IOT Fingerprint Lock System

Siddhartha Lama (2112923)

Sulav Thapa (2012220)

Subash KC (2012190)

Nur Islam (2012189)

Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology

23 December 2022

Abstract

Author:	Siddhartha Lama, Sulav Thapa, Subash, Nur Islam
Title:	IoT Fingerprint Lock System
Number of Pages:	18 pages + 1 appendix
Date:	23 December 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Smart IoT Embedded System
Supervisor:	Joseph Hotchkiss, Lecturer, Metropolia UAS

This project aims to create an automated locking system that collects optical variables such as the amount of light reflected by the fingers to identify rides in the fingers to execute control actions of unlocking doors or any types of automated locks. It also aims to completely automate the lock-unlock mechanism. It further aims to create a status update through the cloud and notify the owner about current status of the lock itself and allow them to completely control it.

Keywords: fingerprint scanner

Table of Contents

List of Abbreviations

OLED	Organic Light Emitting Diode
AC	Alternating current
DC	Direct current
GND	Ground signal
IO	Input output
UART	Universal asynchronous receiver-transmitter
MCU	Microcontroller unit
LED	Light Emitting Diode
TFT	Thin-film Transistor
IOT	Internet of Things
AMOLED	Active-Matrix Organic Light Emitting Diode
CCTV	Closed-Circuit Television

1 Introduction

1.1 Background

Door locks are an essential part of history. Door locks have been in existence ever since the invention of houses and house theft. They have come in many shapes and forms throughout the centuries, but the fundamental principle remains the same, protection of belongings from unwanted intruders, be it humans or animals. As metal works started to improve, so did the locks and modern locks by themselves aren’t any different from what existed a hundred years ago. What has changed is the way the locks can be opened. To unlock a door, one needs a key. Until recent decades, the lock-unlock mechanism has remained the same. The owner, administrator or moderator of the property holds the key to the lock and can lock/unlock it whenever they want. The security mechanism is based on a simple philosophy along the lines of ‘protect the one who protects you’, which in this case means carrying the keys wherever we go and keeping them in safe places where unwanted people cannot find them. But humans are forgetful and can lose their keys very easily.

A subsequent improvement was seen in door locks with the addition of mechanical dials which allowed users to set the password. The correct combination of numbers could unlock the door. It was basically the same principle but without a physical key. The key was now abstract. Now there was a lesser chance of forgetting the keys as one could easily write their passkey down on a piece of paper and would be able to commit the password to memory by opening their lock enough times. Still, with the many improvements brought about by the improvement of key opening technology, the security provided by general keys are nowhere close to what modern technology can facilitate. For older keys, if it’s lock is broken, the owner won’t find out about the event unless they themselves notice it or are notified about it by someone. They are in essence, dependent on the fact that someone notices foul play as soon as possible in order to carry out rapid intervention. A technology that improved this aspect of improving rapid intervention, which is another layer of security that governments usually provide through the Police force, is the natural step in this security improvement progression. This was brought about by electronic locking systems that relied on passwords stored in memory chips to help users' setup their security systems. The electronics of the lock were also connected to the lock itself in order to raise an alarm if the lock is tampered with. This was and has been the peak point of the locking mechanism ever since it's invention.

Modern day locks do not directly improve the physical security itself. But they can improve other layers of security similar to rapid response. They can be made smarter to make the experience seamless for the user while simultaneously, being extremely deterring towards an intruder. They can achieve this by continuously communicating with the owner via their smartphones and raising an alarm on their smartphones and home as well if someone has intruded. The unlocking systems in homes can be made up of fingerprint sensors which eliminate the need for remembering a password as no two people have the same fingerprints and stealing fingerprints is not an easy task. More advanced systems have been developed by companies which scan the iris of a person's eye in order to create a high-resolution image with a lot of 3D data of the person's iris and the shape of their face. The current possibilities for a smart security methodology are plenty and provide many layers of security as required by the scale of protection that the user wants. More advanced facilities are even researching on pattern recognition software to detect intrusive behaviour through CCTV (Closed-circuit Television) cameras for high security facilities like important government offices, banks, facilities or headquarters of high-tech companies and organizations or even low-tech companies with trade secrets to guard. Such systems can provide an intrusion index in order to create early judgements and alert the security personnel to tighten their security.

From a historical standpoint to modern day, door lock security systems are an essential part of any modern security system and will remain so for centuries to come. They are the first layer of security of any institution, organization, corporation or residence. Modern technology deems that they be extremely secure even at a low cost and provide all the benefits that can come with it.

Driven by such a theme, this project explores one such modern security technology, which is a fingerprint enabled door-locking security system.

1.2 Objectives

The main objectives of the project include:

- Exploration and scoping of optical fingerprint scanning technology
- To create an algorithm to capture multiple optical data instances (for each individual) generated by the sensor and process it to create a unique fingerprint database to compare future fingerprints against
- To create a fully automated operation which unlocks the lock after a fingerprint match and promptly closes the lock after a default value of 10 seconds which can be set to other values by the users
- Integration of a display for users to interact with and view the results after their fingerprint has been scanned
- Development of a cloud-based data dispatch system which allows users to virtually control the system parameters to suit their needs and enrol new fingerprints

1.3 Applications

The main applications of this project are listed below:

- Households that desire a 24/7 secure system with continuous updates
- Institutions with classrooms or lab buildings that want only the employees and enrolled students to get entry
- Organizations/corporations/institutions that want to use the fingerprint system both as an attendance system as well as a security layer
- Places that need locks to be modulated for automatically unlocking themselves in timed intervals depending on the nature of the application, examples of which could be art museums, zoos or even public parks which need to remain closed usually during the night and open during the day
- Industrial storage silos & control rooms

2 Project Plan

2.1 Project Description

This project aims to create a digital locking system which improves the security level of any given lock multiple-folds by using the human fingerprint, one of the most unique identifiers of an individual. Such a system is highly reliable and secure. The project aims to achieve smooth communication with the sensor via a microcontroller and similar smooth communication with the cloud. The cloud shall exhibit a monitoring service which allow admins to view the status of all their sensors such that they can completely monitor who is entering where within any given building layout. The sensors also aim to aid such admins to device security clearance level plans and programs to control access of resources within their organizations. The system is also to be fully automated such that the locks can re-lock themselves after being unlocked to add to their already reliable and robust operation.

