all the devices to GPIO21 and GPIO22 pin of ESP32, so we have done an extension of GPIO21 and GPIO22 in the Board.

3.2 Our Device:

We have designed a device like we have to show our hand for temperature scanning and the card for user identification so that it will take the data and show in OLED display and send it to google sheet.

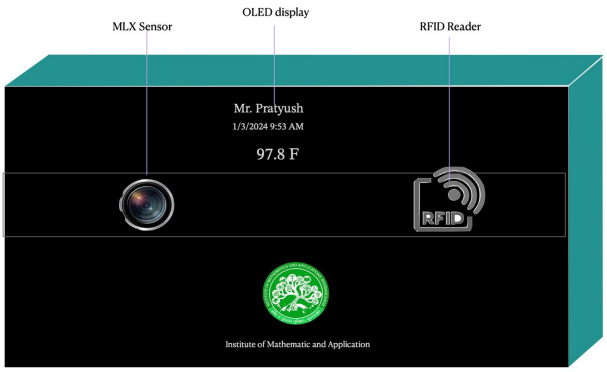


Figure 3.1: Target Prototype

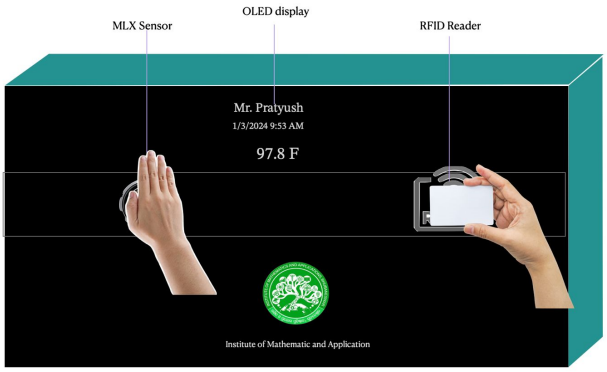


Figure 3.2: Working of target prototype

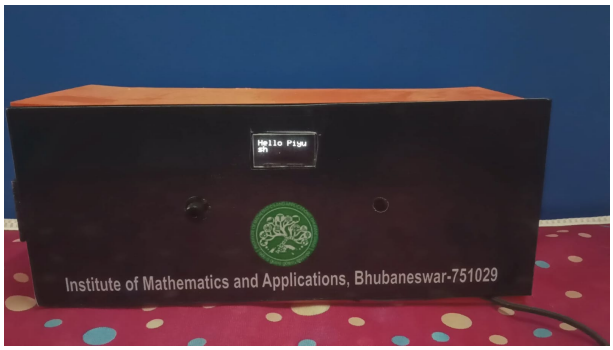


Figure 3.3: Final Prototype

3.2.1 Working Procedure:

- Hand Temperature Measurement: Users have to place their hand in front of the MLX90614 temperature sensor, which utilizes infrared radiation to measure the temperature of the hand without physical contact.
- RFID Card Identification: Simultaneously, users show their RFID card to the PN532 RFID reader, which reads the unique identifier (UID) embedded in the card.
- Detection Trigger: When the RFID reader detects a card, it sends a signal to the ESP32.
- Buzzer Activation: Upon receiving the signal, the microcontroller activates the buzzer to produce a sound.
- Data Processing: The ESP32 microcontroller receives the hand temperature and RFID UID data from the respective sensors. It processes the data and prepares it for transmission.
- OLED Display Output: The processed data, including hand temperature and RFID UID, is displayed on the OLED screen in real-time for user visualization.
- Data Transmission: The ESP32 establishes a Wi-Fi connection to the internet and transmits the collected data to a designated Google Sheet. The transmitted data includes the timestamp, RFID UID and User's temperature.
- Cloud Storage and Analysis: The data received from the device is stored in the Google Sheets spreadsheet, allowing for remote access and analysis. Users can monitor hand temperature trends and track individual user identification events over time.

3.2.2 Transfer of Data from ESP32 to OLED:

Transfer of data from the ESP32 microcontroller to the OLED display involves several steps:

- **Data Acquisition:**
  The ESP32 collects data from various sensors, such as the MLX90614 temperature sensor and the PN532 RFID reader. Once the data is collected, it is processed and prepared for display on the OLED screen.
- **Display Initialization:**
  The OLED display is initialized by the ESP32 using appropriate libraries and commands. This involves specifying parameters such as screen dimensions, text size, and text color.
- **Data Rendering:**
  After initializing the OLED display, the ESP32 instructs the display to render the collected data. The ESP32 determines the layout and content of the data to be displayed on the OLED screen, including text, numerical values, and graphical elements.
- **Display Update:**
  The ESP32 continuously updates the OLED display to reflect any changes in the data

being monitored or displayed. This involves sending commands to the OLED display to clear the screen, set the cursor position, and print new data values.

