**SCHOOL OF DIGITAL MEDIA AND INFOCOMM TECHNOLOGY (DMIT)**

|  |  |
| --- | --- |
| **Date of Submission:** | 19 August 2018 |

|  |  |
| --- | --- |
| **Prepared for:** | Ms Zhao Lin |

|  |  |
| --- | --- |
| **Class:** | DISM/FT/3A/03 |

|  |  |
| --- | --- |
| **Submitted by:** |  |

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 1625509 | Gui Zhang Yan, Dexter |
| 1625439 | Lee Jun Yi, Joshua |

**IOT CA2**

**Step-by-step Tutorial**

**DIPLOMA IN BUSINESS INFORMATION TECHNOLOGY**

**DIPLOMA IN INFORMATION TECHNOLOGY**

**DIPLOMA IN INFOCOMM SECURITY MANAGEMENT**

**ST0324 Internet of Things (IOT)**

**2017/2018 Semester 1**

**Table of Contents**

[Section 1 Overview of project 3](#_Toc522489465)

[A. Where we have uploaded our tutorial 3](#_Toc522489466)

[B. Why have we chosen to upload to this site 3](#_Toc522489467)

[C. What have we uploaded 3](#_Toc522489468)

[D. What is the application about? 3](#_Toc522489469)

[E. Summary of the steps that will be described 4](#_Toc522489470)

[F. How does the final RPI set-up looks like? 5](#_Toc522489471)

[G. How does the web or mobile application look like? 6](#_Toc522489472)

[Section 2 Hardware requirements 10](#_Toc522489473)

[Hardware checklist 10](#_Toc522489474)

[Button 10](#_Toc522489475)

[Light-Dependant Resistor (LDR) 10](#_Toc522489476)

[Resistor for LED, Button and Light-Dependant Resistor (LDR) 11](#_Toc522489477)

[Resistor for LED 12](#_Toc522489478)

[Buzzer 12](#_Toc522489479)

[Buzzer 13](#_Toc522489480)

[I2C LCD Screen 13](#_Toc522489481)

[Section 3 Hardware Set-up 14](#_Toc522489482)

[Set-up the Hardware Required 14](#_Toc522489483)

[a) Gate Security RPI 14](#_Toc522489484)

[Completed Fritzing Diagram 14](#_Toc522489485)

[Connect the Buzzer 14](#_Toc522489486)

[Connect the LEDs 15](#_Toc522489487)

[Connect the Buttons 15](#_Toc522489488)

[Connect the LCD 15](#_Toc522489489)

[b) Shoe Cabinet RPI 16](#_Toc522489490)

[Completed Fritzing Diagram 16](#_Toc522489491)

[Connect the LEDs 16](#_Toc522489492)

[Connect the Buzzer 17](#_Toc522489493)

[Connect the Buttons 17](#_Toc522489494)

[Section 4 Appplication Prerequisites 18](#_Toc522489495)

[All Packages and Directories Required 18](#_Toc522489496)

[Section 5 AWS IoTcore (MQTT) 19](#_Toc522489497)

[Set-up Amazon Web Service IoTcore service 19](#_Toc522489498)

[Section 6 AWS DynamoDB 21](#_Toc522489499)

[Set-up Amazon Web Service NoSQL DynamoDB service 21](#_Toc522489500)

[Section 7 AWS S3 25](#_Toc522489501)

[Set-up Amazon Web Service Image Repository S3 25](#_Toc522489502)

[Section 8 Telegram Bot 26](#_Toc522489503)

[Set-up Telegram Bot 26](#_Toc522489504)

[Section 9 OpenCV Facial Recognition 27](#_Toc522489505)

[Set-up OpenCV Facial Recognition 27](#_Toc522489506)

[Section 10 Coding 32](#_Toc522489507)

[Code the Programs 32](#_Toc522489508)

[Section 11 Testing 66](#_Toc522489509)

[Test the finished Programs 66](#_Toc522489510)

# Section 1 Overview of project

* 1. Where we have uploaded our tutorial

<https://github.com/joshualeejunyi/KremePi>

* 1. Why have we chosen to upload to this site

The reason why we have chosen github is not just because of how reputable it is. In fact, one of the main reasons why we have picked github is because of how easy it is for the public to contribute to our project. Github offers a wiki as well as a issues tracker that would make it easier for us to include a more indepth documentation and gather feedback for our project. Not only so, github is not just a website for documentations and instructions, however, a repository, which means that it would allow us to keep updating ,make changes or improve our project easily. Additionally, github is widely supported by numerous services and operating systems, demonstrating great versatility.

* 1. What have we uploaded

We have uploaded all the source codes required for our project, as well as a clear and concise documentation on how our project functions as well as how any user can replicate our project if they intend to do so. A diagram on how a user should position their raspberry pis outside their house will also be provided if the user wishes to fully replicate our project.

* 1. What is the application about?

Nowadays, there is a trend that most of us would invest in high-end sneakers or shoes. Because of their high prices, we would be afraid to leave them unguarded, therefore bring them into the house. However, it is a chore to clean the shoes so that we do not dirty our house each time we bring it in. As such, our application intends to solve this. Our application will provide a form of security for your shoes, as well as implement a gate security that is convenient for both guests and home owners.

* 1. Summary of the steps that will be described

|  |  |  |
| --- | --- | --- |
|  | Section | Description |
|  | Overview | Project introduction |
|  | Hardware requirements | Hardware required for this project |
|  | Hardware Set-up | Set-up that hardware required for the project |
|  | Application Prerequisites | All packages or directories that are needed for this project. |
|  | AWS IoT Core (MQTT) | Set-up Amazon Web Service IoTcore |
|  | AWS DynamoDB | Set-up Amazon Web Service NoSQL DynamoDB |
|  | AWS S3 | Set-up Amazon Web Service Image Repository S3 |
|  | Telegram Bot | Set-up Telegram Bot |
|  | OpenCV Facial Recognition | Set-up OpenCV Facial Recognition |
|  | Code the Programs | Code the Application |
|  | Test the Programs | Test all the Applications |

* 1. How does the final RPI set-up looks like?

