"Haven's Light Is Our Guide"

Rajshahi University of Engineering & Technology, Rajshahi

Department of Electronics and Telecommunication Engineering

Project Report

Project Title: Facial Recognition Based Entry Logging and Intrusion Detection with Web Alerts and Panel.

Submitted By:
Mahir Labib Chowdhury
Roll Number: 1604006
Department of Electronics and Telecommunication Engineering
Rajshahi University of Engineering & Technology

Submitted To:
Md. Rakib Hossain
Lecturer
Department of Electronics and Telecommunication Engineering
Rajshahi University of Engineering & Technology

Submission Date: December 20, 2020

"Heaven's Light is Our Guide"
DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING
Rajshahi University of Engineering & Technology, Bangladesh

CERTIFICATE

This is to certify that the project entitled "Facial Recognition Based Entry Logging and Intrusion Detection with Web Alerts and Panel" by Mahir Labib Chowdhury, Roll No. 1604006 has been carried out under my supervision. To the best of my knowledge, this project work is an original one and was not submitted anywhere for any degree or diploma.

Supervisor

.....
(Md. Rakib Hossain)
Lecturer
Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology

Declaration

This is to certify that this project work is my own work and I have not submitted elsewhere for the award of any degree or diploma.

(Mahir Labib Chowdhury)
Roll No. 1604006
Department of Electronics & Telecommunication Engineering
Rajshahi University of Engineering & Technology

Abstract

In the modern world the concept of a smart home and smart facilities are not a distant dream anymore. Ideally, using the smart features offered by the modern era we want to monitor our home or facilities in detail. Not only we want our home or facilities to be secure from the intruders and prevent the theft or damage of our belongings we also want to keep tabs on who is entering and leaving the home or facility at what times. This can be useful for identifying missing elements to solving serious crimes.

Simple 'Access control keypads' and 'Key fobs' alone are not secure enough anymore. Since 'Key fobs' and passwords can easily be stolen. I propose a 'Two Factor Authentication System' for access to the home or facility using a combination of both 'Previous knowledge' (such as passwords) or 'Belongings' (such as key fob) and a 'Biometric Vector' (such as Facial Data). This will ensure the person having the right key is also the right person with the adequate clearance or permission to enter the home or facility. There is one more thing to note, often 'The chain is as strong as itś weakest link', so we have to think about other entry point than the main door. We can add sensors to windows and other potential entry point for intrusion detection.

We have achieved this project goals with a combination of single board computers and micro controllers as the main processing side of the project. Networking elements and data collection points (i.e. cameras and sensors) were also integral part of this project.

Key words: Raspberry Pi, Home Automation, Smart Home, Intrusion Detection, Home Security, Face Recognition, Visitor Monitoring, Entry Logging

Acknowledgement

It's my privilege to express my sincerest regards to my project supervisor, Md. Rakib Hossain, for his valuable input, sensible guidance, encouragement, whole hearted cooperation throughout the duration of my project.
I also take this opportunity to thank all teachers for contributing their valuable comments, suggestions and advices helped to complete the project successfully.

Mahir Labib Chowdhury

Contents

List of Figures

Chapter 1

Introduction

Humans by nature are curious and comfort seeking. We have created technologies to have more control over our lives and make it more easier. Modern technologies have given us the power to do things that was unthinkable just a few years ago. Our home is the safest place to us and we want to keep it protected at any cost. Having 24 hours a day and 7 days a week monitoring on our home keeps our mind free from the tension of a burglar entering our home.
This project aims to have constant monitoring of a home or facility, logging who is entering and leaving at what times and have intrusion alert when someone tries to break in from windows and such. We'll be introducing 'Two Factor Authentication' to our home doors and have full control over the whole system from any where of the world over the internet.

1.1 Motivation

Being worried when away from our beloved home/facility is not a new phenomena. We're always very anxious about our precious belongings whenever they leave our line of sight. This prevents us form concentrating on the present. While intrusion detection systems are readily available on the market, they are built on top of propitiator technologies and should not be trusted with personal data. Also, they run the risk of begin out of service due to their reliance on the major cloud services providers.

1.2 Objectives

1. To entry log people entering and leaving
2. To make sure having awareness of someone breaking in
3. To have a second factor of authentication
4. To avoid being vulnerable by loosing a key

1.3 Report Outline

In chapter 1 the importance of this project along with the motivation behind this project with the objectives has has been discussed.

In chapter 2 we discuss about different components and a high level overview of the project. The high level overview includes functional block diagram and system data flow with decision making.
In chapter 3 we outline the design process and decision making for this project. What parts are used including their usage can be found here.
In chapter 4 we conclude the project, summarizing it and discussing it's limitations and future progress scopes.