2.2 Project Communication

The name of our team members in this project are Siddhartha Lama, Sulav Thapa, Subash KC, and Nur Islam. Our lead teacher in this topic is Joseph Hotchkiss. Platforms such as google drive is used for updating the documentation of every step and progress of the project. Tools such as planners or version control might be added later on if necessary. For regular instant communication WhatsApp group has been formed which has appeared to be quite handy and very useful. Meeting will be held at least a day in a week in person as a group for discussions and project implementation. As for the rest of the week planning has been done to arrange zoom and google meet meetings twice a week. In addition to these, 4 meetings will be done with the lead teacher after discussing the appropriate time and his/her suggestions can be considered and the project team can also update about the progress on the project.

2.3 Resourcing

Product Name	**Product No. (With hyper-links to website)**	**Unit price ($)**	**Qty.**	**Estimated Delivery (weeks)**
4-Ch Relay (3.3V trigger)	*ELC ARE00104SL*	14.9	1	1
Jumper wires male to male	*485-4447*	9.95	1	1
Jumper wires female to female	*485-4482*	9.95	1	1
Power Bank 9V 2A	*485-4288*	18.95	1	1
ESP32 Microcontroller	*356ESP32DEVKIT*	9	1	1
OLED Screen	*426-DFR0648*	14.95	1	1
Fingerprint Sensor Module	*485-751*	49.95	1	1
Magnetic Door Lock	*FIT0624*	7.90	1	1

Table 1: Table of relevant products available from Mouser, Partco & Farnell

2.4 Components Description

From the list of components, all are used to realize the system. The ESP32 is the responsible for processing and communications, the OLED display is responsible for viewing and human interaction & the fingerprint sensors and relays are responsible for sensing and actuating actions. Each component serves a whole block of tasks. Sensing has no other parts than the fingerprint scanner. Processing and communication have no other integration requirements except its own programming and internet connectivity. Display has no other purpose than notifying users that they have been granted access and the relays do not have any further power requirements in order to operate higher power rating devices.

ESP 32 Microcontroller

Figure 1: ESP 32 MCU

The ESP32 family of system-on-a-chip microcontrollers with features like integrated Wi-Fi and dual-mode Bluetooth. It is inexpensive and has very low power consumption. The Tensilica Xtensa LX6 dual-core or single-core microprocessor, Tensilica Xtensa LX7 dual-core, or a single-core RISC-V microprocessor are used in the ESP32 series, which also has integrated antenna switches, RF baluns, power amplifiers, low-noise receive amplifiers, filters, and power-management modules. Chinese business Espressif Systems, with headquarters in Shanghai, invented and constructed the ESP32, which is produced by TSMC using their 40 nm technology. It is the ESP8266 microcontroller's replacement. [1] Since the release of the original ESP32, a number of variants have been introduced and announced. They form the ESP32 family of microcontrollers. These chips have different CPUs and capabilities, but all share the same SDK and are largely code-compatible. Additionally, the original ESP32 was revised (see ESP32 ECO V3, for example).

OLED Display

Figure 2: An OLED Display

A light-emitting diode (LED) with an organic compound film as the emissive electroluminescent layer generates light in response to an electric current is referred to as an organic light-emitting diode (OLED or organic LED), also known as an organic electroluminescent (organic EL) diode. This organic layer is sandwiched between two electrodes, usually with at least one transparent electrode. OLEDs are utilized to make digital displays in gadgets like televisions, computer monitors, and portable gaming systems like smartphones. The creation of white OLED components for use in solid-state lighting systems is a significant field of research.[2]

OLEDs go into one of two categories: those using polymers or tiny molecules. An OLED can be converted into a light-emitting electrochemical cell (LEC), which operates in a slightly different manner, by adding mobile ions. A passive-matrix (PMOLED) or active-matrix (AMOLED) control strategy can be used to drive an OLED display. AMOLED control employs a thin-film transistor (TFT) backplane to directly access and switch each individual pixel on or off, enabling for higher resolution and larger display sizes, as opposed to PMOLED control, which sequentially controls each row and line in the display one at a time. OLED differs from LED, which is built on a p-n diode structure, in essential ways. Doping is used in LEDs to create p- and n-regions by altering the host semiconductor's conductivity. P-n structures are not used in OLEDs. By directly altering the quantum-mechanical optical recombination rate, doping of OLEDs is employed to boost radiative efficiency. Additionally, the wavelength of photon emission is determined by doping.

OLED displays don't require backlights because they produce their own visible light. As a result, it may show deep black depths and be more compact and lightweight than a liquid crystal display (LCD). An OLED panel can attain a higher contrast ratio than an LCD in low ambient light situations (like a dark room), regardless of whether the LCD employs cold cathode fluorescent lights or an LED backlight. Like LCDs, OLED screens are produced in the same manner.

The display is coated with hole injection, transport, and blocking layers as well as electroluminescent material after the first two layers, after which ITO or metal may be applied again as a cathode, and later the entire stack of materials is encapsulated. This is in contrast to TFT (Thin-film transistors for active matrix displays), addressable grid (for passive matrix displays), or indium-tin oxide (ITO) segment (for segment displays). The anode, which may be constructed of ITO or metal, is connected to or used as the TFT layer, addressable grid, or ITO segments. OLEDs can be made transparent and flexible, and both types of displays are employed in foldable smartphones and smartphones with optical fingerprint scanners.