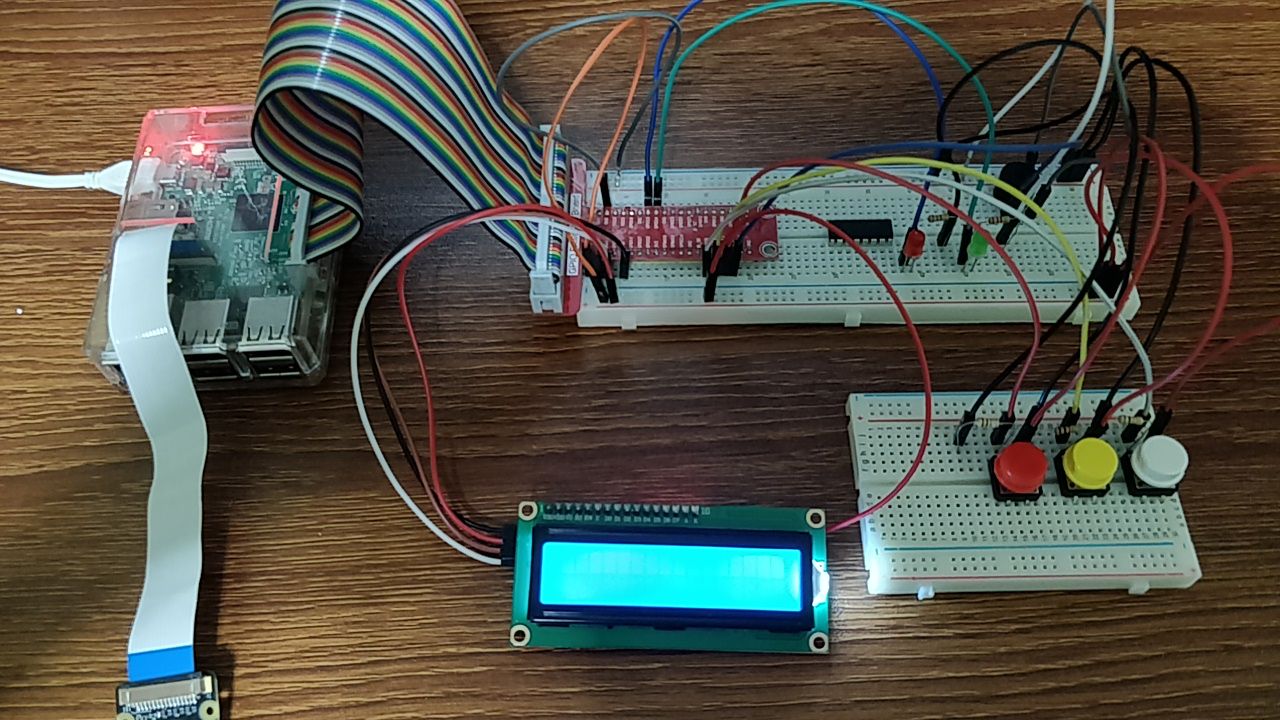


Figure 1: Final product of RPI(doorbell)