- **Data Presentation:**
  The ESP32 sends commands to the OLED display to present the collected data in a readable format. This may involve organizing the data into separate lines or sections on the OLED screen for better readability and user comprehension.
- **Continuous Monitoring:**
  The ESP32 continually monitors the data from sensors and updates the OLED display accordingly. This ensures that users have access to real-time information and can make informed decisions based on the displayed data.

Chapter 4

Implementation and Integration

4.1 Arduino IDE:

The software platform used to program Arduino microcontrollers is called the Arduino IDE (Integrated Development Environment). It provides a user-friendly interface for writing, compiling, and uploading code to Arduino boards.It helps developers to quickly and effectively create embedded projects. The IDE includes a text editor with syntax highlighting, a compiler, and a serial monitor for debugging. It supports a wide range of libraries making it accessible to beginners and advanced users alike.

The software foundation of our project is the code written for the Arduino IDE, which allows us to process data, communicate with external services like Google Sheets, and interface with hardware components.

4.1.1 Arduino IDE Code:

```
#include <Wire.h>
#include <Adafruit_SH110X.h>
#include <Adafruit_PN532.h>
#include <Adafruit_MLX90614.h>
#include <HTTPClient.h>
#include <WiFi.h>
#define SDA_PIN 21 // Define your SDA pin here
#define SCL_PIN 22 // Define your SCL pin here
#define i2c_Address 0x3C // Change this to the desired I2C address

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

```
#define OLED_RESET -1
Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT,
&Wire, OLED_RESET);
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
Adafruit_PN532 nfc(SDA_PIN, SCL_PIN);

const char* ssid = "PIYUSH";
const char* password = "12345678";
const char* GOOGLE_SCRIPT_ID =
"AKfycbwF9oknQ1p491FANXnTnA_bon4748tMolF9im9wxRUZZvCfrCzmp5YGkeU494Yu6Vwtw";

const int buzzerPin = 23;

struct RFIDTag {
  uint8_t uid[4]; // UID of the tag
  uint8_t uidLength; // Length of the UID
  const char* name; // Name associated with the tag
};

// Define an array of RFID tags and their associated names
RFIDTag knownTags[] = {
  {{0x73,0x83,0x36,0xFE},4,"Piyush"},
  {{0xB3,0x8F,0x3F,0xFE},4,"Madhav"},
  {{0x3,0x40,0x42,0xFE},4,"Sandip"},
  {{0x53,0x85,0x3E,0xFE},4,"Ashish"},
  {{0x83,0x29,0x3E,0XFE},4,"Asim"},
  {{0xC3,0XF3,0X92,0XF5},4,"Maharshi"},
  {{0xB3,0x73,0x2C,0xFE},4,"Bibhu"},
  {{0x93,0x89,0x2D,0xFE},4,"Ananta"},
  {{0x23,0xBA,0x2C,0xFE},4,"Sambit"},
  {{0xC3,0x5B,0x2D,0xFE},4,"Ashutosh"},
  {{0xF3,0x22,0x2A,0xFE},4,"Shiv"}
};

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

  Wire.begin(SDA_PIN, SCL_PIN); // Initialize I2C with defined SDA and SCL pins
  mlx.begin();
```

```
  nfc.begin();

  if (!display.begin()) { // Initialize SH110X display
    Serial.println(F("Something went wrong"));
    for (;;);
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SH110X_WHITE);

  // Connect to Wi-Fi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

pinMode (buzzerPin,OUTPUT);

}

void loop() {
  display.clearDisplay();
  display.setCursor(0, 16);
  display.println("Show your card  &   your Palm");
  display.display();

  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength; // Length of the UID (4 or 7 bytes depending on
  ISO14443A card type)
  float temperature = mlx.readObjectTempC();
  String params;
```

```
// Wait for an RFID card
success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

if (success) {
  Serial.println("Found an RFID tag!");
  for (int i = 0; i < sizeof(knownTags) / sizeof(knownTags[0]); i++) {
    if (memcmp(uid, knownTags[i].uid, uidLength) == 0) {
      Serial.print("HELLO ");
      Serial.println(knownTags[i].name);
      // BUZZER SOUND
      digitalWrite(buzzerPin, HIGH);
      delay(1000);
      digitalWrite(buzzerPin,LOW);
      // RFID DATA
      display.clearDisplay();
      display.setCursor(0, 16);
      display.print("Hello ");
      display.println(knownTags[i].name);
      display.display();
      delay(2000);
      // MLX DATA
      display.clearDisplay();
      display.setCursor(0, 16);
      display.print("Temperature: ");
      display.print(temperature);
      display.println(" C");
      display.display();
      delay(3000);
      //THANKS GIVING
      display.clearDisplay();
      display.setCursor(0,16);
      display.print("Thank you ");
      display.println(knownTags[i].name);
      display.display();
      display.clearDisplay();
      delay(3000);