Fingerprint Scanner

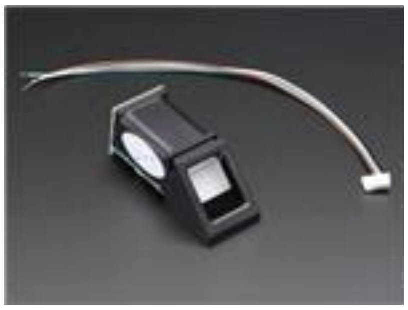
Figure 3: Fingerprint scanning module

Biometric security systems include fingerprint scanners. They are utilized in security businesses, police stations, cell phones, and other mobile devices. Everybody's fingers have distinct patterns of friction ridges, and it is this pattern that is referred to as the fingerprint. Fingerprints are distinctively detailed, reliable across a person's lifetime, and challenging to change. Fingerprints have developed into the best method of identification because there are so many possible combinations. The four different types of fingerprint scanners are optical, capacitance, ultrasonic, and thermal ones [4]. Every sort of scanner's fundamental job is to capture an image of a user's fingerprint and look up a match in its database. Dots per inch are used to gauge how well an image of a fingerprint is captured (DPI). They are categorized as:

Using a digital camera, optical scanners capture a visual image of the fingerprint. Capacitive or CMOS scanners create an image of the fingerprint using capacitors and electrical current. The precision of these scanners frequently exceeds industry standards. The epidermal (outer) layer of the skin is penetrated by high frequency sound waves used in ultrasonic fingerprint scanners. Thermal scanners detect temperature variations between the valleys and ridges of a fingerprint on the contact surface.

The stationary fingerprint scanner and the mobile fingerprint scanner are the of two construction types. One is stagnant: The little scanning area must be dragged by the finger. The movable form is more reliable and more expensive than this. When the finger is not dragged over the scanning area at a steady speed, imaging may not be as good as it could be. Another is moving i.e. the finger is moving while the scanner is running underneath it. Imaging is better since the scanner scans the fingerprint at a steady speed.

Relay Module

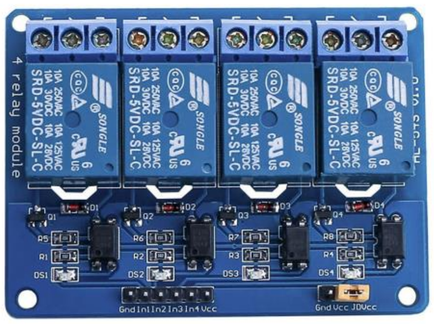

Figure 4: 4-ch Relay module

A relay is an electromagnetic switch that has an activation coil which is energized when a current is passed through it, forming an electromagnet. When triggered, the magnetic field formed pulls the switch towards the contacts and closes the circuit. The relay module consists of 4 such relays, forming a module.

The trigger circuit has a built-in transistor and flyback diode in series with each relay's activation coil. Therefore, pi only needs to provide each trigger with logic signals in order to control the relays. Relays are used in applications where the logic circuit of the microcontroller cannot provide enough power to the external circuitry or in cases when the external circuitry is connected to a higher voltage power source such as 220V household alternating current circuits. The relay modules are used to turn actuators in the system like pumps locks & fans on and off according to the thresholds set in the written algorithms.

In this specific project the actuator i.e., Magnetic lock is driven by 12V DC via the relay module.

3 Methodology

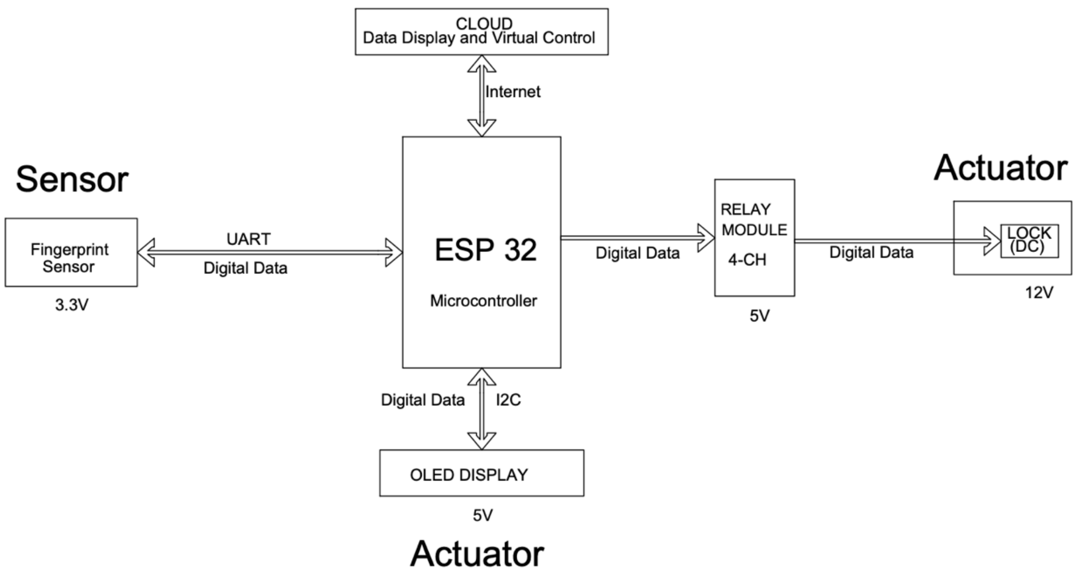

Figure 5: System block diagram

3.1 Block Diagram

The fingerprint sensor is placed in a suitable location and is connected to the ESP32 microcontroller. Once data is fetched by the ESP, it is to be processed and stored by the program inside. When enrolling someone for the first time via cloud, only storage is done, but when someone is trying to gain access, the fingerprint information is compared against existing fingerprints in the database and if verified, the relay is triggered which triggers the Lock's actuator and unlocks the door for a set time, after which it is locked again. The locking and unlocking mechanism can also be done via the cloud. During this, the status of the lock is sent to the OLED display & the Cloud for status monitoring purposes.