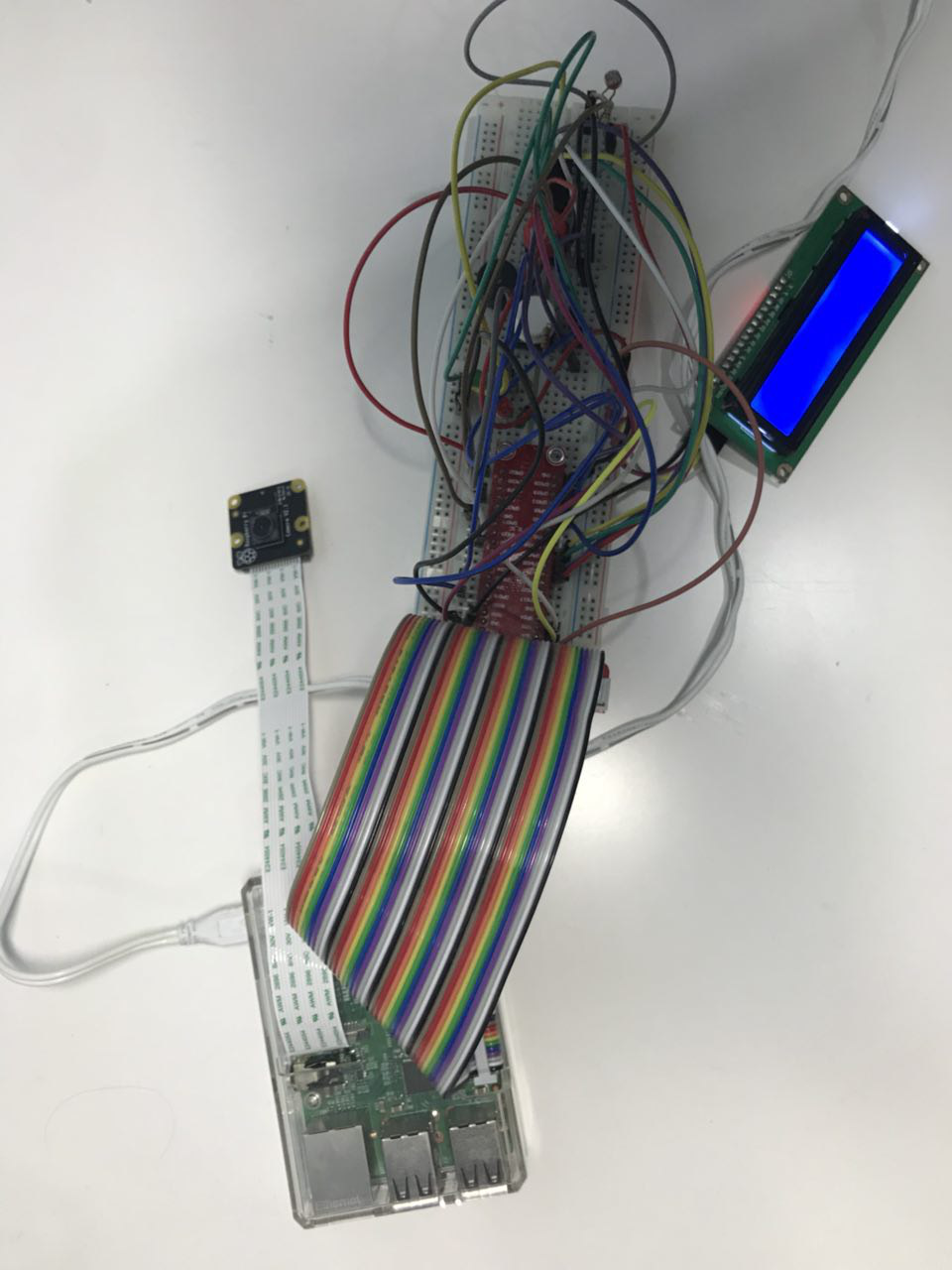
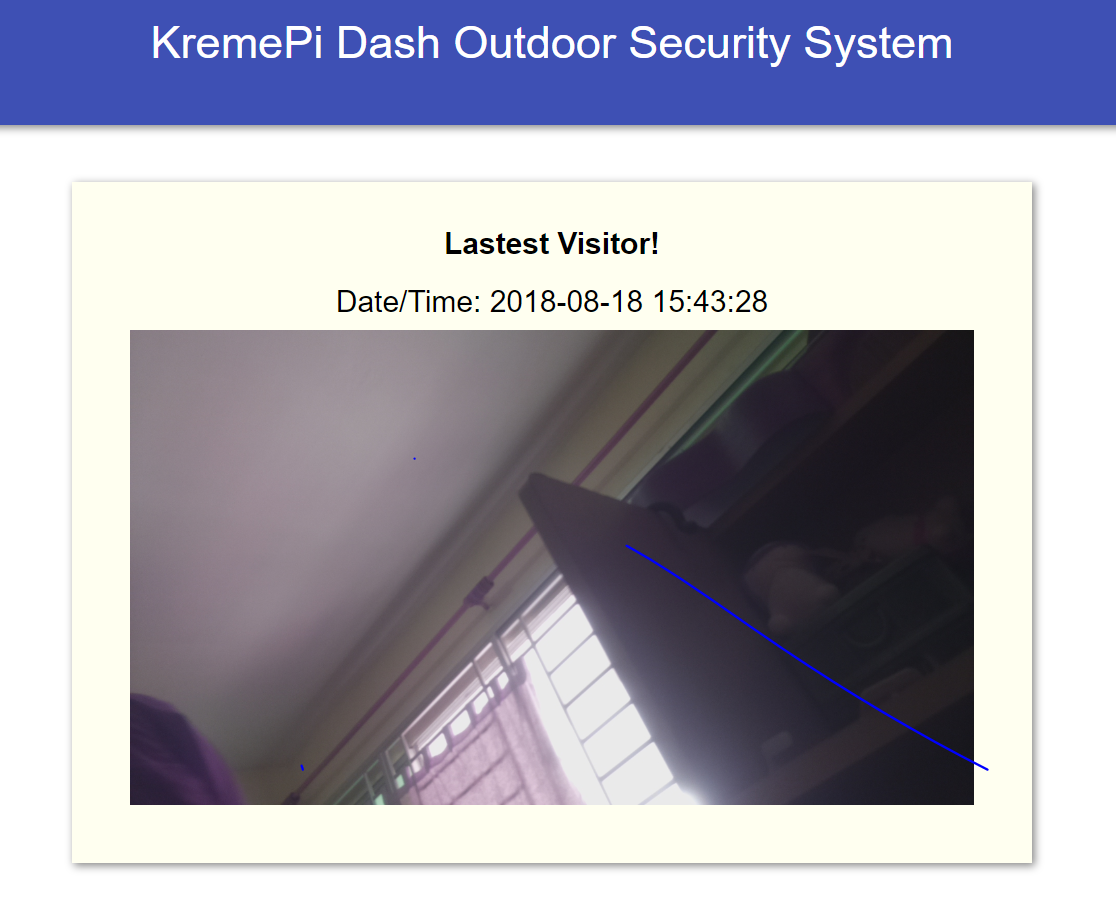
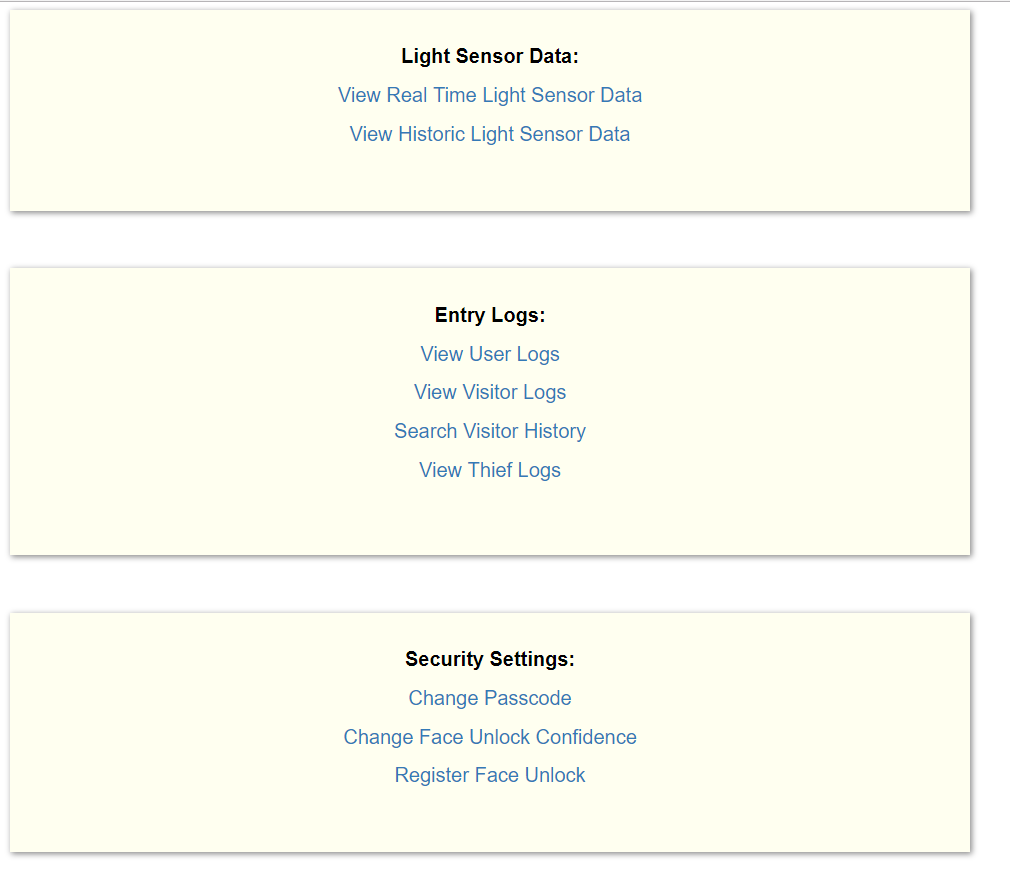
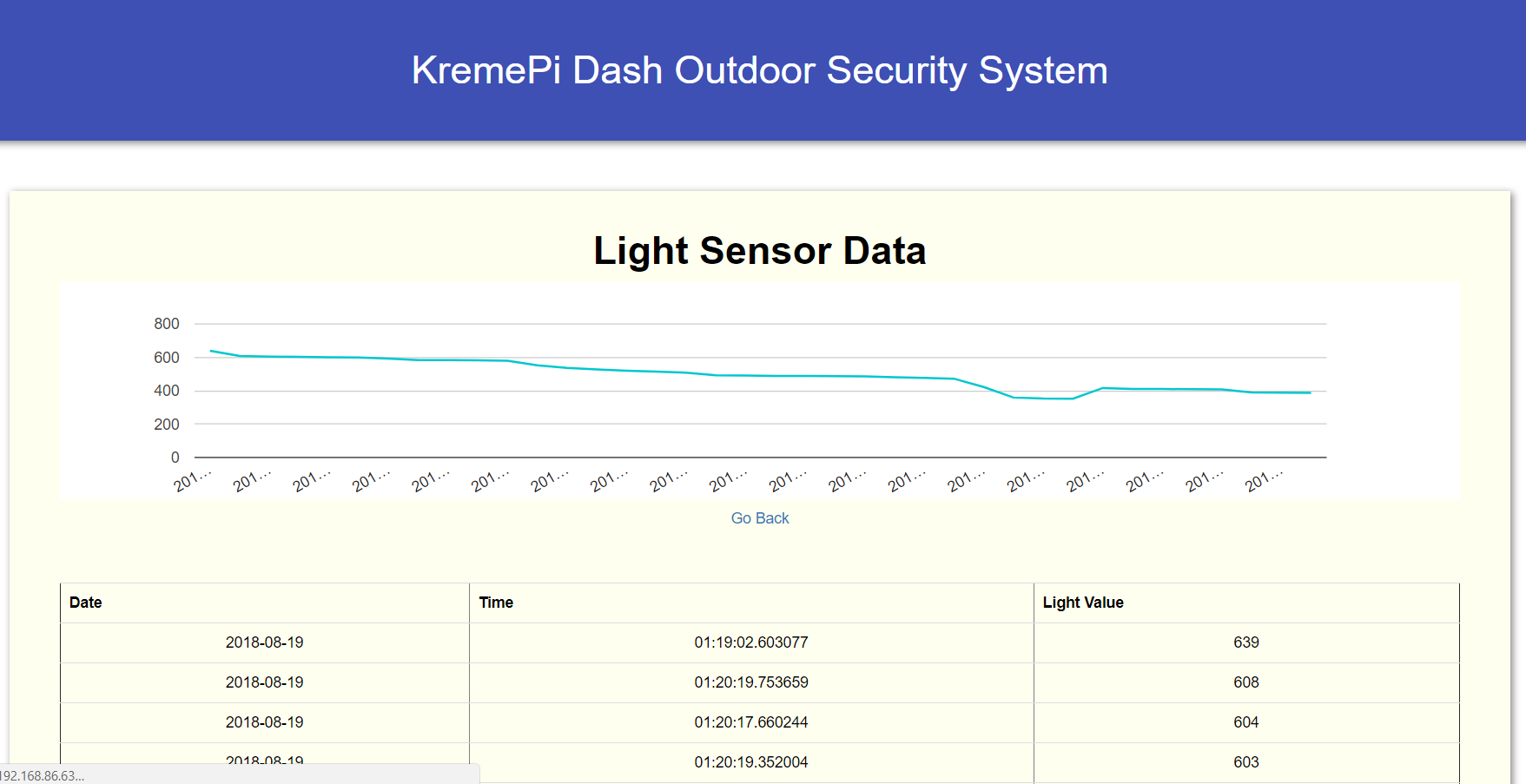


