

**UNIVERSITY OF INFORMATION TECHNOLOGY AND COMMUNICATIONS VIETNAM – KOREA**

**DEPARTMENT OF COMPUTER SCIENCE**



**EMBEDDED SYSTEM REPORT**

**PROJECT: OPEN DOOR**

**BY DETECTION FACE USING ESP 32 CAM**

Students implement : **MAI VĂN HÒA**

**PHAN VĂN TƯỜNG**

**LÊ ĐỨC HIẾU**

Supervising lecturer : **MASTER.** **NGUYỄN VŨ ANH QUANG**

***Đà nẵng, December 202***

**UNIVERSITY OF INFORMATION TECHNOLOGY AND COMMUNICATIONS VIETNAM – KOREA**

**DEPARTMENT OF COMPUTER SCIENCE**

**EMBEDDED SYSTEM REPORT**

**PROJECT: OPEN DOOR**

**BY DETECTION FACE USING ESP 32 CAM**

***Đà nẵng, December 20***

**INSTRUCTOR'S COMMENTS**

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

**INTRODUCTION**

Anyone nowadays is concerned about security, whether it is data security or the security of their own home. Digital door locks have grown quite prevalent in recent years as technology has advanced and the use of IoT has increased. A digital lock does not require a physical key to operate, instead relying on Radio-

Frequency Identification (RFID), fingerprint, Face ID, pins, passwords, and other methods to do so. Using these diverse technologies, we have previously built a number of digital door lock applications. In this article, we'll use the Espressif Systems’ Camera (ESP32 CAM) to create an IoT-based Wi-Fi Door Lock system.

The AI-Thinker ESP32-CAM module is a low-cost development board with a micro-SD card port and a small (OmniVision’s) OV2640 camera. It has a built-in Wi-Fi ESP32 S processor with two high-performance 32-bit LX6 CPUs and a 7-stage pipeline architecture.

The Door Security System application uses ESP32 CAM and Internet of Things (IoT) technology to monitor the status of the door. Blynk is a communication protocol that is used between a smartphone and a door lock system. The Door Security System is available on Android and iOS.

**ACKNOWLEDGEMENT**

As students, we would like to express our sincere appreciation to our teacher, Nguyễn Vũ Anh Quang for their invaluable guidance and support throughout the process of creating this report. Their dedication to our learning has been truly inspiring.

We also want to thank each other, our fellow students, for the collaborative effort in sharing ideas and insights. Together, we've created a richer and more comprehensive report.

This endeavor would not have been as successful without the collective contributions of everyone involved. Thank you, Nguyễn Vũ Anh Quang and fellow students, for making this learning experience meaningful and enjoyable.

**TABLE OF CONTENT**

[**Chapter 1: Introduction 1**](#_heading=h.ihv636)

[1.1 Overview 1](#_heading=h.4d34og8)

[1.2 Researchs and results 1](#_heading=h.1hmsyys)

[1.3 Software and Hardware Requirements 2](#_heading=h.5ww1rglb7glj)

[1.3.1 Software Requirements: 2](#_heading=h.41mghml)

[1.3.2 Hardware Requirements: 2](#_heading=h.2grqrue)

[1.4 System Design 3](#_heading=h.vx1227)

[**Chapter 2: ESP 32 Cam And Components 4**](#_heading=h.3fwokq0)

[2.1 Esp32 cam AI thinker 4](#_heading=h.1v1yuxt)

[**Chapter 3: SMTP Gmail 7**](#_heading=h.4f1mdlm)

[3.1 What is SMTP Gmail 7](#_heading=h.2u6wntf)

[3.2 Mail Processing model 7](#_heading=h.19c6y18)

[3.3 Protocol overview 9](#_heading=h.3tbugp1)

[**Chapter 4: Analysis And Design System 10**](#_heading=h.28h4qwu)

[4.1 Overview model of the research system cần sửa địa chỉ 10](#_heading=h.30j0zll)

[4.2 Design system 11](#_heading=h.nmf14n)

[4.3 Results 14](#_heading=h.2qr2h3gwdbst)

[**Chapter 5: Conclusion 15**](#_heading=h.37m2jsg)

[5.1 What we achived 15](#_heading=h.1mrcu09)

[5.2 What will we do in the future 16](#_heading=h.46r0co2)

# Chapter 1: Introduction

## 1.1 Overview

The idea of a Wi-Fi door lock using ESP32 CAM has recently become an important subject in home appliances. Security is a top priority for everyone nowadays, whether it's data security or personal security. Digital door locks have grown quite prevalent in recent years as technology has advanced and the use of IoT has increased. A digital lock does not require a physical key to operate; instead relies on RFID, fingerprint, Face ID, pins, passwords, and other methods to do so. Using these diverse technologies, researchers have previously built a number of digital door lock applications. We used the ESP32 CAM to create a Wi-Fi Door Lock system in this project. The AI-Thinker ESP32 CAM module is a low-cost development board with a micro-SD card port and a small OV2640 camera. It contains a built-in Wi-Fi and Bluetooth chip, as well as two high-performance 32-bit LX6 CPUs and a 7-stage pipeline architecture. Existing models explained ESP32 CAM in depth and demonstrated how to use it to create a Wi-Fi Video Doorbell. Using the ESP32 CAM and Blynk, we created a Face Recognition-based Door Lock System with a Relay module and Solenoid Lock.