Chapter 2

Background and Preliminaries

2.1 Block Diagram

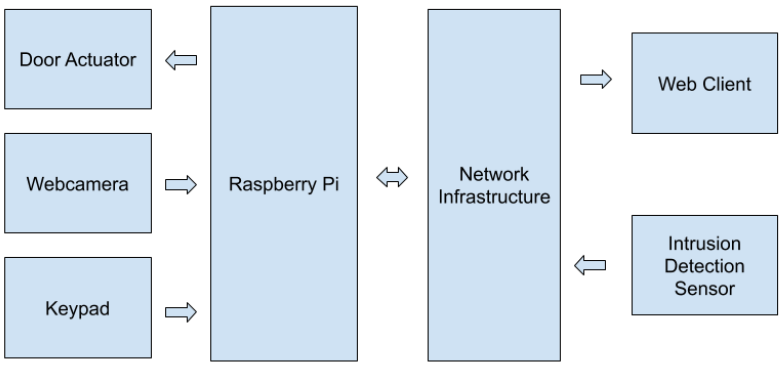

Figure 2.1: Block Diagram of Facial Recognition Based Entry Logging and Intrusion Detection

The proposed system is a combination of interrupt based signal from the sensors and pooling based data from the camera. The block diagram shows how the Raspberry Pi can communicate with the various part of the the system. The intrusion detection sensors are connected by wireless means with the Raspberry Pi. The Networking Infrastructure exposes the Raspberry Pi to the outside world through a secure channel.

2.2 Flow Chart

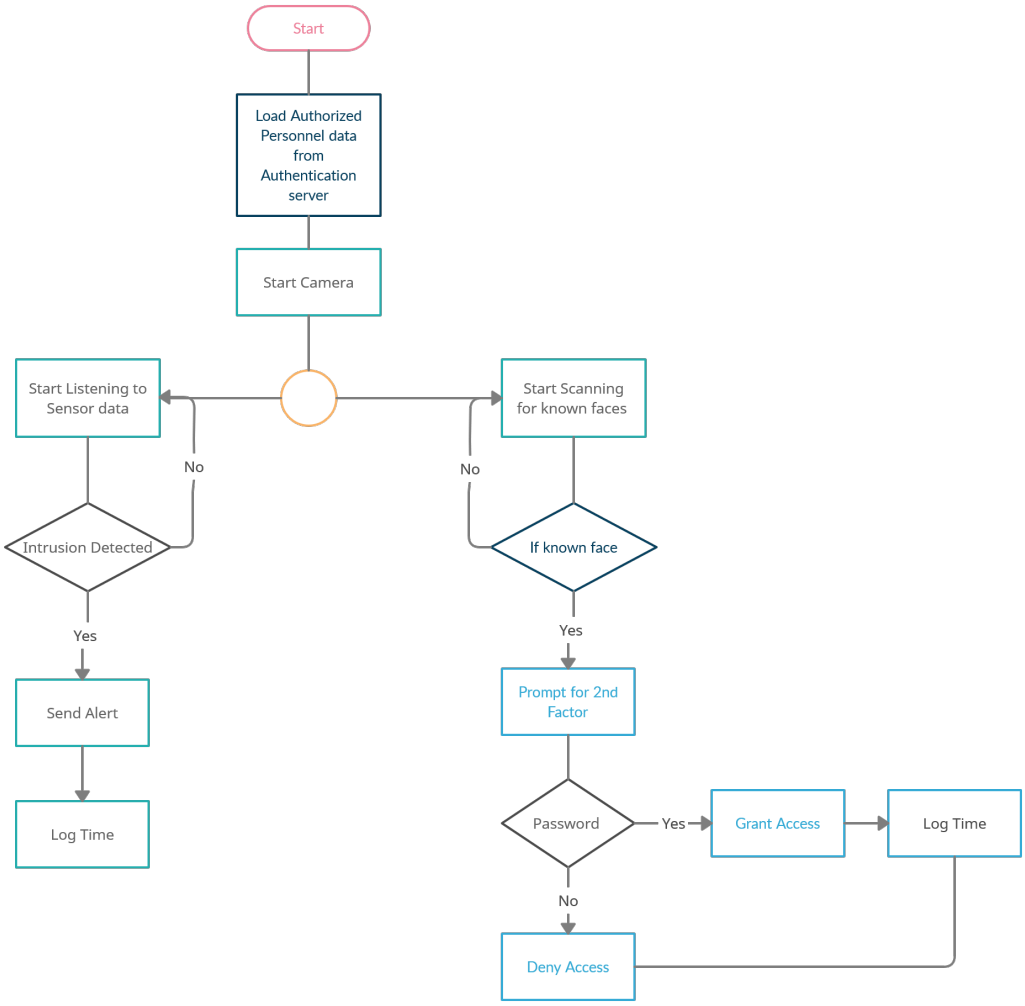

Figure 2.2: Flow Chart of Facial Recognition Based Entry Logging and Intrusion Detection

2.3 Required Components

1. Raspberry Pi 3B(x1)
2. Micro SD Card (x1)
3. Web camera (x1)
4. Keypad (x1)
5. ESP8266 (x1)

6. Magnetic Contact switch (x1)
7. WiFi Router (x1)
8. Power Adapter [5V-2A] (x2)
9. Wires

2.4 Description of Components

2.4.1 Raspberry Pi 3B

Figure 2.3: Raspberry Pi 3B+

Raspberry Pi is the worlds first small single board computer. It has invented the form factor that it still dominates. This small computer has a ARM CPU and GPU with on board robust Networking, the board is also very low power consuming. These characteristics make this board the perfect choice for home Automation and IOT based projects.