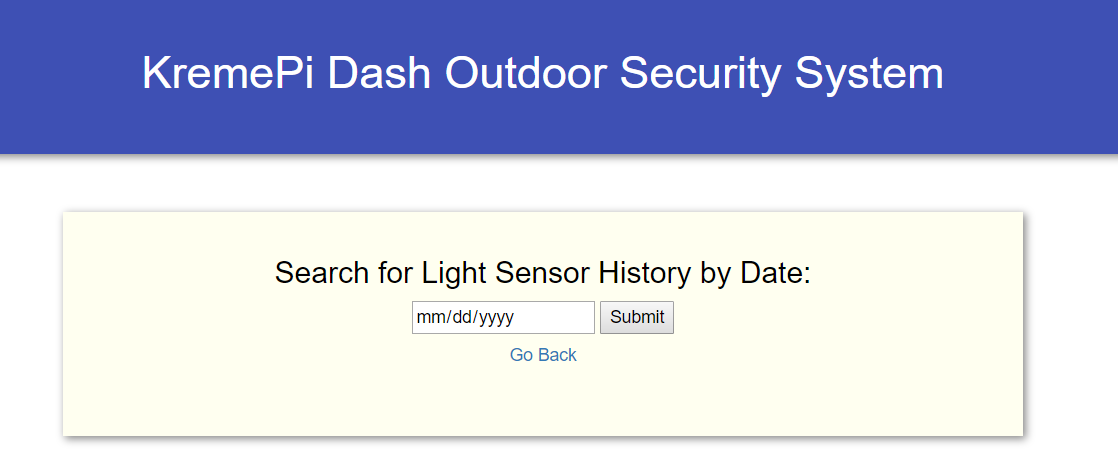
Figure 2: Final product of RPI(shoe cabinet)

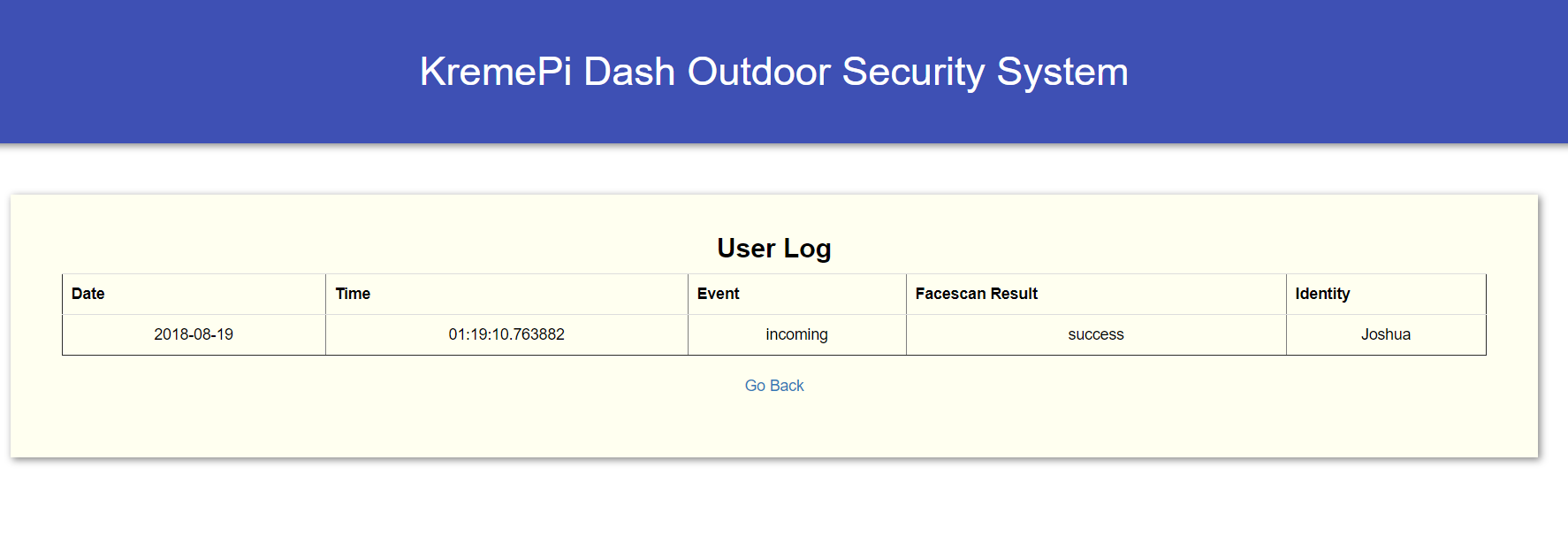
* 1. How does the web or mobile application look like?