    }
  }
```

```
    // Prepare parameters for sending to Google Sheets
    params = "UID=";
    for (uint8_t i = 0; i < uidLength; i++) {
      params += String(uid[i], HEX);
    }
    params += "&Temperature=" + String(temperature);

    // Send data to Google Sheets
    sendData(params);

    delay(1000);
  }
}

void sendData(String params) {
  HTTPClient http;
  String url = "https://script.google.com/macros/s/" +
  String(GOOGLE_SCRIPT_ID) + "/exec?" + params;

  Serial.print("Requesting URL: ");
  Serial.println(url);

  if (http.begin(url)) {
    int httpCode = http.GET();
    http.end();
    Serial.print("Response code: ");
    Serial.println(httpCode);
  } else {
    Serial.println("Failed to connect to Google Sheets");
  }
}
```

4.1.2 Theoretical description:

The theoretical description of the Arduino IDE code includes the following aspects,

- **Initialization:**
  The code starts with importing necessary libraries like Wire.h, Adafruit_SH110X.h, Adafruit_PN532.h, Adafruit_MLX90614.h, HTTPClient.h and WiFi.h. These libraries provide functions and classes for interfacing with various hardware components and online services.
- **Setup Function:**
  The setup() function initializes the microcontroller, configures pins, and establishes connections. This includes initializing the I2C communication for the OLED display and sensors, setting up the PN532 NFC reader, and connecting to a Wi-Fi network.
- **Main Loop:**
  The loop() function is the main execution loop of the program, continuously running and handling tasks. Within the loop, the code performs tasks such as reading RFID tags using the PN532 NFC reader, measuring temperature using the MLX90614 sensor, and displaying information on the OLED display. It also prepares data for transmission to Google Sheets by formatting it appropriately.
- **Sending Data to Google Sheets:**
  The code defines a function sendDataToGoogleSheets() for sending data to Google Sheets. This function establishes a secure connection to the Google Sheets API using the WiFiClientSecure library. It constructs a URL containing the data to be sent as parameters and makes an HTTP GET request to the Google Apps Script deployed as a web service. The Google Apps Script processes the received data and appends it to a designated Google Sheet.
- **Error Handling:**