## 1.2 Researches and results

\* Method of research:

* Conducted market research and engaged in dialogues with users to clearly define the system's requirements and objectives.
* Selection of Technology and Libraries: Chose the ESP32-CAM AI Thinker and suitable facial recognition libraries that align with the project's requirements.
* System Architecture Design: Designed the system architecture model integrating ESP32-CAM, door module, and security measures.
* Facial Recognition Algorithm Deployment: Developed and deployed a facial recognition algorithm optimized to run directly on ESP32-CAM, considering available resources.
* Development of Control Software:Created control software to communicate with ESP32-CAM and manage the door module.
* Security Optimization:Implemented additional security measures such as data encryption and authentication to ensure the system's safety.
* Deployment and Testing: Deployed the system in a real-world environment and conducted diverse tests to ensure stability and reliability.
* Feedback Collection and Evaluation: Gathered feedback from end users to assess performance and identify necessary improvements.
* Optimization and Expansion:Based on feedback and test results, implemented optimizations and improvements. Explored possibilities for future feature expansion.

\* Results:

* Flexible Security System Built: Successfully created an efficient security solution, reducing reliance on traditional keys and optimizing access management.
* Utilization of ESP32-CAM AI Thinker: Leveraged the direct on-device artificial intelligence processing capabilities of ESP32-CAM, reducing the load on the network and enhancing responsiveness.
* Integration of Advanced Security Measures: Incorporated data encryption and advanced security measures to ensure the safety and integrity of the system.

## 1.3 Software and Hardware Requirements

The main purpose of this project was to develop and create a door lock system that allows users to unlock a door using face recognition via a door camera. We began our investigation by confirming the demand for such a system among possible customers, and then built a door lock system using a customized version of the ESP32 Cam. We use the following software and hardware to build the Wi-Fi door lock system.

### 1.3.1 Software Requirements:

ARDUINO Integrated Development Environment (IDE). This is a cross-platform application (for Windows, Mac OS X, and Linux) written in C and C++ functions. It's used to write and upload programs to Arduino-compatible boards, as well as other vendor development boards with the support of third-party cores.

### 1.3.2 Hardware Requirements:

• ESP32-CAM

• Electronic door lock 12V

• Breadboard

• 7805 voltage Regulator (5V)

• 10k Resistors (2no.)

• Capacitor 220uF

• Push-button

• 12V DC adapter

• Future Technology Devices International Limited’s FTDI232 USB to Serial Interface

This FTDI232 USB to Serial Interface Board is the core hardware component of this model and the features of this chip are as follows:

• Dual Channel Serial/Parallel Single Chip USB Ports with a variety of configurations

• Entire USB protocol handled on the chip. No USB specific firmware programming required

• FT232BM-style Universal Asynchronous Receiver/Transmitter (UART) interface option with full Handshaking & Modem interface signals

• UART Interface supports 7- or 8-bit data, 1/2 stop bits, and Odd/Even/Mark/Space/No Parity

• Transfer Data Rate 300 to 1 Mega Baud (RS232)

• Transfer Data Rate 300 to 3 Mega Baud (TTL and RS422/RS485)

• “Auto Transmit Enable control” for RS485 serial applications

## 1.4 System Design

The prototype is built using an iterative process that matches the design specifications during the development and implementation phase. We can create and test in repeating sequences by breaking down the design into little bits. New features can be developed and evaluated in each iteration until we have a fully functional system that meets the thesis's goals. From the pilot research, a complete prototype specification is produced and considered. The security problems and the common architecture of IoT systems are taken into account while making design decisions. A suitable microcontroller to serve the SDL's

functionality, wireless devices transmitting a continuous radio signal that can be detected by smart devices (e.g., Smartphones) via a connective protocol (e.g., Bluetooth), a cloud to aid in secure and stable communication, and an API to handle the SDL's functionality were among the preferences.

Wi-Fi Door Lock with ESP32 CAM uses Internet of Things (IoT) technology to monitor the status of the door, control it, and improve home security. Blynk is a communication protocol that is used to increase the security of a home.

We can create and test in repeating sequences by breaking down the design into little bits. New features can be developed and evaluated in each iteration until we have a fully functional system that meets the thesis's goals.