3.2 Decision Making Process

The fingerprint sensor is always ready to check the fingerprint placed on the sensor, each and every process of fingerprint collection is displayed on the OLED display. If the fingerprint placed matches the stored fingerprints template, then the magnetic door lock is unlocked for 10 seconds and locks again. If the fingerprint fails to match then the same process is started again unless there are commands coming from IOT cloud. There are two main messages coming from the cloud platform, one is enroll message the other is lock/unlock message. Lock/unlock message is simply used to lock or unlock the lock based on user's desire. Enroll message is used to enroll a user if needed. After the enrolment, user can simply unlock lock by placing their fingerprint. Through both of the process, the number of people enrolled is updated to cloud on a timely basis. The system runs on a loop so after completion of each commands the system starts over and over again.

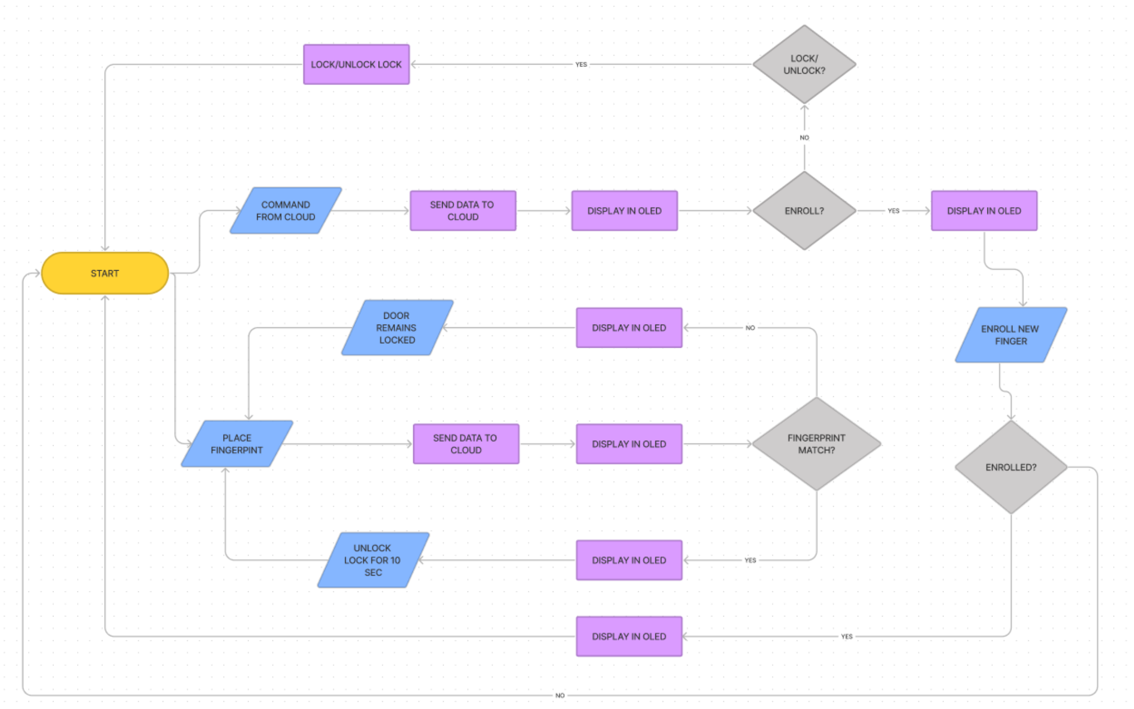

Figure 6: Functional Flowchart

4 System Design

The hardware is comprised of 6 parts, sensing, processing, storing and wireless communication, power supply and actuation unit. The schematic diagram details out the hardware design as well as emulates its final implementation.