  The code includes error handling mechanisms to deal with potential issues such as failed connections to Wi-Fi or Google Sheets. Error messages are printed to the serial monitor for debugging purposes.
- **External Dependencies:**
  The code relies on external resources such as Google Sheets and Google Apps Script. The Google Apps Script serves as an intermediary between the ESP32 and Google Sheets, facilitating the storage of data.

4.1.3 Output:

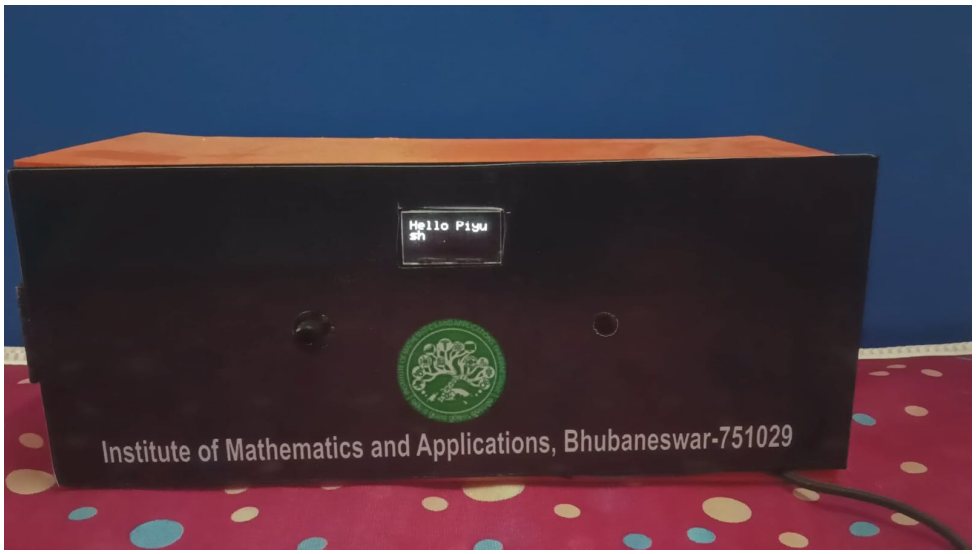


Figure 4.1: Welcome User

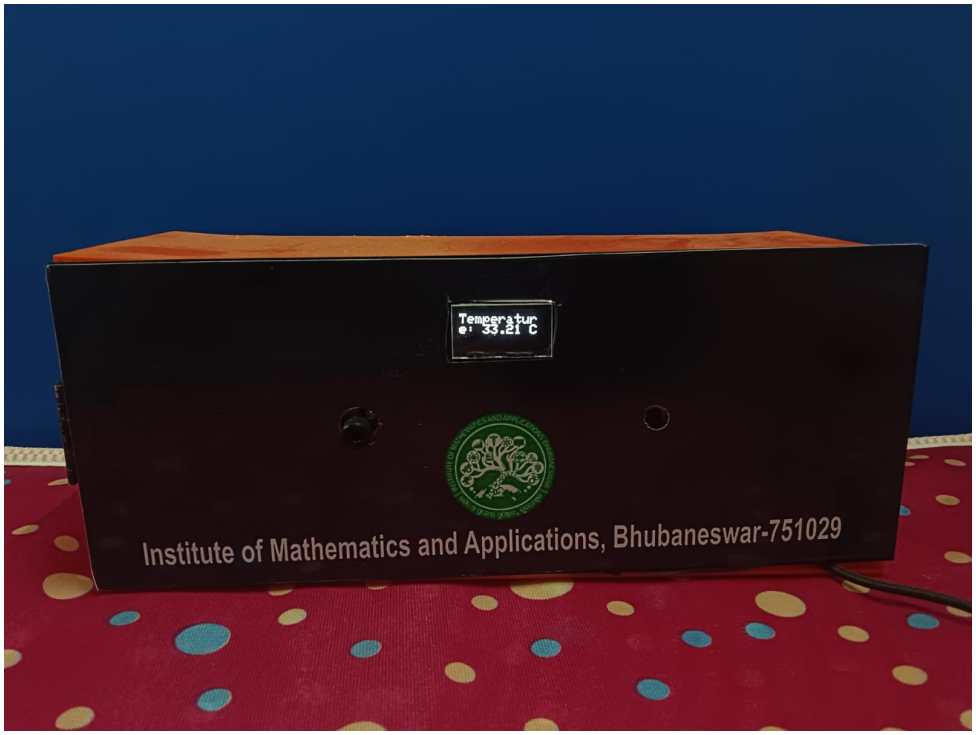


Figure 4.2: Temperature of the User

4.2 Google Sheet Integration:

Google Sheet is a cloud-based spreadsheet application developed by Google. It offers users the ability to create, edit, and collaborate on spreadsheets in real-time, with features such as formulas, charts, and conditional formatting. Sheets can be accessed from any device with an internet connection, making it highly versatile and accessible.

So the procedure of sending data from ESP32 to Google sheet is given below.

- **Set up Google Sheets:**
  Open your web browser then Google drive and navigate to Google Sheets (sheets.google.com). Create a new spreadsheet by clicking on the "+" button or selecting "Blank" from the template gallery.