Specification:

- CPU Clock Speed: 1.2GHz
- RAM: 1GB
- WiFi: 2.4GHz
- GPIO: 40pins
- Power: 20W
- USB: 4x USB 2.0

2.4.2 Micro SD Card

Figure 2.4: Micro SD Card

Micro SD card is another modern marvel of technology. It's one of the smallest and most reliable data storage medium. For running a Operating system in the Raspberry Pi, we need to boot from the SD card.

Specification:

- Dimensions: $15 * 11 * 1$mm
- Speed Class: 10
- Seq. Write: 10MB/s
- Seq. Read: 15MB/s

2.4.3 Web Camera

Figure 2.5: USB Web Camera

A web camera is essentially a USB camera that can be connected to any computer including the Raspberry Pi. In our project this camera will be used to detect the facial features of the users face.

Specification:

- Resolution: 720p
- USB: USB 3.0
- Focus: Infinite/Fixed
- Mic: Yes
- Framerate: 30FPS
- FOV: 60∘

2.4.4 Keypad

Figure 2.6: Keypad

Keypad can be used to input password to the device. It's a cheap and effective input device. Other than taking the input from the user, this same keyboard can be used to configure the device when first setting up.

Specification:

- Matrix Config: 4X4
- Max Voltage: 24V
- Max Current: 30mA
- Very thin
- Adhesive backing
- Operating Temperature: 0C to 50C

2.4.5 ESP8266

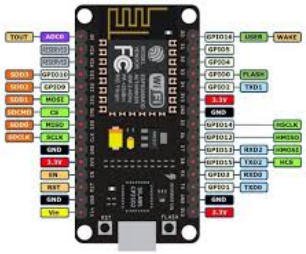

Figure 2.7: ESP8266

A single board micro controller board with on board WiFi capabilities and very low power consumption. Being a micro controller it has very good interface potential and robust connectivity/expandability.

Specification:

- Wi-fi: 802.11 b/g/n, Wi-Fi Direct, soft-AP
- Flash: 4MB
- CPU: 80MHz
- Standby power: 1.0mW

2.4.6 Magnetic Contact Switch

Figure 2.8: Magnetic Contact Switch

A switch that closes contacts when magnets are moved away from the device. This will be responsible for waking up the micro controller and sending the signal.

2.4.7 Wi-Fi Router

Figure 2.9: Wi-Fi Router

Router will be the central part of the Networking Infrastructure. This router will connect the devices internally then will be used to interface with the internet. The firewall on the router will be responsible for securing it from outside threat.

Specification:

- Wi-Fi: 2.4GHz
- Mode: 802.11 b/g/n
- Channel: 11
- Power: 20W
- Uplink: 100Mbps

2.4.8 Power Adapter

Figure 2.10: Power Adapter

Power adapter will be used to power the Raspberry Pi and the ESP8266.

Specification:

- Power: 20W
- Voltage: 5V
- Current: 2A

2.4.9 Wires

Figure 2.11: Wires

Wires will be used to connect the different parts

2.5 Required Software Stack

1. Visual Studio Code
2. EasyEDA
3. OpenCV

2.6 Description of Software Stack

2.6.1 Visual Studio Code

Visual Studio Code is a text editor that can be used to write code/text. This is a very flexible and light weight text editor. All codes and text including this report has been written with Visual Studio Code.

2.6.2 EasyEDA

EasyEDA is a EDA or Electronics Design Automation tool. It can be used to design circuits and run simulations.

2.6.3 OpenCV

OpenCV is a C powered image processing library. It can be used to process both still images and video stream directly from camera. We'll be using it's stream analysis capabilities to find out faces in the FOV of the camera.

Chapter 3

Design and Implementation

3.1 Network Infrastructure

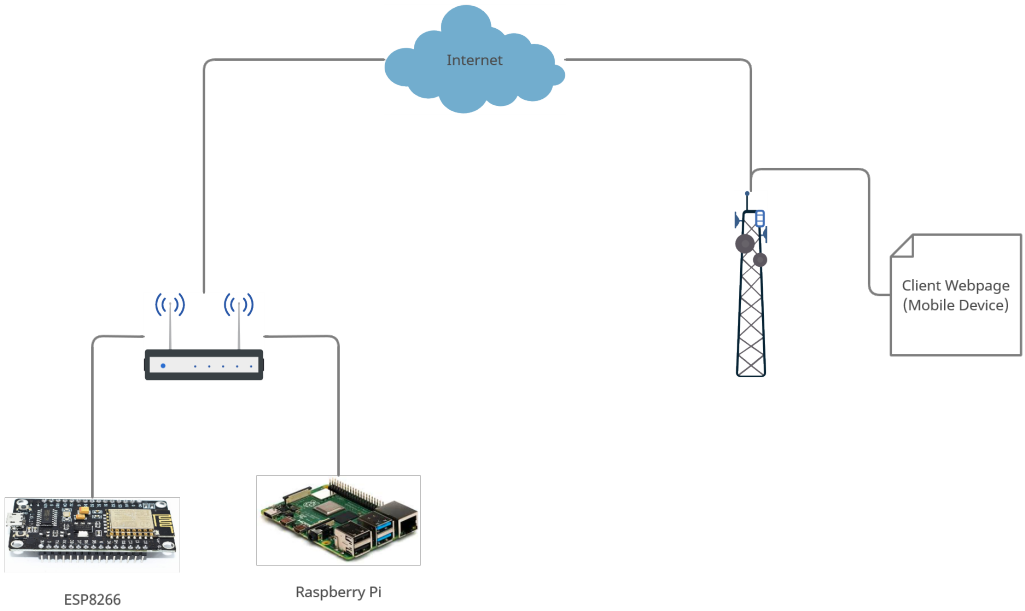

Figure 3.1: Network Infrastructure

The networking layout was designed because it was the most important stack of all. The network infrastructure is not only responsible for connecting the device to one another but also to expose the device to the outside world. The router runs a custom **OpenWRT** based firmware with a **Wireguard VPN** service to make sure no one else can access the private data from our authentication system. Using **NAT**, the router merges the private network to the public internet. On the client device a **Wireguard Configuration** will be installed to connect with the private network.