3.1 Schematic Design for hardware components

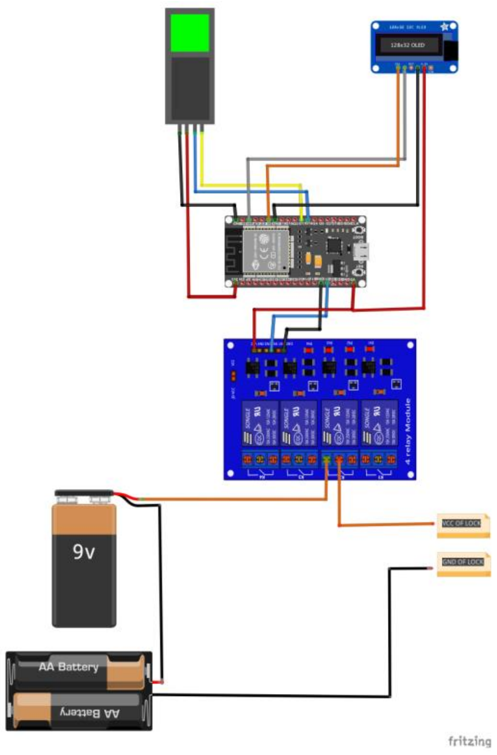

Figure 7: Hardware configuration

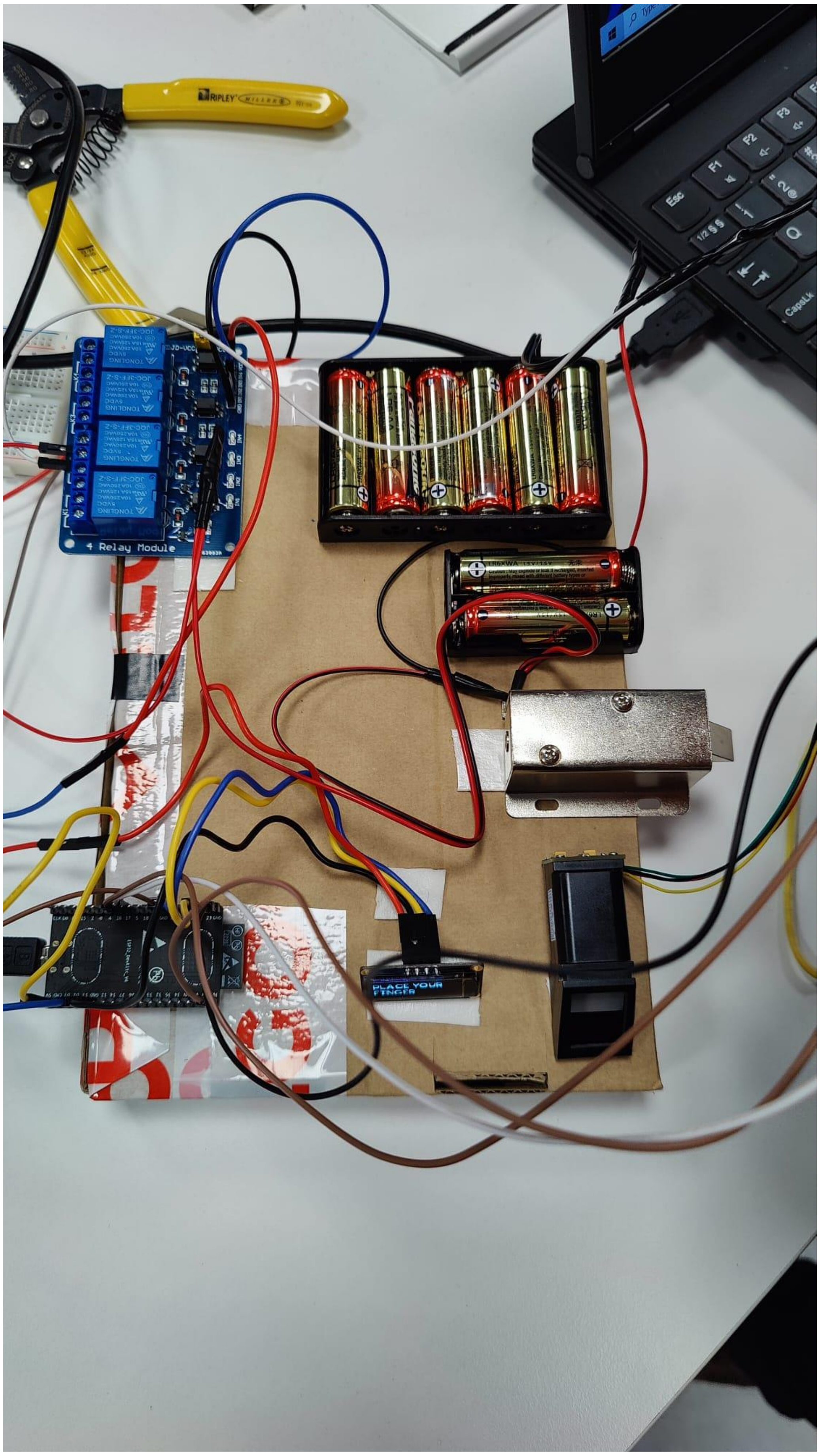

Figure 8: Real Hardware Implementation

3.2 Communication with Cloud

In this project, for communication with cloud, the ESP32’s built-in Wi-Fi module is used to transfer data over the 2.4GHz Wi-Fi frequency band. A direct internet connection using an Ethernet connection is possible to set up through an ethernet to micro-USB converter but this is not necessarily a solution to anything significant. For this specific project mobile hotspot was created and used. The system continuously communicates with the could via the written program and provides remote monitoring capabilities. The cloud based user interface is designed like belows.

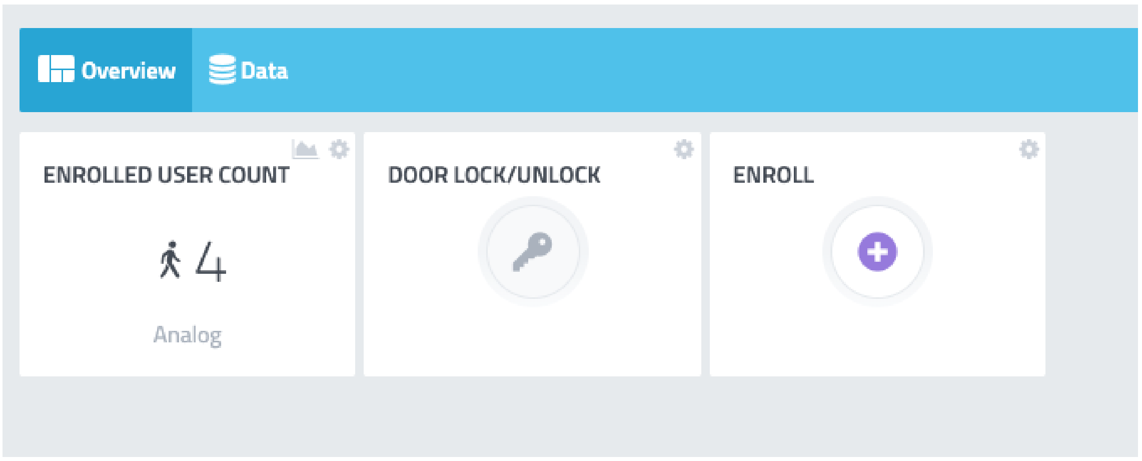

Figure 9: Cloud based user interface

3.3 System operation details

The system operation starts at the sensor as always. The sensor communicates with the ESP via UART (Universal asynchronous receiver-transmitter), which is a serial communication protocol required for reliably communicating data. The fingerprint scanner is capable of sending data at a maximum baud rate of 576000, which means that if we consider 1 baud = 1 bit per second, the data transfer rate will be 70.3125 Kilobytes per second at max. The sensor itself is a power-hungry device since it has so many tiny optical devices in the scanning surface. Its power requirement can go up to 0.9W while drawing a maximum possible current of 150mA. The imaging time is <1s and it has a false positive allowance rate of 0.001%. The imaging sensor can itself store 162 different fingerprint templates to compare against. For most applications this is enough, but to make sure that it is reliable as the number of enrolled fingers goes up, the data is also stored in the ESP.

The ESP32 then receives the data & stores it. For the processing part, the program, which is written in C++, first runs checks to see setup the fingerprint sensor by clearing all pixels in the memory frame buffer, which holds optical information about any other objects that might have been in contact with the sensing surface, generally dust or similar small particulate matter. This is done every time before the scanning process. After the device powers on, a prompt is sent through the OLED display which prompts users to place their fingers. Once a finger in placed, the program starts scanning it against existing fingerprints and if matched, access is granted and simultaneously, the relay is tripped which activates the actuator of the lock. If not, the relay isn’t tripped and a prompt is sent to the user saying that it’s an invalid finger.