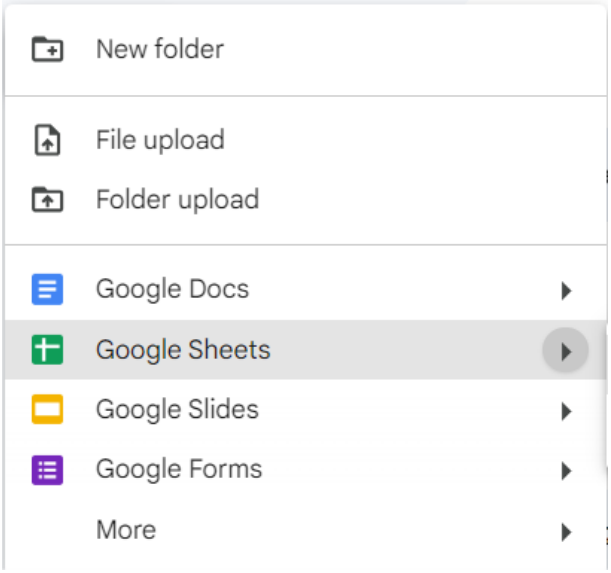


Figure 4.3: Creating new Google sheet

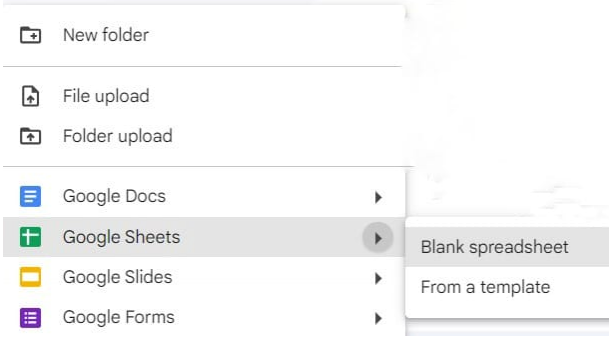


Figure 4.4: Click on Blank Spreadsheet

- **Create a Google Apps Script:**
  In Google Sheets, click on "Extensions" in the top menu. Select "Apps Script" from the dropdown menu.

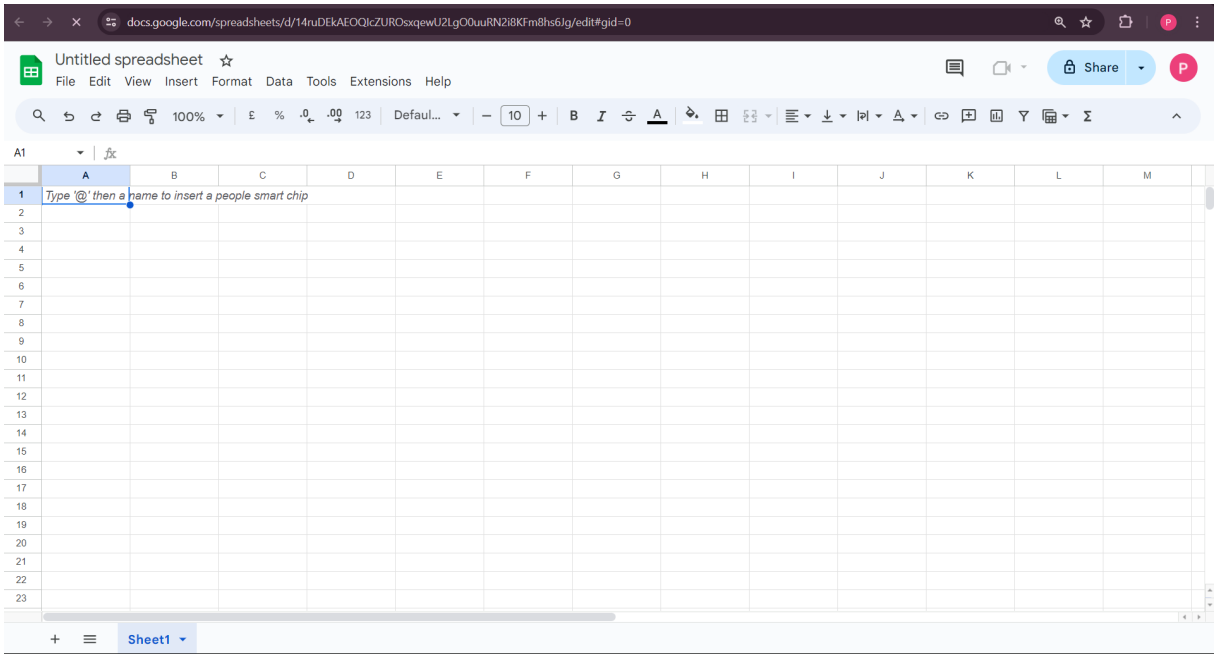


Figure 4.5: Blank Sheet

In the Google Apps Script editor, delete any existing code and paste the provided script.

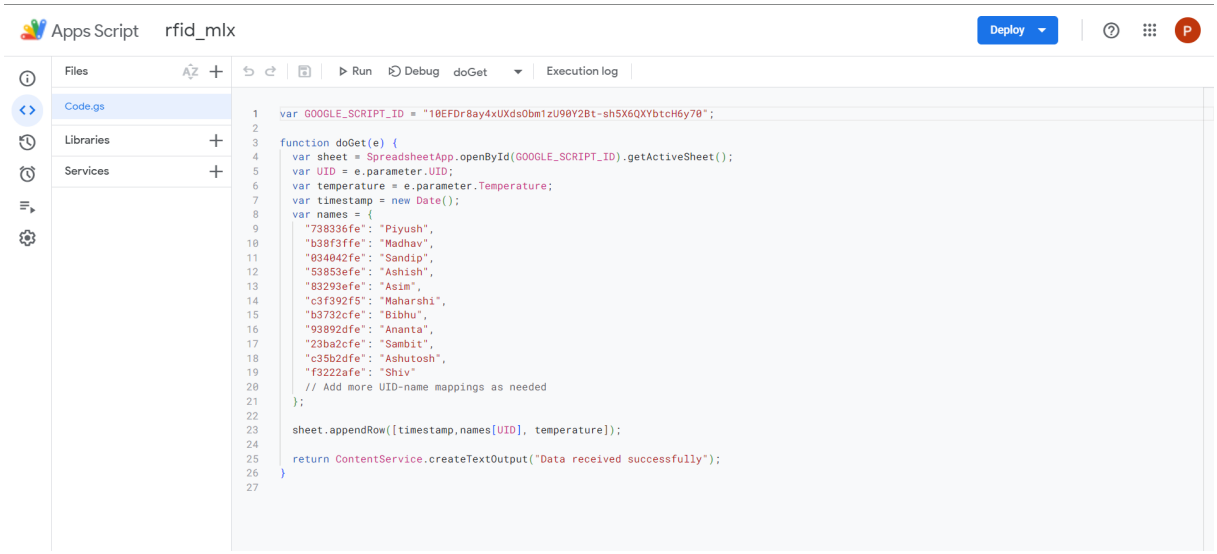


Figure 4.6: Required code

Save the script by clicking on the disk icon or selecting "Save" from the "File" menu. Close the script editor tab.

- **Get Script ID:**
  In the Google Apps Script editor, click on the "Deploy" in the top right corner.

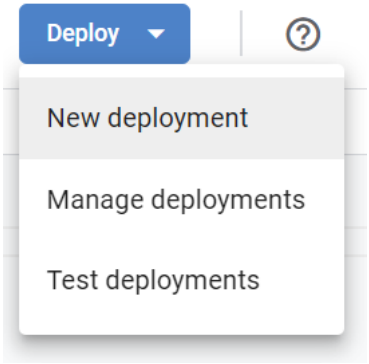


Figure 4.7: Deploy the Script

Select as "Web App" and then "Deploy".

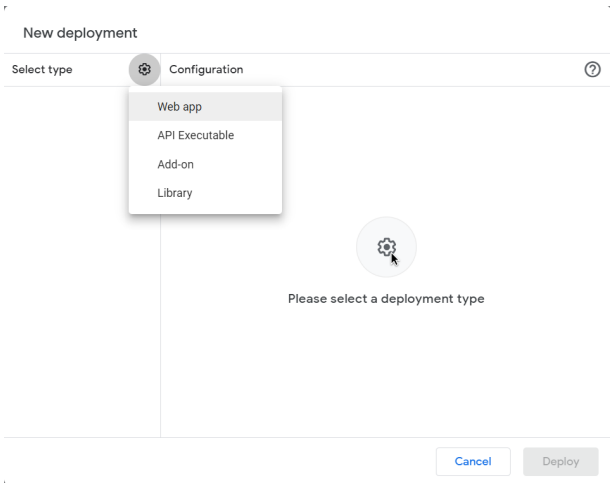


Figure 4.8: Deploy it as Web app

It will show a deployment ID and URL , then Copy the "Deployment ID".

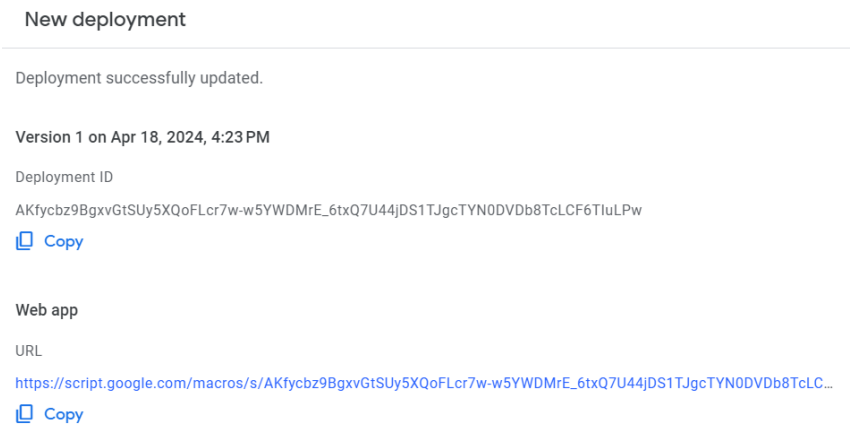


Figure 4.9: Deployment ID

- **Configure ESP32:**