# Chapter 2: ESP 32 Cam And Components

## 2.1 Esp32 cam AI thinker

The ESP32-CAM AI-Thinker module is made up of many elements [5]–[7], including:

• ESP32-S Chip: The module's primary chip, which is utilised for all processing and functionality, has two very fast 32-bit LX6 CPUs and a 7-stage pipeline design.

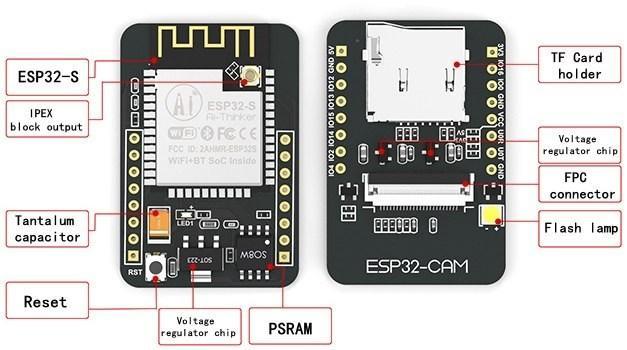
• IPEX block output: The printed IPEX links GSM antennas to the transmitting signal.

• Tantalum capacitor: Small-size modules primarily employ this kind of capacitor. They are strong and provide fine signal quantity power supply filtering.

• Reset button: Pressing this button causes the module's code to be restarted.

• Voltage regulator chip: The module's voltage regulator chip keeps the output voltage constant despite changes in the input supply. The voltage is controlled at 3.3 volts.

• PSRAM: The module has a 4MB low-power pseudo-random access memory for quick instruction processing. It ensures seamless operation of the camera. A micro-SD card holder is included in the ESP32 series to store data. All communication is done through a Serial Peripheral Interface.



FPC connection: The ESP32 module comes with a flexible printed circuit connector that may be used to instal the camera. The dependability of the signal is caused by its fine pitch.

• Flashlight: To enable the camera to take clear pictures, the flash bulb emits electric pulses that act as a flash for the camera. Pinout for ESP32-CAM AI-Thinker The pinout of the ESP32-CAM AI-Thinker Module is described in this section. Although the ESP32-S chip has 34 total pins, only 16 of them are visible to the pinout headers. ESP32 CAM AI TICKER PINOUT

• Power Plugs The module features two positive power supply pins, such as 5V and 3.3V connectors, and three ground pins. The ESP32-CAM AI-Thinker module may be powered by these pins. However, it is not advised to power this development board using a 3.3V pin since it does not provide reliable power.

One power output pin is also provided by the ESP32-CAM, as shown by the colour yellow in the pinout diagram above. This VCC pin may provide either 3.3V or 5V.Jumper connections on the ESP32-CAM indicate that the VCC pin outputs 3.3V.

**Built-in Red LED in GPIO33**

A RED LED is also included in the AI-Thinker board. This LED is located next to the reset button. Connected to GPIO 33 in a logic low active state is this built-in red LED. Therefore, we must push GPIO 33 to a logic low level to turn on the LED. In the same manner, we push GPIO 33 to the active high state to turn off LED.

**UART pins**

The majority of the ESP32-CAM's GPIO pins are multipurpose pins. For the serial transmission and receipt of data for the UART port, respectively, GPIO1 and GPIO3 have alternative uses. There is no onboard programmer included with the AI-Thinker board. To upload the code, programming and communication with the PC are done via these UART pins.

The function of a pin

• GPIO 1 U 0 T X (UART Transmission pin)

• UORXD GPIO3 ( UART Reception pin)

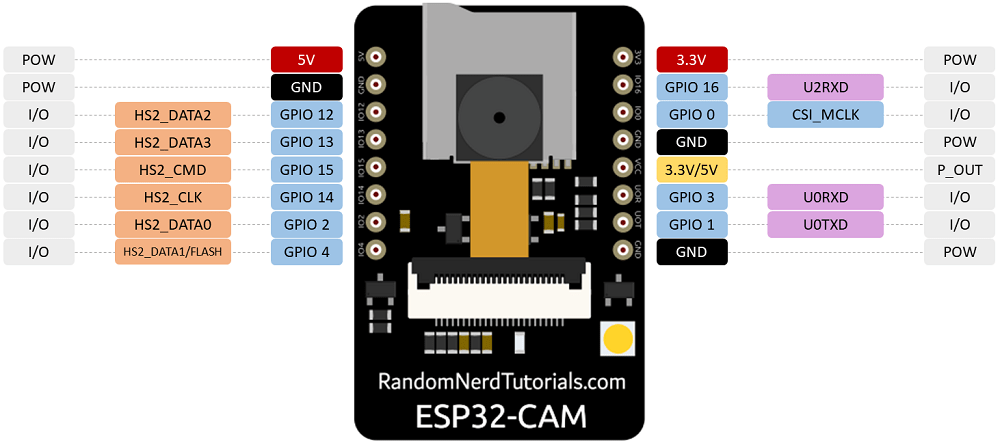
• Flash Mode Selection for GPIO0 Pin Table 3.1-UART Pins

This pin controls the module's mode, which may be either standard or flash mode. GPIO0 is pushed low while in flashing mode, indicating that it has to be connected to the ground. The ESP32-CAM enters flashing mode and allows us to flash the code when GPIO0 is connected to the ground. Disconnecting GPIO0 from the ground will allow the module to operate normally, or enter normal mode, once the code has been flashed to the board.