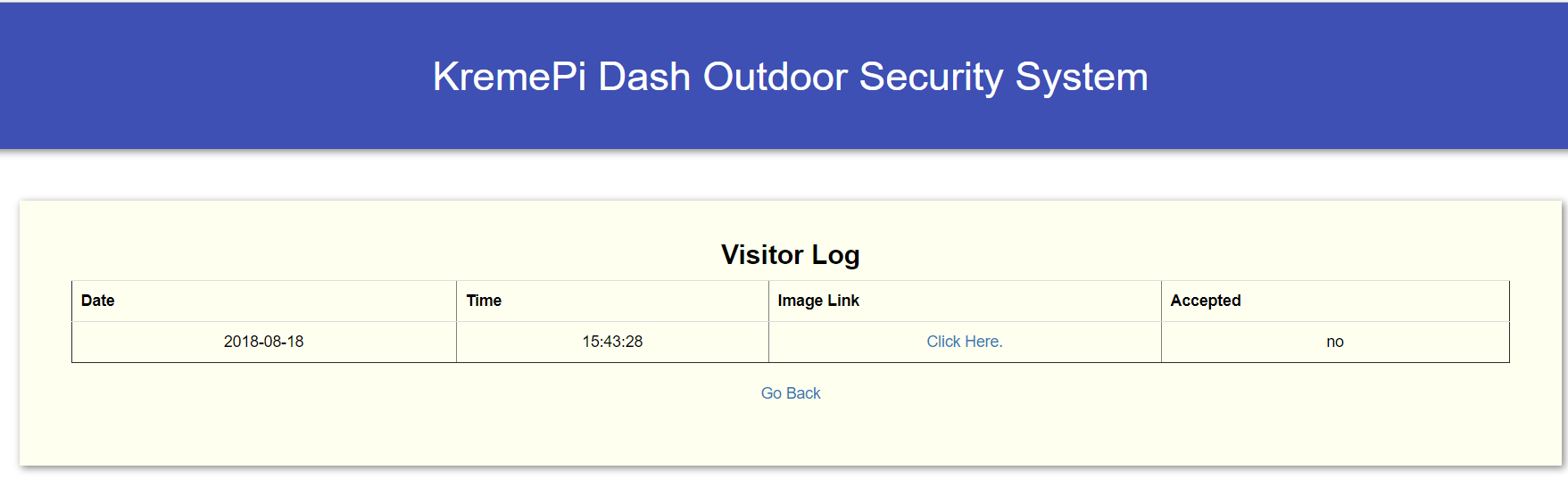


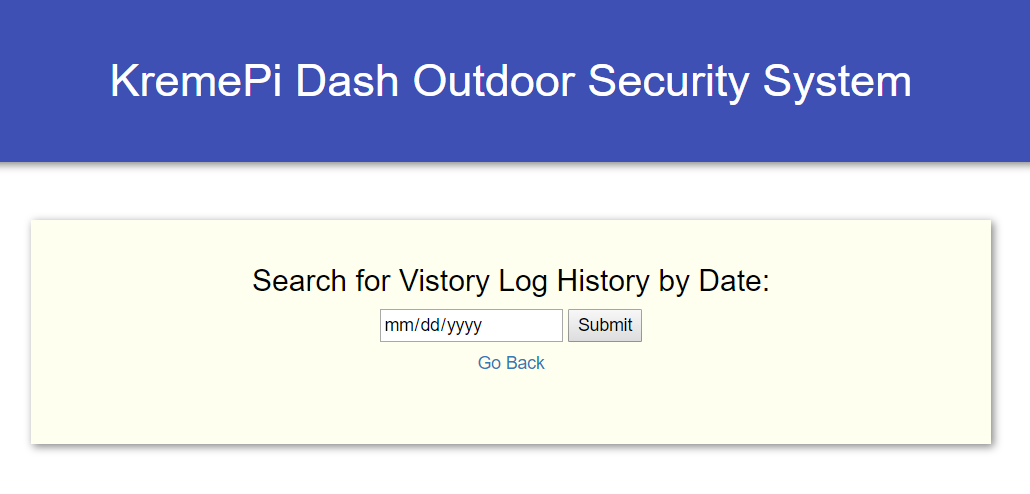


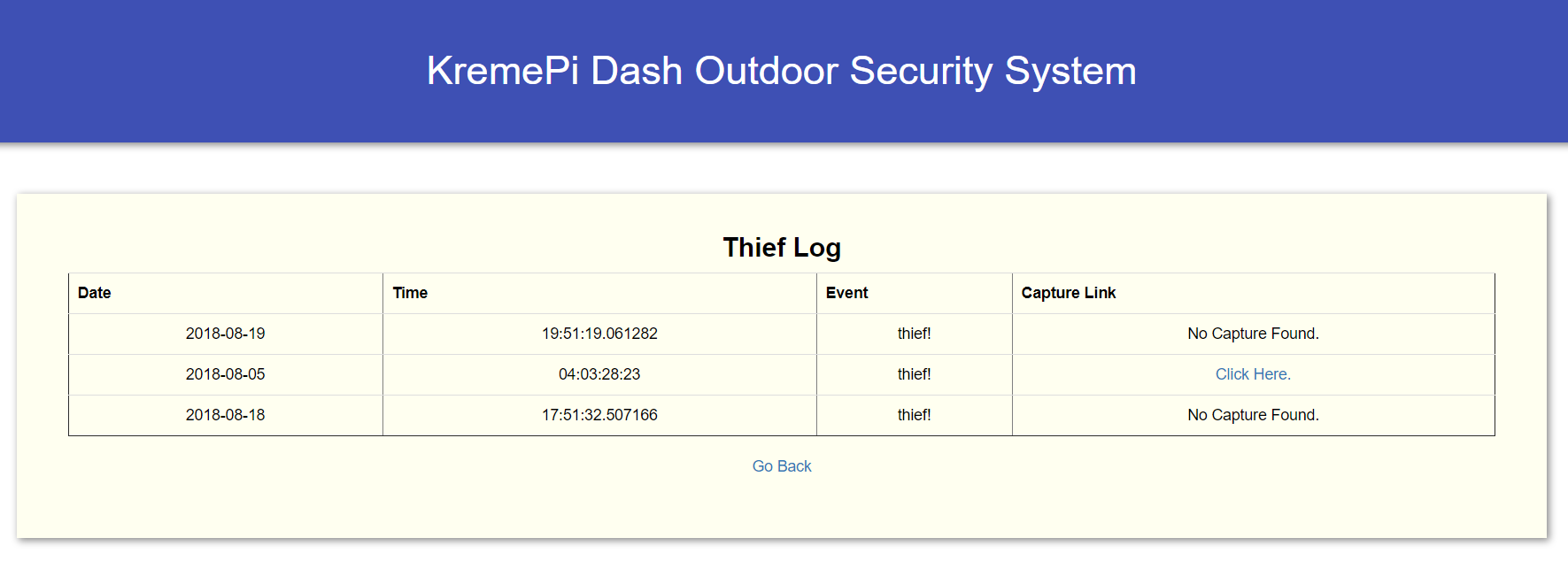


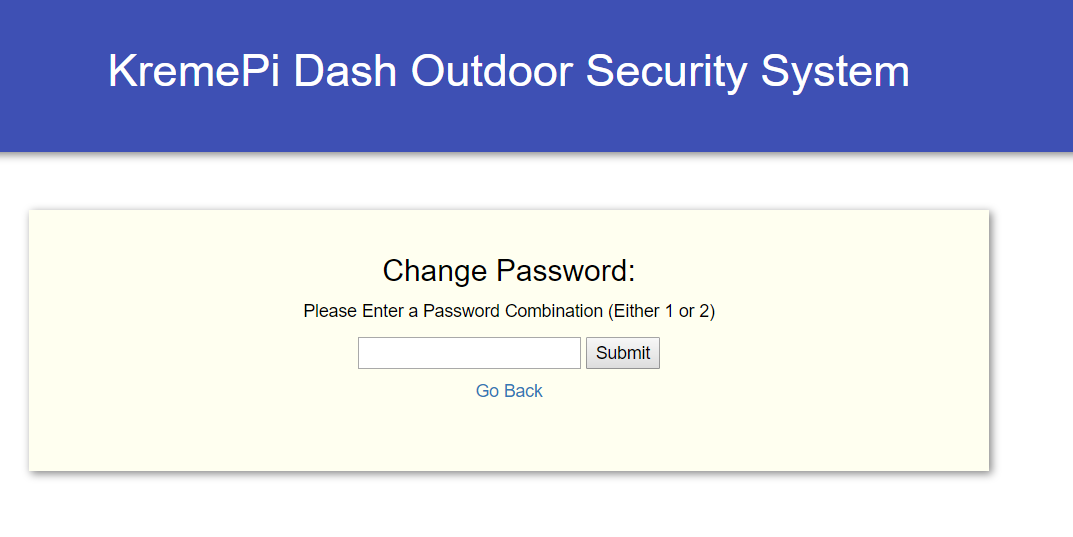


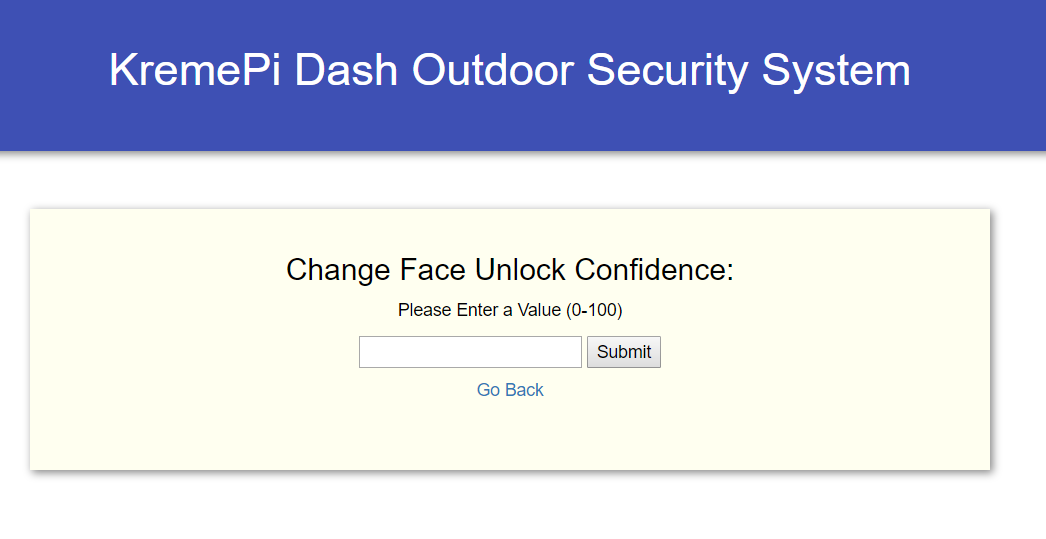


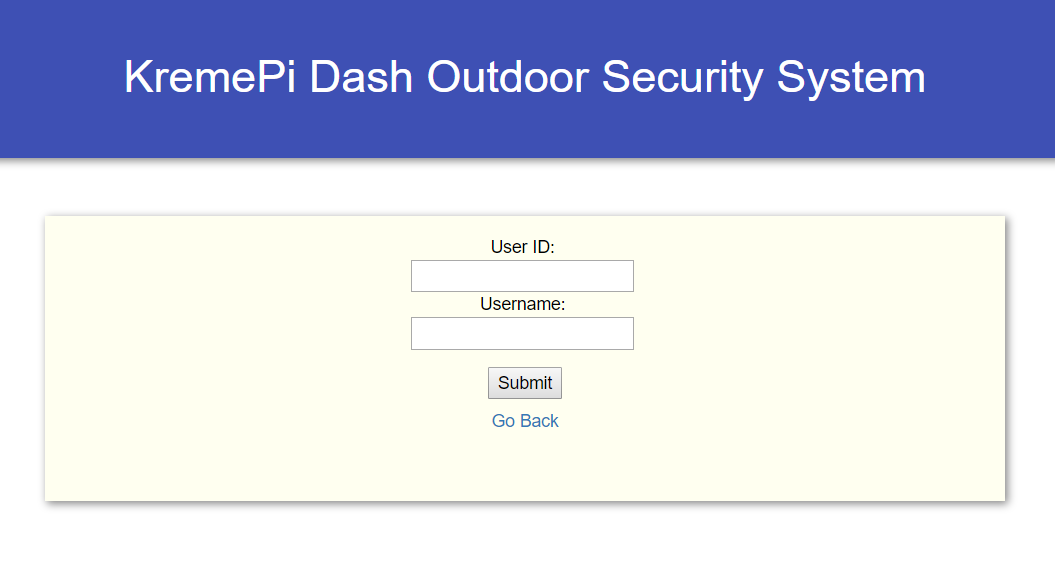












# Section 2 Hardware requirements

Hardware checklist

### Button

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * The picture here shows a tactile push-button * Note that it has 4 'legs'. These legs are the ones that send the signals. * Note that unlike a LED which is an actuator (output), push-buttons act as sensors (inputs) * To know whether a push-button is pressed or unpressed, we can detect its HIGH or LOW state, which are passed through the 'legs' | **BUTTON** |

### Light-Dependant Resistor (LDR)

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * Light-Dependant Resistor (LDR) are light sensitive resistors which change resistance based on how much light they are exposed to. * The more light a LDR receives, the less resistant it becomes, i.e. lets more current flow * When it’s in the dark, the resistance is very high * The resistance of an LDR may typically have the following resistances:   + Daylight = 5000Ω   + Dark = 20000000Ω | **LDR** |

### Resistor for LED, Button and Light-Dependant Resistor (LDR)

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * Resistors can range from 100Ω to 10000Ω * The value of a resistor is marked with coloured bands along the length of the resistor body. * For the LED of this project, you will be using a **330Ω** resistor. * You can identify the 330Ω resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied:   For example, on a four color band resistors, a 330Ω resistor is colored Orange, Orange, Brown, and then Gold. On a five color band 330Ω resistor, the colours are Orange, Orange, Black, Black, Brown. | <https://www.allaboutcircuits.com/tools/resistor-color-code-calculator/>  **330 ohms RESISTOR**    Figure 3: Orange,Orange,Brown |
|  | Adding a button or LDR to the circuit can introduce irregularities in the current flow to the RPi and damage our RPi  To ensure our RPi stays safe, we will be adding a 10K Ω **pull-down** resistor to moderate the current flow through the circuit. You can recognise the 10K ohms resistor by its color bands (brown-black-orange-gold)  Note that when the pull-down resistor is used in the circuit for the Button, it will change the current flow as follows:   * When the button is pressed: A small current flows through the input pin connected to the button, and the pin will read HIGH * When the button is unpressed: The current passes through the resistor and directly to GND pin. There is no current passing through the input pin and thus it will read LOW | **10K Ω RESISTOR** |