The data is communicated by transmitting it to the cloud, where the current status, i.e., locked/unlocked state is viewable. The system can be also programmed to respond with an alert to the administrator if someone tries to scan their fingers multiple times, although this is not what an intruder would do. Therefore, for a better solution, it is recommended that the administrator install an extra sensor that detects uneven strong vibration in the sensor module or the actuator itself, which likely happen when they are being cut or being struck by a hammer in an impulse. Through the cloud, the admin can also send out requests to open the lock without fingerprint access and control locking and unlocking mechanism.

5 Results & Conclusion

The resulting system is a reliable, self-developed and efficient security lock system. It is able to both lock and unlock itself without any need for any manual intervention. The status of the lock is continuously monitored and the administrator is notified of any possible mishaps immediately. The resulting system is also controllable from the IoT based platform for both locking/unlocking and adding users remotely. The user experience is extremely simple and pleasant as the fingerprint sensor is an extremely functional tool with a sense of amusement. The amusement part comes from the lighting of the sensor itself, which makes users feel like it is a futuristic technology. This sparks a lot of curious conversations, especially within institutions which can be a really good aspect for the morale of the people. An unanticipated benefit.

Besides the simple implementation, security locks as such, when used in multiple quantities can help to develop a form of security clearance level system where there are multiple rooms within a given building but not everyone has the authority to access all of them. The use-cases of such a system are endless.

References

1. *ESP32* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/ESP32 (Accessed: December 23, 2022).

2. *OLED* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/OLED (Accessed: December 23, 2022).

3. *4CH Relay* (no date) *Scribd*. Scribd. Available at: https://www.scribd.com/document/385424532/4Ch-relay (Accessed: December 17, 2022).

4. *Fingerprint scanner* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Fingerprint_scanner (Accessed: December 23, 2022).