3.2 Intrusion Detection Sensor

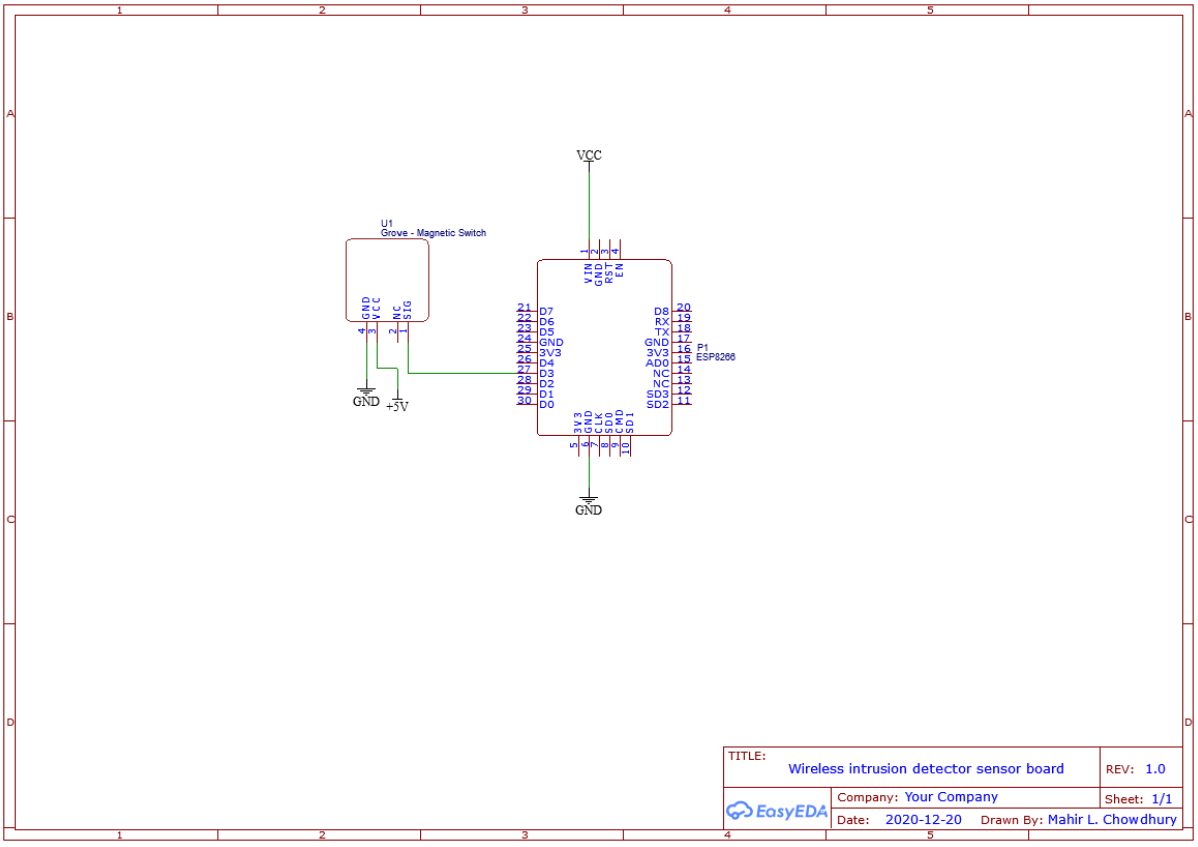

Figure 3.2: Schematic of Intrusion Detection Sensor

The intrusion the detection sensor is in reality a very simple device. It uses a magnetic contact switch connected to the ESP8266 and is powered from a 5V wall adapter. When the door/window is opened it senses the change by the switch activating and detecting this change the ESP8266 will send an message to the MQTT broker. The message in the broker is then used to trigger the alert from the authentication server.