  Open the Arduino IDE on your computer. Copy and paste the provided code given before into a new sketch. Replace GOOGLE_SCRIPT_ID with the Script ID obtained from the Google Apps Script. Upload the sketch to your ESP32 board

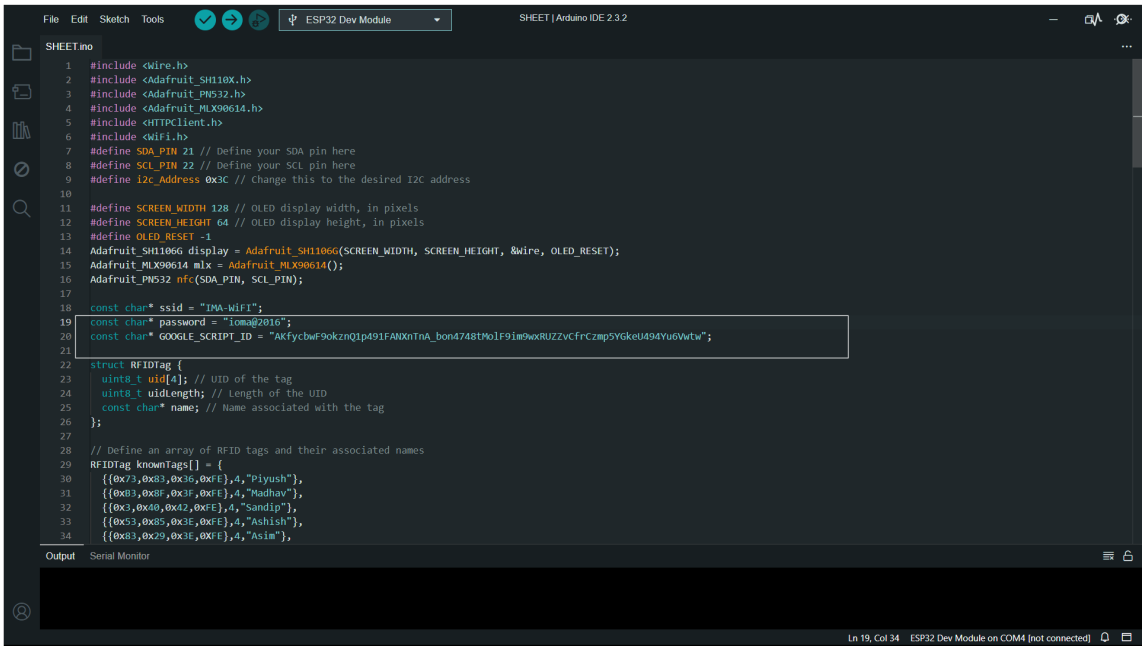


Figure 4.10: Pasting the Deployment ID in Arduino IDE

.

- **Send Data from ESP32:**
  Power on your ESP32 board. The ESP32 will collect data (e.g., sensor readings) according to the code and send it to the Google Apps Script using HTTP requests.
- **Handle Data in Google Apps Script:**
  The Google Apps Script will receive the data sent by the ESP32 and append it to the Google Sheets spreadsheet.
- **Test and Debug:**
  Monitor the Google Sheets spreadsheet to ensure that data is being correctly appended. Use the Arduino IDE serial monitor to debug any issues with the ESP32 code.

4.2.1 Script code:

The script code is written in JavaScript which serves as the back-end logic for web applications. In our project, the script code is used in conjunction with Google Apps Script to handle incoming requests from the ESP32 device and manipulate Google Sheets data accordingly. Here is the script code we have used ,

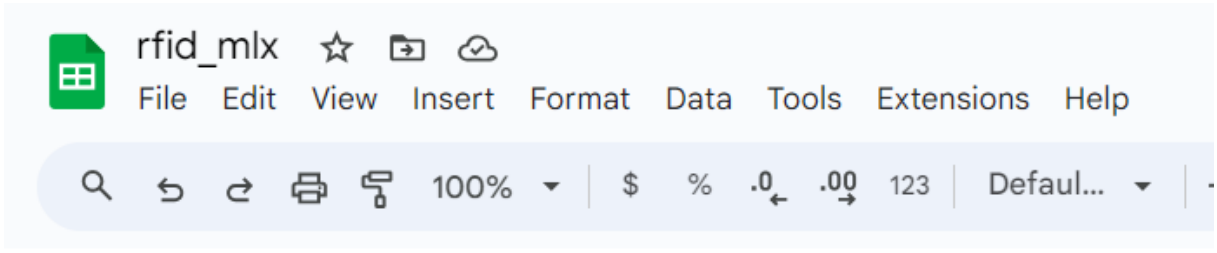
```
var GOOGLE_SCRIPT_ID = "10EFDr8ay4xUXdsObm1zU90Y2Bt-sh5X6QXYbtcH6y70";