Appendix A: Source Code

```
#include <Adafruit_Fingerprint.h>

#include <Arduino.h>
#include <U8g2lib.h>
#include <Wire.h>
#include "WiFi.h"
#define CAYENNE_PRINT Serial    // Comment this out to disable prints and save space
#include <CayenneMQTTESP32.h> // Change this to use a different communication device. See
Communications examples.

// Cayenne authentication info. This should be obtained from the Cayenne Dashboard.
char username[] = "4459c180-724b-11ed-8d53-d7cd1025126a";
char password[] = "c893d40402e5d2b5d6982389bdbf93a7586ee038";
char clientID[] = "4f92fc60-724b-11ed-8d53-d7cd1025126a";



#if (defined(__AVR__) || defined(ESP8266)) && !defined(__AVR_ATmega2560__)
  SoftwareSerial mySerial(2, 3);

#else
  #define mySerial Serial2

#endif



// Use Virtual Channel 5 for uptime display.

#define VIRTUAL_CHANNEL 5

char ssid[] = "Redmi Note 10 5G";
char wifiPassword[] ="123Buddha";

Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);
int status = 2;
```

```
uint8_t id;

U8G2_SSD1306_128X32_UNIVISION_F_HW_I2C  u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);   //
M0/ESP32/ESP8266/mega2560/Uno/Leonardo

void setup()
{
  Serial.begin(115200);  ///initializing serial communication between esp32 and PC

  Cayenne.begin(username, password, clientID, ssid, wifiPassword); //initializing cloud

  pinMode(13, OUTPUT);   //initializing gpio pin modes to use relay
  digitalWrite(13, HIGH);

  u8g2.begin();          //intializing i2c display
  u8g2.setFontPosTop();   /**When you use drawStr to display strings, the default criteria is to display
the lower-left
     * coordinates of the characters.The function can be understood as changing the coordinate position
to the upper left
     * corner of the display string as the coordinate standard.*/

  while (!Serial);    //initializing serial communication between fingerprint sensor and esp32
  delay(100);
  Serial.println("\n\nAdafruit Fingerprint intialization");

  // set the data rate for the sensor serial port
  finger.begin(57600);

  if (finger.verifyPassword())
  {
    Serial.println("Found fingerprint sensor!!!");
  }
  else
  {
    Serial.println("Did not find fingerprint sensor!!!");
  }
}
```

```
uint8_t readnumber(void)
{
  uint8_t num = 0;
  while (num == 0)
  {
    while (! Serial.available());
    num = Serial.parseInt();
  }
  return num;
}

void loop()                  // run over and over again
{
  Cayenne.loop();

  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "PLACE YOUR");
  u8g2.drawStr(0, 20, "FINGER");
  u8g2.sendBuffer();
  delay(100);

  getFingerprintID();
  delay(50);

}

uint8_t getFingerprintID()
{
  uint8_t p = finger.getImage();
  switch (p)
  {
    case FINGERPRINT_OK:
      u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
      u8g2.setFont(u8g2_font_heavybottom_tr);
      u8g2.drawStr(0, 0, "IMAGE");
      u8g2.drawStr(0, 20, "TAKEN");
      u8g2.sendBuffer();
      delay(2000);
      break;
    case FINGERPRINT_NOFINGER:
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      return p;
    case FINGERPRINT_IMAGEFAIL:
      return p;
    default:
      return p;
  }

// OK success!

  p = finger.image2Tz();
  switch (p)
  {
    case FINGERPRINT_OK:
      u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
      u8g2.setFont(u8g2_font_heavybottom_tr);
      u8g2.drawStr(0, 0, "IMAGE");
      u8g2.drawStr(0, 20, "CONVERTED");
      u8g2.sendBuffer();
      delay(2000);
      break;
    case FINGERPRINT_IMAGEMESS:
      u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
      u8g2.setFont(u8g2_font_heavybottom_tr);
      u8g2.drawStr(0, 0, "IMAGE");
      u8g2.drawStr(0, 20, "MESSY");
      u8g2.sendBuffer();
      delay(2000);
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      return p;
    case FINGERPRINT_FEATUREFAIL:
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      return p;
    default:
      return p;
  }

// OK converted!
  p = finger.fingerSearch();
  if (p == FINGERPRINT_OK)
  {
    Serial.println("Found a print match!");
  }
  else if (p == FINGERPRINT_PACKETRECIEVEERR)
  {
    //Serial.println("Communication error");
    return p;
  }
  else if (p == FINGERPRINT_NOTFOUND)
  {
    Serial.println("Did not find a match");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "NO MATCH");
    u8g2.drawStr(0, 20, "FOUND");
    u8g2.sendBuffer();
    delay(2000);
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "TRY");
    u8g2.drawStr(0, 20, "AGAIN");
    u8g2.sendBuffer();
    delay(2000);
    return p;
  }
  else
  {
    //Serial.println("Unknown error");
  return p;
}

// found a match!
Serial.print("Found ID #");
Serial.print(finger.fingerID);
char cstr[16];
itoa(finger.fingerID, cstr, 10);
Serial.println();
u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
u8g2.setFont(u8g2_font_heavybottom_tr);
u8g2.drawStr(0, 0, "FOUND ID");
u8g2.drawStr(0, 20, cstr);
u8g2.sendBuffer();
delay(2000);
if (finger.fingerID == 2)
{
  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "WELCOME");
  u8g2.drawStr(0, 20, "SIDDHARTHA");
  u8g2.sendBuffer();
  digitalWrite(13, LOW);
  delay(10000);
  digitalWrite(13, HIGH);
}

else if (finger.fingerID == 1)
{
  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "WELCOME");
  u8g2.drawStr(0, 20, "SUBASH");
  u8g2.sendBuffer();
  digitalWrite(13, LOW);
  delay(10000);
  digitalWrite(13, HIGH);
}
```

```
  else if (finger.fingerID == 3)
  {
    u8g2.clearBuffer();     // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "WELCOME");
    u8g2.drawStr(0, 20, "SULAV");
    u8g2.sendBuffer();
    digitalWrite(13, LOW);
    delay(10000);
    digitalWrite(13, HIGH);
  }

  else
  {
    u8g2.clearBuffer();     // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "WELCOME");
    u8g2.sendBuffer();
    digitalWrite(13, LOW);
    delay(10000);
    digitalWrite(13, HIGH);
  }
  return finger.fingerID;
}



uint8_t getFingerprintEnroll()
{
  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #");
  Serial.println(id);
  u8g2.clearBuffer();     // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "PLACE VALID");
  u8g2.drawStr(0, 20, "FINGER");
  u8g2.sendBuffer();
  delay(1000);
```

```
 while (p != FINGERPRINT_OK)
 {
   p = finger.getImage();
   switch (p)
   {
   case FINGERPRINT_OK:
     Serial.println("Image taken");
     u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
     u8g2.setFont(u8g2_font_heavybottom_tr);
     u8g2.drawStr(0, 0, "IMAGE");
     u8g2.drawStr(0, 20, "TAKEN");
     u8g2.sendBuffer();
     delay(2000);
     break;
   case FINGERPRINT_NOFINGER:
     u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
     u8g2.setFont(u8g2_font_heavybottom_tr);
     u8g2.drawStr(0, 0, "NO FINGER");
     u8g2.drawStr(0, 20, "PLACED");
     u8g2.sendBuffer();
     delay(2000);
     //Serial.println(".");
     break;
   case FINGERPRINT_PACKETRECIEVEERR:
     //Serial.println("Communication error");
     break;
   case FINGERPRINT_IMAGEFAIL:
     //Serial.println("Imaging error");
     break;
   default:
     //Serial.println("Unknown error");
     break;
   }
 }

// OK success!

 p = finger.image2Tz(1);
```

```
switch (p)
{
  case FINGERPRINT_OK:
    Serial.println("Image converted");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "IMAGE");
    u8g2.drawStr(0, 20, "CONVERTED");
    u8g2.sendBuffer();