3.3 Authentication Server

The authentication server is arguably the most important part of this whole project. The server is responsible for the following aspects:

1. Authorization
2. Face Recognition
3. Managing The Camera
4. Interfacing with The Keypad

5. Registration of Face Data
6. Logging of Entry and Exit
7. Running The Database Server
8. Sending Alert to The User of Intrusion Detection
9. Running MQTT broker

3.3.1 The Functional Blocks of Authentication Server

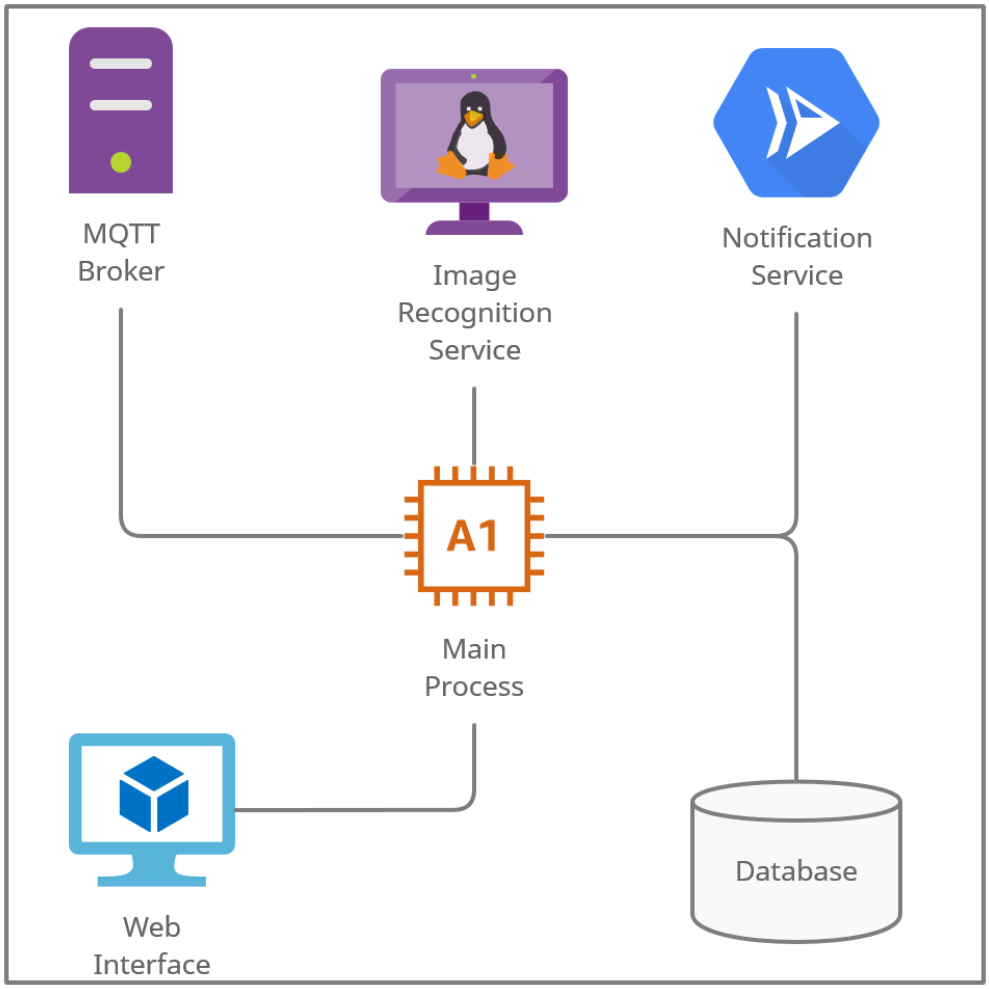

Figure 3.3: Functional Part of Authentication Server

Main Process

The main process ties everything together. It is responsible for managing and interfacing with all the other parts of the server.

Image Recognition Service

This used **OpenCV** with **dlib** to recognize faces and check their authorization.

Figure 3.4: Demonstrating Facial Recognition Capabilities

MQTT Broker

MQTT Broker is a message queue and broker. It supports very low latency message delivery and a variety of message channels. It is used to receive data from the "Intrusion Detection Sensor" and send to the Main Process.

Notification Service

Sends 'Push Notification' to the various client devices that are configured to receive.

Web Interface

The web interface is used for managing the whole system. It is used to enrol new authorized user to the authorization list also to see the log data.