### Resistor for LED

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * Resistors can range from 100Ω to 10000Ω * The value of a resistor is marked with coloured bands along the length of the resistor body. * In this lab, you will be using a **330Ω** resistor. * You can identify the 330Ω resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied: * For example, on a four color band resistors, a 330Ω resistor is colored Orange, Orange, Brown, and then Gold. On a five color band 330Ω resistor, the colours are Orange, Orange, Black, Black, Brown. | <https://www.allaboutcircuits.com/tools/resistor-color-code-calculator/>  **330 ohms RESISTOR**    Figure 4: Orange,Orange,Brown |

### Buzzer

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | A buzzer is an audio signaling device which is commonly found in circuits to create a buzzing or beeping noise.  Buzzers can be categorized as active buzzers and passive ones. For our lab, we will use active buzzers as they are a lot simpler to use than passive ones though they are slightly more expensive.  An active buzzer can be connected just like an LED but they are even easier to use because a resistor is not needed.  A buzzer typically has 2 pins   * + VOUT – Connect this to a GPIO pin to control its value   + GND – Connect this to ground | **Buzzer** |

### Buzzer

|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * The Raspberry Pi camera is an official accessory that hooks up to a special connector on the Raspberry Pi. * It can be used on any model of the Raspberry Pi except the first version of Pi Zero * The camera is mounted on a small printed circuit board and connects through a ribbon cable. * The connector provides direct access between the camera and the processor. This is more efficient than using a webcam, which needs to connect through the USB protocol * The picture on the right shows the camera connected to a Raspberry Pi. | **piCam** |

### I2C LCD Screen

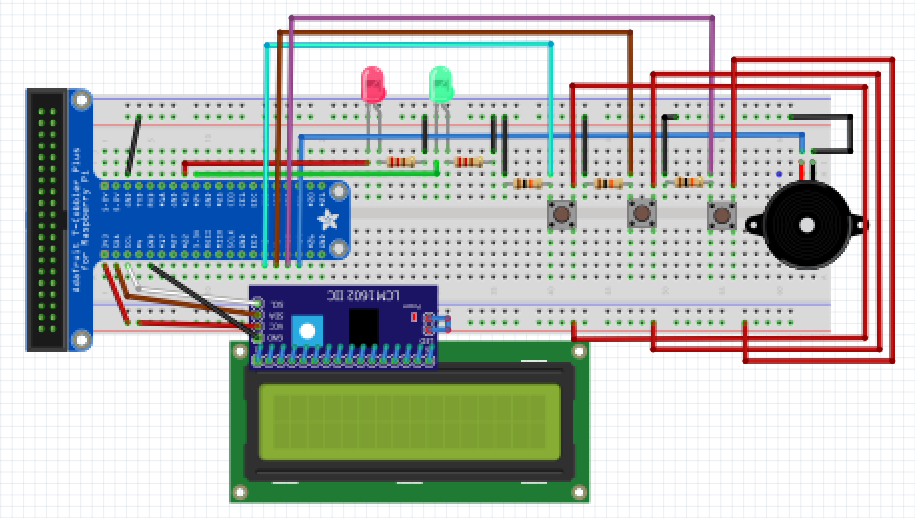
|  |  |  |
| --- | --- | --- |
|  | Task | |
|  | * LCDs are available in tons of colors and sizes. For example, you might have 8x1 LCDs or 20x4 LCDs * For this application, we will use the commonly available 16x2 LCD which can display up to 32 characters. * We will use the i2C version which require you to connect only 2 GPIO pins to your Raspberry Pi. * If you are buying your own LCDs, do make sure you buy the i2C version though they might cost a bit more. The non-i2C versions require you to connect double the number of GPIO pins! | **i2c LCD Screen** |