    delay(2000);
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "IMAGE");
    u8g2.drawStr(0, 20, "MESSY");
    u8g2.sendBuffer();
    delay(2000);
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    //Serial.println("Communication error");
    return p;
  case FINGERPRINT_FEATUREFAIL:
    //Serial.println("Could not find fingerprint features");
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    //Serial.println("Could not find fingerprint features");
    return p;
  default:
    //Serial.println("Unknown error");
    return p;
}

Serial.println("Remove finger");
u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
u8g2.setFont(u8g2_font_heavybottom_tr);
u8g2.drawStr(0, 0, "REMOVE");
```

```
u8g2.drawStr(0, 20, "FINGER");
u8g2.sendBuffer();
delay(4000);
p = 0;
while (p != FINGERPRINT_NOFINGER)
{
  p = finger.getImage();
}
Serial.print("ID "); Serial.println(id);
p = -1;
Serial.println("Place same finger again");
u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
u8g2.setFont(u8g2_font_heavybottom_tr);
u8g2.drawStr(0, 0, "PLACE FINGER");
u8g2.drawStr(0, 20, "AGAIN");
u8g2.sendBuffer();
delay(1000);
while (p != FINGERPRINT_OK)
{
  p = finger.getImage();
  switch (p)
  {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
      u8g2.setFont(u8g2_font_heavybottom_tr);
      u8g2.drawStr(0, 0, "IMAGE");
      u8g2.drawStr(0, 20, "TAKEN");
      u8g2.sendBuffer();
      delay(2000);
      break;
    case FINGERPRINT_NOFINGER:
      //Serial.print(".");
      u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
      u8g2.setFont(u8g2_font_heavybottom_tr);
      u8g2.drawStr(0, 0, "NO FINGER");
      u8g2.drawStr(0, 20, "PLACED");
      u8g2.sendBuffer();
```

```
         delay(2000);
         break;
       case FINGERPRINT_PACKETRECIEVEERR:
         //Serial.println("Communication error");
         break;
       case FINGERPRINT_IMAGEFAIL:
         //Serial.println("Imaging error");
         break;
       default:
         //Serial.println("Unknown error");
         break;
     }
   }

   // OK success!

   p = finger.image2Tz(2);
   switch (p)
   {
     case FINGERPRINT_OK:
       Serial.println("Image converted");
       u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
       u8g2.setFont(u8g2_font_heavybottom_tr);
       u8g2.drawStr(0, 0, "IMAGE");
       u8g2.drawStr(0, 20, "CONVERTED");
       u8g2.sendBuffer();
       delay(2000);
       break;
     case FINGERPRINT_IMAGEMESS:
       Serial.println("Image too messy");
       u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
       u8g2.setFont(u8g2_font_heavybottom_tr);
       u8g2.drawStr(0, 0, "FINGER");
       u8g2.drawStr(0, 20, "MESSY");
       u8g2.sendBuffer();
       delay(2000);
       return p;
     case FINGERPRINT_PACKETRECIEVEERR:
       //Serial.println("Communication error");
       return p;
     case FINGERPRINT_FEATUREFAIL:
       //Serial.println("Could not find fingerprint features");
       return p;
     case FINGERPRINT_INVALIDIMAGE:
       //Serial.println("Could not find fingerprint features");
       return p;
     default:
       //Serial.println("Unknown error");
       return p;
   }

   // OK converted!
   Serial.print("Creating model for #");
   Serial.println(id);
   u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
   u8g2.setFont(u8g2_font_heavybottom_tr);
   u8g2.drawStr(0, 0, "CREATING");
   u8g2.drawStr(0, 20, "createModel");
   u8g2.sendBuffer();
   delay(2000);

   p = finger.createModel();
   if (p == FINGERPRINT_OK)
   {
     Serial.println("Prints matched!");
     u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
     u8g2.setFont(u8g2_font_heavybottom_tr);
     u8g2.drawStr(0, 0, "FINGER");
     u8g2.drawStr(0, 20, "MATCHED");
     u8g2.sendBuffer();
     delay(2000);
   }
   else if (p == FINGERPRINT_PACKETRECIEVEERR)
   {
     //Serial.println("Communication error");
     return p;
```

```
}
else if (p == FINGERPRINT_ENROLLMISMATCH)
{
  Serial.println("Fingerprints did not match");
  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "FINGER DID");
  u8g2.drawStr(0, 20, "NOT MATCHED");
  u8g2.sendBuffer();
  delay(2000);
  return p;
}
else
{
  //Serial.println("Unknown error");
  return p;
}

Serial.print("ID ");
Serial.println(id);
p = finger.storeModel(id);
if (p == FINGERPRINT_OK)
{
  Serial.println("Stored!");
  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "FINGER");
  u8g2.drawStr(0, 20, "STORED");
  u8g2.sendBuffer();
  delay(2000);
}
else if (p == FINGERPRINT_PACKETRECIEVEERR)
{
  Serial.println("Communication error");
  u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
  u8g2.setFont(u8g2_font_heavybottom_tr);
  u8g2.drawStr(0, 0, "FINGER NOT");
  u8g2.drawStr(0, 20, "STORED");
```

```
    u8g2.sendBuffer();
    delay(2000);
    return p;
  }
  else if (p == FINGERPRINT_BADLOCATION)
  {
    Serial.println("Could not store in that location");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "FINGER NOT");
    u8g2.drawStr(0, 20, "STORED");
    u8g2.sendBuffer();
    delay(2000);
    return p;
  }
  else if (p == FINGERPRINT_FLASHERR)
  {
    Serial.println("Error writing to flash");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "FINGER NOT");
    u8g2.drawStr(0, 20, "STORED");
    u8g2.sendBuffer();
    delay(2000);
    return p;
  }
  else
  {
    Serial.println("Unknown error");
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "FINGER NOT");
    u8g2.drawStr(0, 20, "STORED");
    u8g2.sendBuffer();
    delay(2000);
    return p;
  }
  return true;
}



CAYENNE_IN(4)   //locking and unlocking function from cloud
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  if (getValue.asInt() == 1)
  {
    digitalWrite(13, LOW);
  }
 else if (getValue.asInt() == 0)
  {
    digitalWrite(13, HIGH);
  }

}

CAYENNE_IN(6)  //enrolling function from cloud
{
  CAYENNE_LOG("Channel %u, value %s", request.channel, getValue.asString());
  if (getValue.asInt() == 1)
  {
    u8g2.clearBuffer();    // Clears all pixel in the memory frame buffer.
    u8g2.setFont(u8g2_font_heavybottom_tr);
    u8g2.drawStr(0, 0, "ENROLL YOUR");
    u8g2.drawStr(0, 20, "FINGER");
    u8g2.sendBuffer();
    delay(2000);
    Serial.println("Ready to enroll a fingerprint!");
    Serial.println("Please type in the ID # (from 1 to 127) you want to save this finger as...");
    finger.getTemplateCount();
    Serial.println(finger.templateCount);
    id = finger.templateCount + 1;
    if (id == 0)
    {
      return;  // ID #0 not allowed, try again!
    }
```

```
        char cstr[16];
        itoa(finger.fingerID, cstr, 10);
        Serial.print("Enrolling ID #");
        u8g2.clearBuffer();     // Clears all pixel in the memory frame buffer.
        u8g2.setFont(u8g2_font_heavybottom_tr);
        u8g2.drawStr(0, 0, "ENROLL ID:");
        u8g2.drawStr(0, 20, cstr);
        u8g2.sendBuffer();
        delay(2000);

        while (! getFingerprintEnroll() );
    }
}

CAYENNE_OUT(VIRTUAL_CHANNEL)  //displaying number of user in cloud
{
    CAYENNE_LOG("Send data for Virtual Channel %d", VIRTUAL_CHANNEL);
    // This command writes the device's uptime in seconds to the Virtual Channel.
    finger.getTemplateCount();
    int counterValue =  finger.templateCount;
    Serial.println(counterValue);
    Cayenne.virtualWrite(VIRTUAL_CHANNEL, counterValue);
}
```