• GPIO0 is grounded and the ESP32-CAM is in flash mode.

• With the ESP32-CAM in regular programme execution mode, GPIO0 is not connected to the ground.

• the SD Card pins



The ES32-CAM board, which was covered in the section before, features a built-in SD card connection that may be used to attach an SD card. When reading and writing data to SD cards, these GPIO pins are used to interface with micro SD cards. If the SD card is not in use, these GPIO pins may be utilized as regular I/O pins.

Tag Name

Connection to an SD Card Pin GPIO2

Data0 (RTC & ADC supported)

GPIO4/Blinking light

• pin data one (RTC & ADC supported)

• Pin GPIO12 Data2 (RTC & ADC supported)

• Pin GPIO13 Data3 (RTC & ADC supported)

• the GPIO14 CLK (RTC & ADC supported)

• 15 GPIO CMD (RTC & ADC supported)

A high-intensity flashlight is also incorporated into the flash LED pin of the ESP32-CAM. You may use a camera with this flashlight to take pictures in the dark. An inbuilt flash bulb that, if configured to do so, flashes, when the camera takes photographs, is linked to GPIO4. It can be challenging to access both at once since GPIO4 is also linked to the SD card. As a result, while utilizing an SD card, a flashlight may shine inappropriately. The GPIO pin connections for the OV2640 Camera are listed in the table below, along with the variable names we'll be using for each one

.

# Chapter 3: SMTP Gmail

## 3.1 What is SMTP Gmail

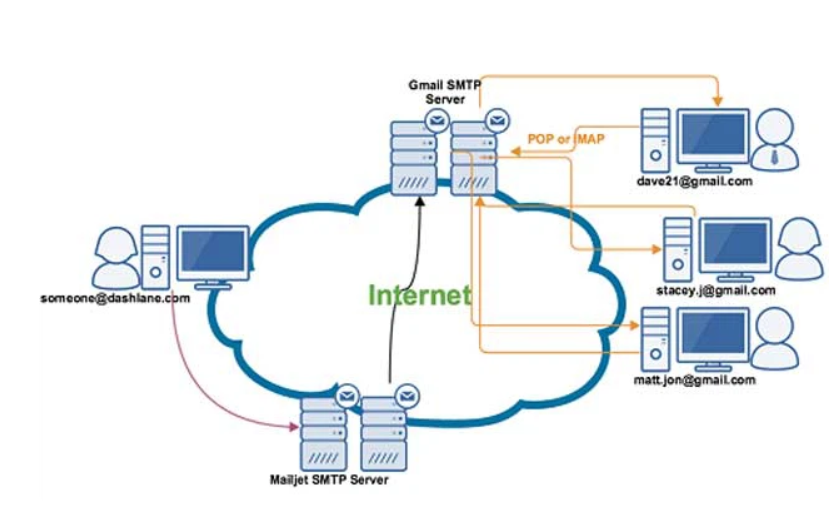
The Simple Mail Transfer Protocol (SMTP) is an [Internet standard](https://en.wikipedia.org/wiki/Internet_Standard) [communication protocol](https://en.wikipedia.org/wiki/Communication_protocol) for [electronic mail](https://en.wikipedia.org/wiki/Email) transmission. Mail servers and other [message transfer agents](https://en.wikipedia.org/wiki/Message_transfer_agent) use SMTP to send and receive mail messages. User-level [email clients](https://en.wikipedia.org/wiki/Email_client) typically use SMTP only for sending messages to a mail server for relaying, and typically submit outgoing email to the mail server on port 587 or 465 per [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [8314](https://datatracker.ietf.org/doc/html/rfc8314). For retrieving messages, [IMAP](https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) (which replaced the older [POP3](https://en.wikipedia.org/wiki/Post_Office_Protocol)) is standard, but proprietary servers also often implement proprietary protocols, e.g., [Exchange ActiveSync](https://en.wikipedia.org/wiki/Exchange_ActiveSync).

SMTP's origins began in 1980, building on concepts implemented on the [ARPANET](https://en.wikipedia.org/wiki/ARPANET) since 1971. It has been updated, modified and extended multiple times. The protocol version in common use today has extensible structure with various extensions for [authentication](https://en.wikipedia.org/wiki/SMTP_Authentication), [encryption](https://en.wikipedia.org/wiki/Email_encryption), binary data transfer, and [internationalized email addresses](https://en.wikipedia.org/wiki/International_email). SMTP servers commonly use the [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) on [port number](https://en.wikipedia.org/wiki/Port_number) 25 (for plaintext) and 587 (for encrypted communications).