function doGet(e) {
  var sheet = SpreadsheetApp.openById(GOOGLE_SCRIPT_ID).getActiveSheet();
  var UID = e.parameter.UID;
  var temperature = e.parameter.Temperature;
  var timestamp = new Date();
  var names = {
    "738336fe": "Piyush",
    "b38f3ffe": "Madhav",
    "034042fe": "Sandip",
    "53853efe": "Ashish",
    "83293efe": "Asim",
    "c3f392f5": "Maharshi",
    "b3732cfe": "Bibhu",
    "93892dfe": "Ananta",
    "23ba2cfe": "Sambit",
    "c35b2dfe": "Ashutosh",
    "f3222afe": "Shiv"
    // Add more UID-name mappings as needed
  };

  sheet.appendRow([timestamp,names[UID], temperature]);

  return ContentService.createTextOutput("Data received successfully");
}
```

4.2.2 Theoretical Description:

So the script code defines functions that execute when triggered by HTTP requests. For instance, the doGet(e) function is called when the ESP32 device sends data via a GET request. Within this function, the code accesses the Google Sheets spreadsheet, extracts parameters from the request (such as UID and temperature), and appends the data to the spreadsheet.

It also includes mappings for UID values to corresponding names, facilitating the organization of data in the spreadsheet. This script code runs on Google's servers, enabling seamless integration between the ESP32 device and Google Sheets, and facilitating real-time data logging and analysis.

4.2.3 Output:



	A	B	C	D	E
1	DATE AND TIME	UID	TEMPERATURE		
2	4/9/2024 13:02:41	Ashish	31.27		
3	4/9/2024 13:03:09	Asim	31.27		
4	4/9/2024 13:03:20	Maharshi	30.99		
5	4/9/2024 13:03:52	Madhav	30.81		
6	4/9/2024 13:17:30	Ananta	30.47		
7	4/9/2024 13:18:56	Madhav	31.07		
8	4/9/2024 13:21:22	Piyush	30.99		
9	4/9/2024 17:50:15	Ashish	29.99		
10	4/9/2024 17:50:43	Ananta	29.77		
11	4/10/2024 19:01:1	Ashish	30.05		
12	4/18/2024 23:16:5	Shiv	32.47		
13	4/18/2024 23:17:2	Sambit	32.63		
14	4/18/2024 23:17:4	Madhav	32.99		
15	4/18/2024 23:18:4	Shiv	33.35		
16	4/18/2024 23:19:2	Madhav	33.19		
17	4/18/2024 23:21:0	Ashish	32.91		
18	4/18/2024 23:22:5	Ashish	32.91		
19	4/18/2024 23:23:0	Piyush	32.67		
20	4/18/2024 23:23:2	Maharshi	33.03		

Figure 4.11: The Google sheet with Date, Time, UID and Temperature

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Our project represents a successful combination of RFID technology, temperature sensing and cloud-based data management using the ESP32 microcontroller. Seamlessly integrating these components, we have created a versatile and efficient data collection system. Using Google Sheets as a back end database allows for convenient storage, retrieval and analysis of collected data. This system facilitates real-time monitoring and enables users to remotely access and analyze information.

5.2 Future Work

In the future, possible improvements might include adding more sensors for thorough environmental monitoring, deploying sophisticated data analysis algorithms for more detailed analysis and improving the user interface for better usability and interactivity with further refinement and development. Our project holds promise for various applications ranging from industrial automation to healthcare and beyond. In near future, we would like to do the following:

- **Enhanced Data Analysis:** Implement more advanced data analysis techniques on the collected data to derive deeper insights or predictive models.
- **User Authentication:** Integrate user authentication mechanisms to ensure secure access to the system and data.
- **Real-Time Monitoring:** Enable real-time monitoring of temperature and RFID data, allowing users to receive immediate alerts or notifications.
- **Mobile Application:** Develop a mobile application to provide a user-friendly interface for interacting with the device and accessing data remotely.
- **Integration With IoT Platforms:** Integrate our device with IoT platforms like Google Cloud IoT or AWS IoT for scalability, data management and analytics capabilities.

- **Hardware Optimization:** We will explore ways to optimize the hardware design for better performance, power efficiency and cost-effectiveness.
- **Expand Sensor Capabilities:** Add additional sensors for monitoring environmental conditions, such as humidity, air quality, or motion detection.
- **Machine Learning Integration:** Implement machine learning algorithms for anomaly detection, pattern recognition, or predictive maintenance based on the collected sensor data. [10]
- **Feedback Mechanism:** Incorporate a feedback mechanism to gather user feedback and improve the system based on user experiences and suggestions.
- **Cellular automata Integration:** In future we can incorporate Cellular automata with IoT.

References

[1] R. Arif, M. Irviana, W. Rahma, K. Santoso, and W. S., "Use of the mlx 90164 sensor and the thingspeak platform for internet of things-based animal body temperature check," *ARSHI Veterinary Letters*, vol. 5, pp. 35–36, Oct. 2021.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] F. Z. Chafi and Y. Fakhri, "Introduction to integrate the cellular automata concept within the internet of things: The use of the dynamic management of bridge approach," in *Innovations in Smart Cities Applications Volume 4* (M. Ben Ahmed, Rakıp Karaş, D. Santos, O. Sergeyeva, and A. A. Boudhir, eds.), (Cham), pp. 1030–1043, Springer International Publishing, 2021.

[4] P. Rusimamto, R. Harimurti, E. Endryansyah, Y. Anistyasari, and L. Anifah, "Design and implementation of thermal body system employing thermal sensor mlx 90614 for covid-19 symptoms early detector," 01 2020.

[5] A. Ahmed, M. N. Abdullah, and I. Taib, "Design of a contactless body temperature measurement system using arduino," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, p. 1251, 09 2020.

[6] K. M. Abubeker, S. Baskar, and M. K. Roberts, "Internet of healthcare things-enabled open-source non-invasive wearable sensor architecture for incessant real-time pneumonia patient monitoring," in *Innovations in VLSI, Signal Processing and Computational Technologies* (G. Mehta, N. Wickramasinghe, and D. Kakkar, eds.), (Singapore), pp. 217–225, Springer Nature Singapore, 2024.

[7] H. Navin Kumar, S. Navya, P. Agnihotri, V. K. Pratima, and P. N. Sudha, "Literature review on "hybrid temperature sensing and monitoring system with built-in sanitizer"," *International Advanced Research Journal in Science, Engineering and Technology*, vol. 7, pp. 66–70, December 2020.

[8] S. Lee, J. H. Kim, and H. Lee, "Rfid technology and its applications," *International Journal of Distributed Sensor Networks*, vol. 13, no. 10, pp. 1–14, 2017.

[9] M. Garcia and C. Martinez, "Iot applications for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 75–88, 2016.

[10] Y. Bengio, Y. LeCun, and G. Hinton, "Machine learning: From theory to applications," *Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1–4, 2021.