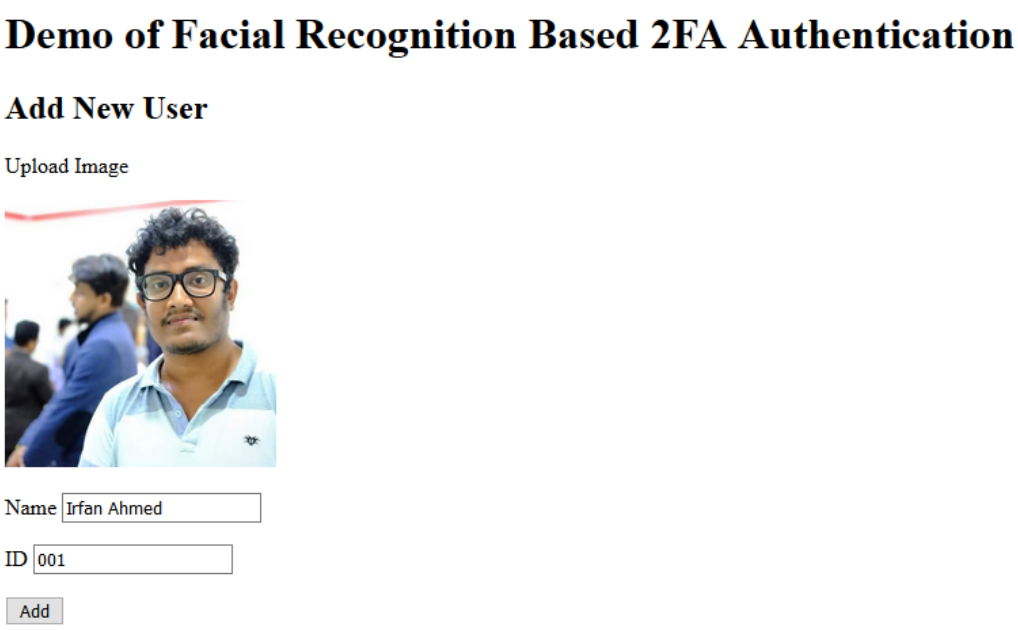

Figure 3.5: User Adding Interface

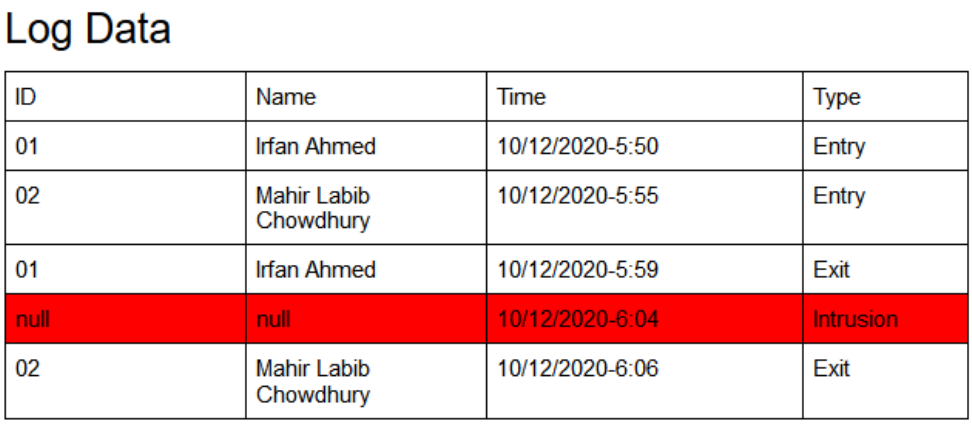

Figure 3.6: Entry/Exit Log

Database

The database is used to keep the time logs. It is running a redis server underneath. So the data is stored in a key-value pair for faster access times and lower latency.

Chapter 4

Conclusions

This is the final chapter of this report and it summarizes the key features, limitations of the project and suggests a future direction for further improvement.

4.1 Summary

This project implements 2FA on smart home/facilities. It uses a Facial Recognition System along with a traditional Keypad for pass code entry. It also uses wireless intrusion detection sensors to prevent intrusions going unnoticed. The information from this project can be accessed from anywhere of the world.

4.2 Limitations

This project uses a single camera with no depth sensor. There is a potential of fooling the system with a photograph of the authorized person. Even though this could not be reproducible in our testing but it is a limitation of the technology used.
The processor on the main unit Raspberry Pi is not very fast. Hence if more than 10 face is available in front of the camera it struggles to recognize the correct face.

4.3 Direction For Future Work

4.3.1 Camera

The problem of fooling the system with a photograph can be solved by using a stereo camera or by adding a depth camera with the system. The face encoding data then can not be used with normal photographs, it has to be collected with a similar depth/stereo camera.

4.3.2 Hardware

The Raspberry Pi 3B used in this project is a very old device. The newer Raspberry Pi 4B is a more capable device. This can solve this issue. Also, a full server grade hardware can easily solve this too while adding headroom for future expansion.