## 3.2 Mail Processing model

Email is submitted by a mail client ([mail user agent](https://en.wikipedia.org/wiki/Mail_user_agent), MUA) to a mail server ([mail submission agent](https://en.wikipedia.org/wiki/Mail_submission_agent), MSA) using SMTP on [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) port 587. Most [mailbox providers](https://en.wikipedia.org/wiki/Mailbox_Provider) still allow submission on traditional port 25. The MSA delivers the mail to its mail transfer agent ([mail transfer agent](https://en.wikipedia.org/wiki/Mail_transfer_agent), MTA). Often, these two agents are instances of the same software launched with different options on the same machine. Local processing can be done either on a single machine, or split among multiple machines; mail agent processes on one machine can share files, but if processing is on multiple machines, they transfer messages between each other using SMTP, where each machine is configured to use the next machine as a [smart host](https://en.wikipedia.org/wiki/Smart_host). Each process is an MTA (an SMTP server) in its own right.



The boundary MTA uses [DNS](https://en.wikipedia.org/wiki/Domain_name_system) to look up the [MX (mail exchanger) record](https://en.wikipedia.org/wiki/MX_record) for the recipient's domain (the part of the [email address](https://en.wikipedia.org/wiki/Email_address) on the right of @). The MX record contains the name of the target MTA. Based on the target host and other factors, the sending MTA selects a recipient server and connects to it to complete the mail exchange.

Message transfer can occur in a single connection between two MTAs, or in a series of hops through intermediary systems. A receiving SMTP server may be the ultimate destination, an intermediate "relay" (that is, it stores and forwards the message) or a "gateway" (that is, it may forward the message using some protocol other than SMTP). Per [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [5321](https://datatracker.ietf.org/doc/html/rfc5321) section 2.1, each hop is a formal handoff of responsibility for the message, whereby the receiving server must either deliver the message or properly report the failure to do so.

Once the final hop accepts the incoming message, it hands it to a [mail delivery agent](https://en.wikipedia.org/wiki/Mail_delivery_agent) (MDA) for local delivery. An MDA saves messages in the relevant [mailbox](https://en.wikipedia.org/wiki/Email_mailbox) format. As with sending, this reception can be done using one or multiple computers, but in the diagram above the MDA is depicted as one box near the mail exchanger box. An MDA may deliver messages directly to storage, or [forward](https://en.wikipedia.org/wiki/Email_forwarding) them over a network using SMTP or other protocol such as [Local Mail Transfer Protocol](https://en.wikipedia.org/wiki/Local_Mail_Transfer_Protocol) (LMTP), a derivative of SMTP designed for this purpose.

Once delivered to the local mail server, the mail is stored for batch retrieval by authenticated mail clients (MUAs). Mail is retrieved by end-user applications, called email clients, using [Internet Message Access Protocol](https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) (IMAP), a protocol that both facilitates access to mail and manages stored mail, or the [Post Office Protocol](https://en.wikipedia.org/wiki/Post_Office_Protocol) (POP) which typically uses the traditional [mbox](https://en.wikipedia.org/wiki/Mbox) mail file format or a proprietary system such as Microsoft Exchange/Outlook or [Lotus Notes](https://en.wikipedia.org/wiki/Lotus_Notes)/[Domino](https://en.wikipedia.org/wiki/IBM_Lotus_Domino). [Webmail](https://en.wikipedia.org/wiki/Webmail) clients may use either method, but the retrieval protocol is often not a formal standard.

SMTP defines message *transport*, not the message *content*. Thus, it defines the mail *envelope* and its parameters, such as the [envelope sender](https://en.wikipedia.org/wiki/Envelope_sender), but not the header (except *trace information*) nor the body of the message itself. STD 10 and [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [5321](https://datatracker.ietf.org/doc/html/rfc5321) define SMTP (the envelope), while STD 11 and [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [5322](https://datatracker.ietf.org/doc/html/rfc5322) define the message (header and body), formally referred to as the [Internet Message Format](https://en.wikipedia.org/wiki/Internet_Message_Format).