# Section 3 Hardware Set-up

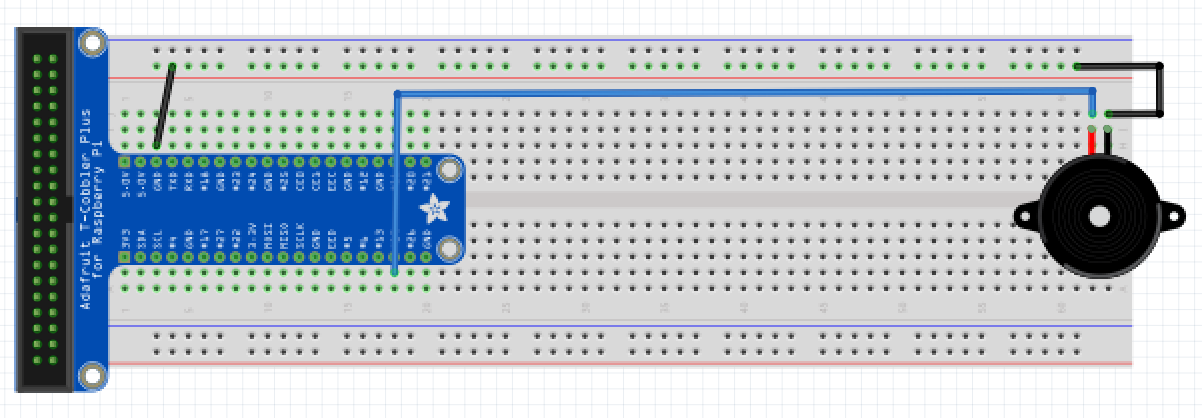
Set-up the Hardware Required

1. Gate Security RPI

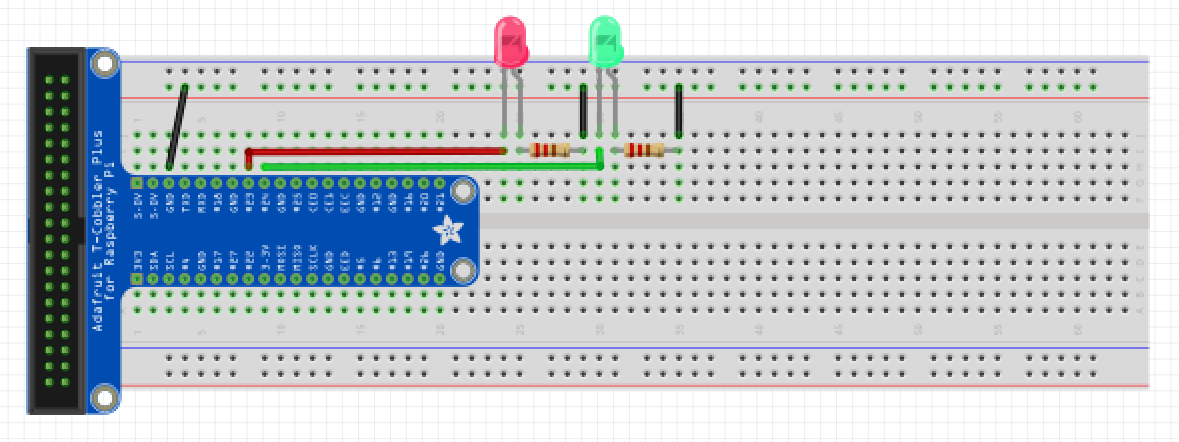
### Completed Fritzing Diagram



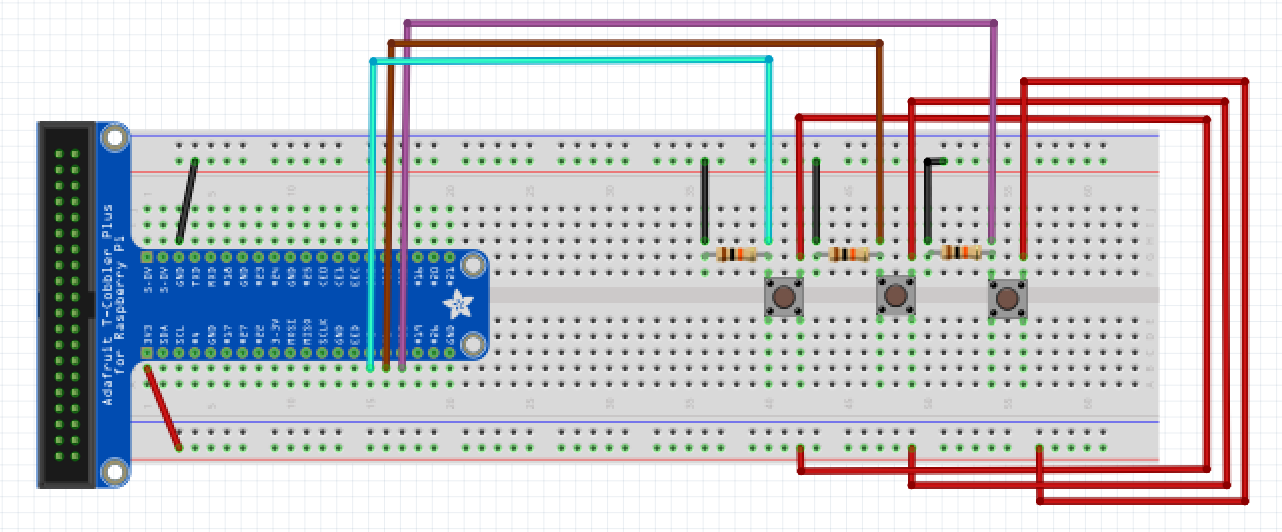