Appendix A

Main Process

This code handles everything including the Facial Recognition Aspects.

```
import functools
import face_recognition
import cv2
import numpy as np

from flask import (
    Blueprint, flash, g, redirect, render_template,
    request, session, url_for
    )
from werkzeug.security import check_password_hash,
   generate_password_hash

from bio_metric_atm.db import get_db
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
import serial
import hashlib
from pyfingerprint.pyfingerprint import PyFingerprint

reader = SimpleMFRC522()

bp = Blueprint('auth', __name__, url_prefix='/auth')


@bp.route('/register', methods=('GET', 'POST'))
def admin_register():
    if request.method == 'POST':
        user_id = request.form['user_id']
        name = request.form['name']
        password = request.form['password']
        balance = request.form['balance']
        face_encoding = request.form['image']
```

```
finger_id = request.form['finger_id']
finger_id_emergency = request.form['finger_id_emergency']
db = get_db()
error = None

if not user_id:
    error = 'User ID is required.'

elif not name:
    error = 'Full Name is requiered.'

elif not password:
    error = 'Password is reqired.'

elif not balance:
    error = 'Initial Balance is required.'

elif face_encoding and finger_id:
    error = 'fi'

elif face_encoding:
    error = 'f'

elif finger_id:
    error = 'i'

if error is 'fi':
    db.execute(
        'INSERT INTO user (user_id , name, password , balance ,
           face_encoding , finger_id , finger_id_emergency)
           VALUES (?, ?, ?, ?, ?, ?, ?)',
        (user_id , name, generate_password_hash(password),
            balance , face_encoding , finger_id ,
                finger_id_emergency)
        )
    db.commit()
    return redirect(url_for('auth.success'))

elif error is 'i':
    db.execute(
        'INSERT INTO user (user_id , name, password , balance ,
           finger_id , finger_id_emergency) VALUES (?, ?, ?, ?,
           ?, ?)',
        (user_id , name, generate_password_hash(password),
            balance , finger_id , finger_id_emergency)
```

```
                )
            db.commit()
            return redirect(url_for('auth.success'))

        elif error is 'f':
            db.execute(
                'INSERT INTO user (user_id, name, password, balance,
                    face_encoding) VALUES (?, ?, ?, ?, ?)',
                (user_id, name, generate_password_hash(password),
                    balance, face_encoding)
                )
            db.commit()
            return redirect(url_for('auth.success'))

        elif error is None:
            db.execute(
                'INSERT INTO user (user_id, name, password, balance)
                   VALUES (?, ?, ?, ?)',
                (user_id, name, generate_password_hash(password),
                    balance)
                )
            db.commit()
            return redirect(url_for('auth.success'))

        flash(error)
    return render_template('register.html')


@bp.route('/success', methods=('GET', 'POST'))
def success():
    return render_template('success.html')


@bp.route('/login', methods=('GET', 'POST'))
def login():
    session.clear()
    return render_template('wait.html')


@bp.route('/password', methods=('GET', 'POST'))
def password():
    error = None
    db = get_db()

    try:
```

```
        user_id, text = reader.read()

    finally:
        GPIO.cleanup()
    db.execute(
        'INSERT INTO internal (user_id, facematch, fingermatch) VALUES
            (NULL, 0, 0)'
        )
    db.commit()
    user = db.execute(
        'SELECT * FROM user WHERE user_id = ?', (user_id,)
        ).fetchone()

    if request.method == 'POST':
        password = request.form['password']

        if not check_password_hash(user['password'], password):
            error = 'Incorrect password'
        else:
            session['user_id'] = user['user_id']
            if user['finger_id']:
                return redirect(url_for('auth.finger_view'))
            elif user['face_encoding']:
                return redirect(url_for('auth.face_view'))
            else:
                return redirect(url_for('auth.balance_view'))

    if user is None:
        error = 'Please put your card and press "Continue"'
        flash(error)
        return redirect(url_for('auth.login'))
    else:
        if error:
            flash(error)
        return render_template('password.html', user=user)


@bp.route('/finger', methods=('GET', 'POST'))
def finger_view():
    """
    This will be the view for the finger print id
    """
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('auth.login'))
```

```
    return render_template('finger.html')

@bp.route('/finger_verification', methods=('GET', 'POST'))
def finger_verification():
    user_id = session.get('user_id')
    error = None
    db = get_db()
    try:

        f = PyFingerprint('/dev/ttyUSB1', 57600, 0xFFFFFFFF, 0x00000000
            )

        if(f.verifyPassword() == False):
            raise ValueError('The given password is wrong!')

    except Exception as e:
        print('Not initialized')
        print('Exception:' + str(e))

    try:
        while (f.readImage() == False):
            pass
        f.convertImage(0x01)
        result = f.searchTemplate()
        positionNumber = result[0]
        accuracyScore = result[1]
        if (positionNumber == -1):
            error = 'Finger Print not matched. Try again.'
            flash(error)
            return redirect(url_for('auth.finger_view'))
        else:
            user = db.execute(
                'SELECT * FROM user WHERE user_id =?',
                (user_id,)).fetchone()
            print(user['finger_id'])
            if user['finger_id'] == positionNumber:
                if user['face_encoding']:
                    return redirect(url_for('auth.face_view'))
                else:
                    return redirect(url_for('auth.balance_view'))
            elif user['finger_id_emergency'] == positionNumber:
                error = "Emergency Protocol Activated"
                flash(error)
                return redirect(url_for('auth.finger_view'))
```

```
    except Exception as e:
        error = str(e)
        flash(error)
        return redirect(url_for('auth.finger_view'))


@bp.route('/face', methods=('GET', 'POST'))
def face_view():
    """
    This will be the view for the facial recognation
    """
    db = get_db()
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('auth.login'))
    user = db.execute(
        'SELECT * FROM user WHERE user_id = ?', (user_id,)
        ).fetchone()

    return render_template('face.html')


@bp.route('/face_verification', methods=('GET', 'POST'))
def face_verification():
    user_id = session.get('user_id')
    db = get_db()
    if not user_id:
        return redirect(url_for('auth.login'))
    user = db.execute(
        'SELECT * FROM user WHERE user_id =?', (user_id,)
        ).fetchone()
    video_capture = cv2.VideoCapture(0)

    filename ="/home/pi/"+ user['face_encoding']
    file = face_recognition.load_image_file(filename)
    file_encoding = face_recognition.face_encodings(file)[0]
    known_face_encodings = [
        file_encoding,
        ]
    known_face_names = [
        user['name'],
        ]
    face_locations = []
    face_encodings = []
    face_names = []
```

```
    process_this_frame = True
    while True:
        ret, frame = video_capture.read()
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

        rgb_small_frame = small_frame[:,:,::-1]
        if process_this_frame:
            face_locations = face_recognition.face_locations(
                rgb_small_frame)
            face_encodings = face_recognition.face_encodings(
                rgb_small_frame, face_locations)
            face_names = []
            for face_encoding in face_encodings:
                matches = face_recognition.compare_faces(
                    known_face_encodings, face_encoding)
                name = "Unknown"
                face_distances = face_recognition.face_distance(
                    known_face_encodings, face_encoding)
                best_match_index = np.argmin(face_distances)
                if matches[best_match_index]:
                    name = known_face_names[best_match_index]
                face_names.append(name)
        process_this_frame = not process_this_frame


        if user['name'] in face_names:
            break
    video_capture.release()
    cv2.destroyAllWindows()
    return redirect(url_for('auth.balance_view'))
```

```
@bp.route('/balance', methods=('GET', 'POST'))
def balance_view():
    """
    This will be the view for the balance information
    """
    db = get_db()
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('auth.login'))
```

```
    user = db.execute(
        'SELECT * FROM user WHERE user_id = ?', (user_id,)
        ).fetchone()
    return render_template('balance.html', user=user)


@bp.route('/withdraw', methods=('GET', 'POST'))
def withdraw():
    db = get_db()
    user_id = session.get('user_id')
    if not user_id:
        return redirect(url_for('auth.login'))
    user = db.execute(
        'SELECT * FROM user WHERE user_id = ?', (user_id,)
        ).fetchone()
    if request.method == 'POST':
        ammount = request.form['withdraw']
        balance = user['balance']
        if int(ammount) > balance:
            error = 'You do not have enough money'
            flash(error)
            return redirect(url_for('auth.balance_view'))
        else:
            """
            The code logic that will send the number of notes to the
            Arduino based cash dispensing system through serial.
            Also the balance of the user will be updated here.
            """
            note = int(ammount) /100
            ser = serial.Serial('/dev/ttyUSB0')
            print(ser)
            note = int(note)
            print(type(note))
            ser.write(b'%d' % note)
            print(note)
            ser.close()
            new_balance = balance - int(ammount)
            db.execute(
                'UPDATE user SET balance=? WHERE user_id=?',
                (new_balance, user_id))
            db.commit()

            return redirect(url_for('auth.balance_view'))

    return render_template('withdraw.html', user=user)
```