## 3.3 Protocol overview

SMTP is a [connection-oriented](https://en.wikipedia.org/wiki/Connection-oriented_communication), [text-based protocol](https://en.wikipedia.org/wiki/Text-based_protocol) in which a mail sender communicates with a mail receiver by issuing command strings and supplying necessary data over a reliable ordered data stream channel, typically a [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) connection. An *SMTP session* consists of commands originated by an SMTP [client](https://en.wikipedia.org/wiki/Client_(computing)) (the initiating [agent](https://en.wikipedia.org/wiki/Software_agent), sender, or transmitter) and corresponding responses from the SMTP [server](https://en.wikipedia.org/wiki/Server_(computing)) (the listening agent, or receiver) so that the session is opened, and session parameters are exchanged. A session may include zero or more SMTP transactions. An *SMTP transaction* consists of three command/reply sequences:

1. **MAIL** command, to establish the return address, also called return-path, reverse-path, [bounce address](https://en.wikipedia.org/wiki/Bounce_address), mfrom, or envelope sender.
2. **RCPT** command, to establish a recipient of the message. This command can be issued multiple times, one for each recipient. These addresses are also part of the envelope.
3. **DATA** to signal the beginning of the *message text*; the content of the message, as opposed to its envelope. It consists of a *message header* and a *message body* separated by an empty line. DATA is actually a group of commands, and the server replies twice: once to the *DATA command* itself, to acknowledge that it is ready to receive the text, and the second time after the end-of-data sequence, to either accept or reject the entire message.

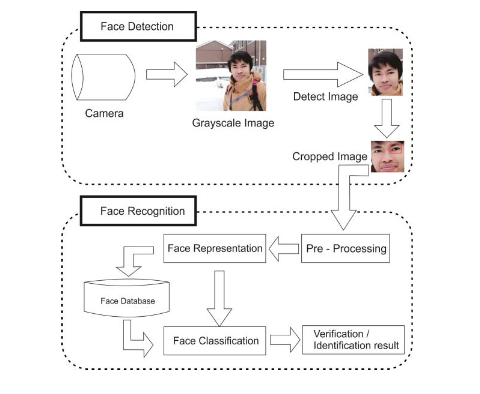
Besides the intermediate reply for DATA, each server's reply can be either positive (2xx reply codes) or negative. Negative replies can be permanent (5xx codes) or transient (4xx codes). A **reject** is a permanent failure and the client should send a bounce message to the server it received it from. A **drop** is a positive response followed by message discard rather than delivery.

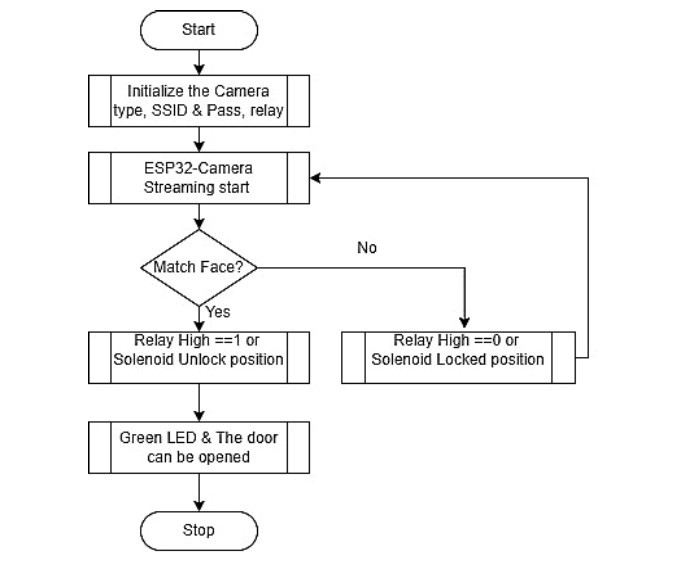
The initiating host, the SMTP client, can be either an end-user's [email client](https://en.wikipedia.org/wiki/Email_client), functionally identified as a [mail user agent](https://en.wikipedia.org/wiki/Mail_user_agent) (MUA), or a relay server's [mail transfer agent](https://en.wikipedia.org/wiki/Mail_transfer_agent) (MTA), that is an SMTP server acting as an SMTP client, in the relevant session, in order to relay mail. Fully capable SMTP servers maintain queues of messages for retrying message transmissions that resulted in transient failures.

# Chapter 4: Analysis And Design System

## 4.1 Overview model of the research system cần sửa địa chỉ

The analytical method in this research is shown in 2 flowcharts in Figure 4.1 and Figure 4.2; the analysis is obtained from the following flowchart. HTTP address in this research uses the address [*http://192.168.146.240/*](http://192.168.146.240/)Moreover, The software used in this research is Wireshark, which specifically analyzes the throughput generated from the communication between the ESP32-Cam and the server.





## 4.2 Design system

As shown in Figure 4.2, the proposed system sets up some requirement for the code, then door lock detect face if the visitor is true, it will set the output is high. After that the door will open and the led tuns on. If not it sets the output is low and nothing happens.

First step: initial libraries, ssid and password

#include "esp\_camera.h"

#include <WiFi.h>

#include <SMTP.h>

#define CAMERA\_MODEL\_AI\_THINKER

#include "camera\_pins.h"

const char\* ssid = "YOUR\_WIFI\_SSID"; // name of wife

const char\* password = "YOUR\_WIFI\_PASSWORD"; // password

SMTPData smtpData;

SMTPClient smtp;

Second step: set up the configuration of the system

void setup() {

  Serial.begin(115200);

  Serial.setDebugOutput(true);

  Serial.println();

  camera\_config\_t config;

  config.ledc\_channel = LEDC\_CHANNEL\_0;

  config.ledc\_timer = LEDC\_TIMER\_0;

  config.pin\_d0 = Y2\_GPIO\_NUM;

  config.pin\_d1 = Y3\_GPIO\_NUM;

  config.pin\_d2 = Y4\_GPIO\_NUM;

  config.pin\_d3 = Y5\_GPIO\_NUM;

  config.pin\_d4 = Y6\_GPIO\_NUM;

  config.pin\_d5 = Y7\_GPIO\_NUM;

  config.pin\_d6 = Y8\_GPIO\_NUM;

  config.pin\_d7 = Y9\_GPIO\_NUM;

  config.pin\_xclk = XCLK\_GPIO\_NUM;

  config.pin\_pclk = PCLK\_GPIO\_NUM;

  config.pin\_vsync = VSYNC\_GPIO\_NUM;

  config.pin\_href = HREF\_GPIO\_NUM;

  config.pin\_sscb\_sda = SIOD\_GPIO\_NUM;

  config.pin\_sscb\_scl = SIOC\_GPIO\_NUM;

  config.pin\_pwdn = PWDN\_GPIO\_NUM;

  config.pin\_reset = RESET\_GPIO\_NUM;

  config.xclk\_freq\_hz = 20000000;

  config.pixel\_format = PIXFORMAT\_JPEG;

  if (psramFound()) {

    config.frame\_size = FRAMESIZE\_UXGA;

    config.jpeg\_quality = 10;

    config.fb\_count = 2;

  } else {

    config.frame\_size = FRAMESIZE\_SVGA;

    config.jpeg\_quality = 12;

    config.fb\_count = 1;

  }

  esp\_err\_t err = esp\_camera\_init(&config);

  if (err != ESP\_OK) {

    Serial.printf("Camera init failed with error 0x%x", err);

    return;

  }

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

    Serial.println("Connecting to WiFi...");

  }

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");

  Serial.print(WiFi.localIP());

  Serial.println("' to connect");

}

Third Step: design the flows of code

void loop() {

  camera\_fb\_t\* fb = NULL;

  fb = esp\_camera\_fb\_get();

  if (fb) {

    // If the face is detected

    if (matchFace == true && activateRelay == false) {

      activateRelay = true;

      prevMillis = millis();

      // send email

      sendEmail();

    }

    if (activateRelay == true && millis() - prevMillis > interval) {

      activateRelay = false;

      matchFace = false;

    }

    esp\_camera\_fb\_return(fb);

  }

}

Next setup: code for send mail using SMTP protocol

void sendEmail() {

  smtpData.setLogin(smtpUsername, smtpPassword);

  if (smtp.connect(smtpServer, smtpPort)) {

    if (smtp.startTLS()) {

      if (smtp.login(smtpUsername, smtpPassword)) {

        if (smtp.beginMessage(emailSender, emailRecipient)) {

          smtp.addRecipient(emailRecipient);

          smtp.setSubject(emailSubject);

          smtp.println(emailText);

          smtp.endMessage();

          Serial.println("Email sent!");

        } else {

          Serial.println("Error starting message!");

        }

        smtp.close();

        return;

      } else {

        Serial.println("Error login!");

      }

    } else {

      Serial.println("Error starting TLS!");

    }

    smtp.close();

  } else {

    Serial.println("Error connecting to SMTP server!");

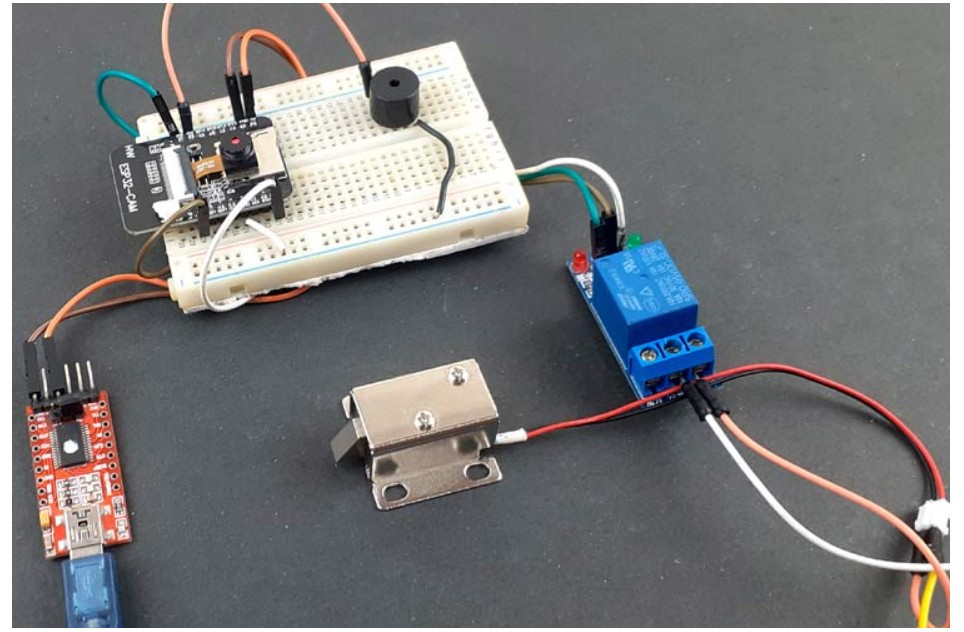
  }

  smtp.close();