Database

This code interfaces the database.

```
import sqlite3

import click
from flask import current_app, g
from flask.cli import with_appcontext


def get_db():
    if 'db' not in g:
        g.db = sqlite3.connect(
            current_app.config['DATABASE'],
            detect_types=sqlite3.PARSE_DECLTYPES
            )
        g.db.row_factory = sqlite3.Row

    return g.db


def close_db(e=None):
    db = g.pop('db', None)

    if db is not None:
        db.close()


def init_db():
    db = get_db()
    with current_app.open_resource('schema.sql') as f:
        db.executescript(f.read().decode('utf8'))


@click.command('init_db')
@with_appcontext
def init_db_command():
    init_db()
    click.echo("Initialized the Database.")


def init_app(app):
    app.teardown_appcontext(close_db)
    app.cli.add_command(init_db_command)
```

App Initialization

This initializes the Application.

```
import os

from flask import Flask
from . import db
from . import auth


def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        DATABASE=os.path.join(app.instance_path, 'bio_metric_atm.sqlite
            ')
        )
    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)

    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)

    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    # a simple page that says hello world
    @app.route('/hello')
    def hello():
        return "Hello, World"

    db.init_app(app)
    app.register_blueprint(auth.bp)
    return app
```

ESP8266 Intrusion Detection

This is the part that sends MQTT messages.

```
// Base ESP8266
#include <ESP8266WiFi.h>
WiFiClient WIFI_CLIENT;

// MQTT
#include <PubSubClient.h>
PubSubClient MQTT_CLIENT;

#define AP_NAME "AP_NAME"
#define AP_PASS "AP_PASS"
#define LIGHT_SENSOR A0
#define LED 15
#define BUTTON 4

// This function runs once on startup
void setup() {
  // Initialize the serial port
  Serial.begin(115200);

  // Configure light sensor pin as an input
  pinMode(LIGHT_SENSOR, INPUT);

  // Configure LED pin as an output
  pinMode(LED, OUTPUT);

  // Configure BUTTON pin as an input with a pullup
  pinMode(BUTTON, INPUT_PULLUP);

  // Attempt to connect to a specific access point
  WiFi.begin(AP_NAME, AP_PASS);

  // Keep checking the connection status until it is connected
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
  }

  // Print the IP address of your module
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

// This function connects to the MQTT broker
void reconnect() {
  // Set our MQTT broker address and port
  MQTT_CLIENT.setServer("iot.eclipse.org", 1883);
```

```
  MQTT_CLIENT.setClient(WIFI_CLIENT);

  // Loop until we're reconnected
  while (!MQTT_CLIENT.connected()) {
    // Attempt to connect
    Serial.println("Attempt to connect to MQTT broker");
    MQTT_CLIENT.connect("<your_random_device_client_id>");

    // Wait some time to space out connection requests
    delay(3000);
  }

  Serial.println("MQTT connected");
}

// This function runs over and over again in a continuous loop
void loop() {

  // Check if we're connected to the MQTT broker
  if (!MQTT_CLIENT.connected()) {
    // If we're not, attempt to reconnect
    reconnect();
  }

  // Publish a message to a topic
  MQTT_CLIENT.publish("10.0.1.4", "breakin");

  // Wait five seconds
  delay(5000);
}
```