}

## 4.3 Results

Results The picture below is to represent the result of this work. In this IOT based working model, we have made a Smart WIFI door lock using ESP32-CAM. In this model, when someone look at the cam, the house owner will get a notification on the mobile. After checking the photo, visitor can unlock the door from a system.



We successfully designed an IoT-based Wi-Fi door lock security system using the ESP32Cam to monitor the status of the door and boost the home security. In this circumstance, due to the current COVID scenario, the smart locking door system is quite important and applying this Wi-Fibased door lock system without using our hands is essential.

Also, our proposed model in this paper can be extended by integrating temperature sensors, which can be used to trigger the system to automatically open and close the doors as per the variations of the room temperature.

# Chapter 5: Conclusion

## 5.1 What we achieved

In this project, we have proposed a security monitoring system based on IoT technology. Our proposed system consists of ESP32 Cam AI Thinker, PCB Board, Step Down 12V to 5V, Adaptor 12V, green LED, two Mosfet,, and Electric Strike door lock. The system we proposed is a system that can detect the face and control the door remotely, receive a warning if a motion is detected near the door, granting door access to trusted people to control the door, get a notification that the door is still open after the limit time has passed.

## 5.2 What will we do in the future

For future research, we plan to explore:

1) the usage of camera, so the user can find out who is visiting;

2) new emergency call feature to directly call the police;

3) new filter feature when see a report;

4) new feature for two-way communication between the owner of the door and the guess

To complete this project, once again, I sincerely thank Master Nguyễn Vũ Anh Quang, who has shown care and provided assistance to us throughout the project. I would also like to express my gratitude to the professors and teachers in the Information Technology department who have helped me complete this foundational project. I genuinely appreciate your support!

**References**

**Cloud-Based Surveillance:**

Kaur, A., Jadli, A., Sadhu, A., Goyal, S., Mehra, A., & Rahul. (2021). "Cloud-Based Surveillance using ESP32 CAM." In *2021 International Conference on Intelligent Technology, System and Service for the Internet of Everything (ITSS-IoE)*, pp. 1-5. DOI: [10.1109/ITSS-IoE53029.2021.9615334](https://doi.org/10.1109/ITSS-IoE53029.2021.9615334).

**Access Point Placement Recommendation:** 2. Mukti, F. S., Adi, P. D. P., Prasetya, D. A., Sihombing, V., Rahanra, N., Yuliawan, K., & Simatupang, J. (2021). "Integrating Cost-231 Multiwall Propagation and Adaptive Data Rate Method for Access Point Placement Recommendation." In *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(4), 2021. DOI: [10.14569/IJACSA.2021.01204943](https://doi.org/10.14569/IJACSA.2021.01204943).

**Face Mask Detection:** 3. Kumar, M. S. H. (2022). "ESP32-CAM for Face Mask Detection." In *International Journal of Advanced Research in Science, Communication, and Technology (IJARSCT)*, February 2022. DOI: [10.48175/IJARSCT-25094](https://doi.org/10.48175/IJARSCT-25094).

**Web Server for Photovoltaic System Monitoring:** 4. Allafi, I., & Iqbal, T. (2017). "Design and implementation of a low-cost web server using ESP32 for real-time photovoltaic system monitoring." In *2017 IEEE Electrical Power and Energy Conference (EPEC)*, pp. 1-5. DOI: [10.1109/EPEC.2017.8286184](https://doi.org/10.1109/EPEC.2017.8286184).

**Secure Smart Locking System:** 5. Venkata, N. Y. L., Rupa, C., Dharmika, B., Nithin, T. G., & Vineela, N. (2021). "Intelligent Secure Smart Locking System using Face Biometrics." In *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pp. 268-273. DOI: [10.1109/RTEICT52294.2021.9573869](https://doi.org/10.1109/RTEICT52294.2021.9573869).

**Security Testing of ESP32 IoT Devices:** 6. Barybin, O., Zaitseva, E., & Brazhnyi, V. (2019). "Testing the Security ESP32 Internet of Things Devices." In *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, pp. 143-146. DOI: [10.1109/PICST47496.2019.9061269](https://doi.org/10.1109/PICST47496.2019.9061269).