### Connect the Buzzer



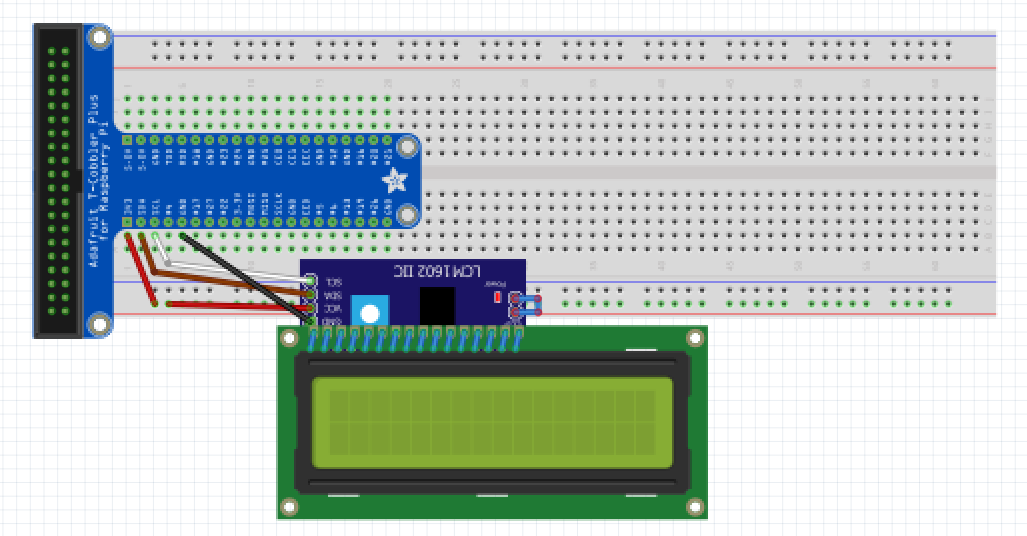
### Connect the LEDs



### Connect the Buttons

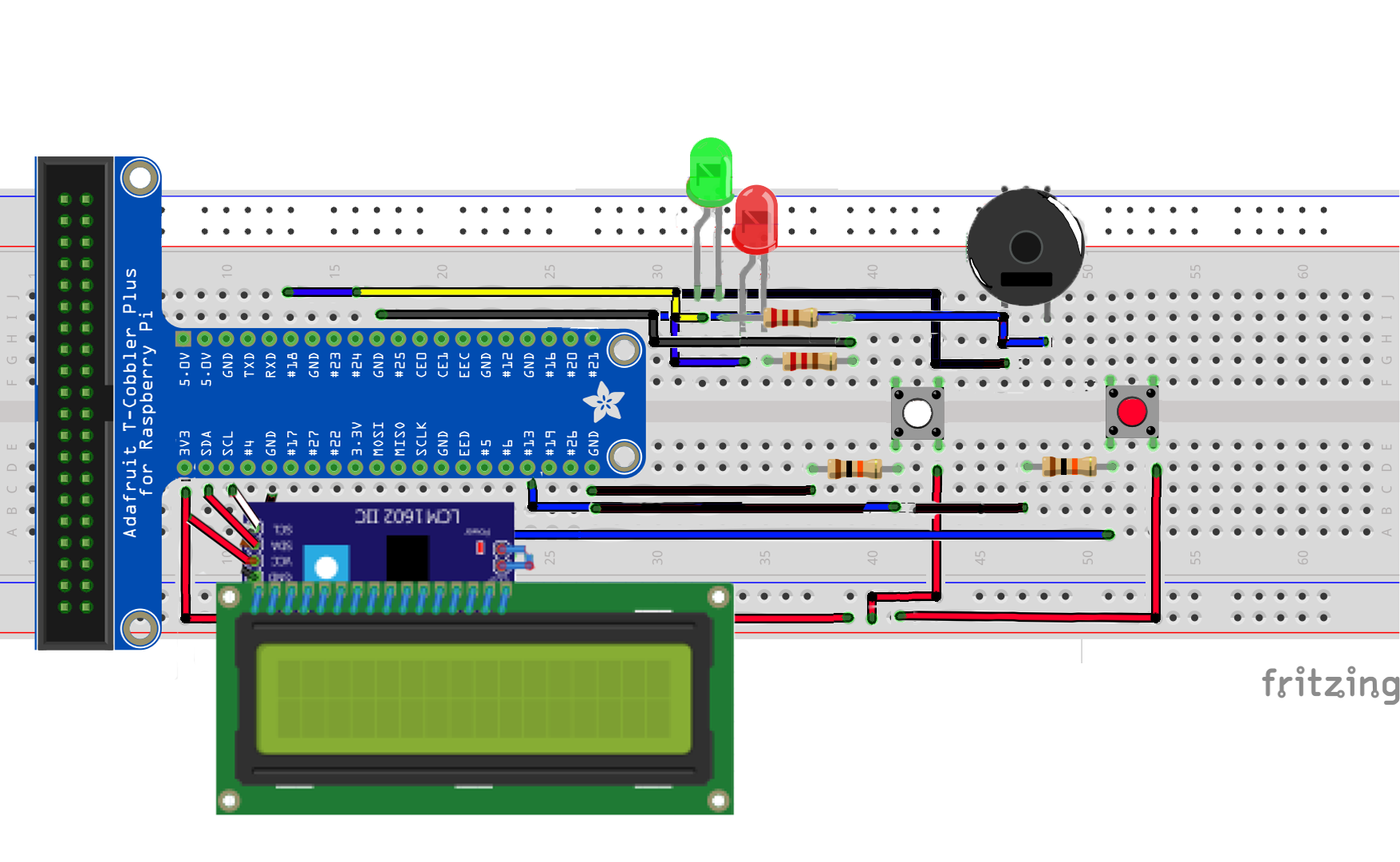


### Connect the LCD

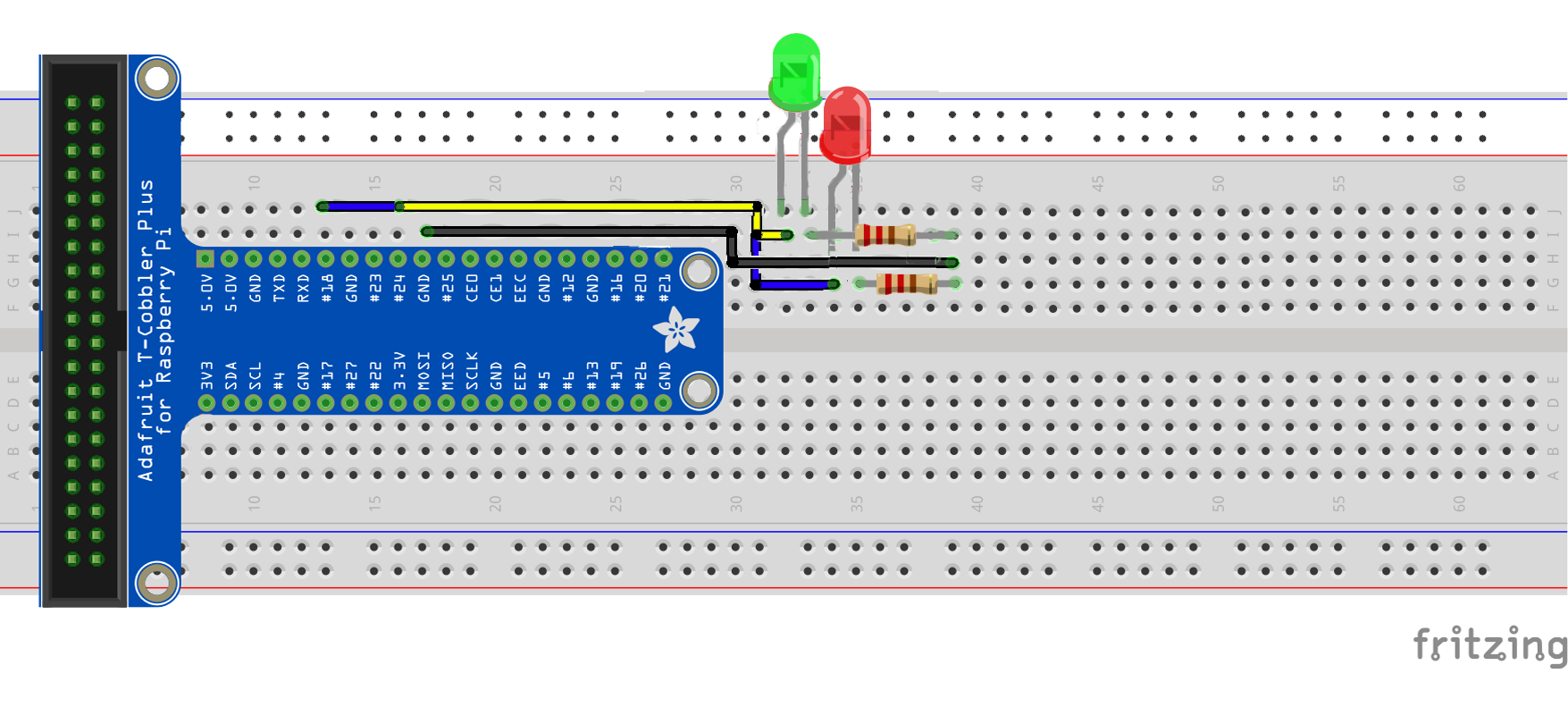


1. Shoe Cabinet RPI

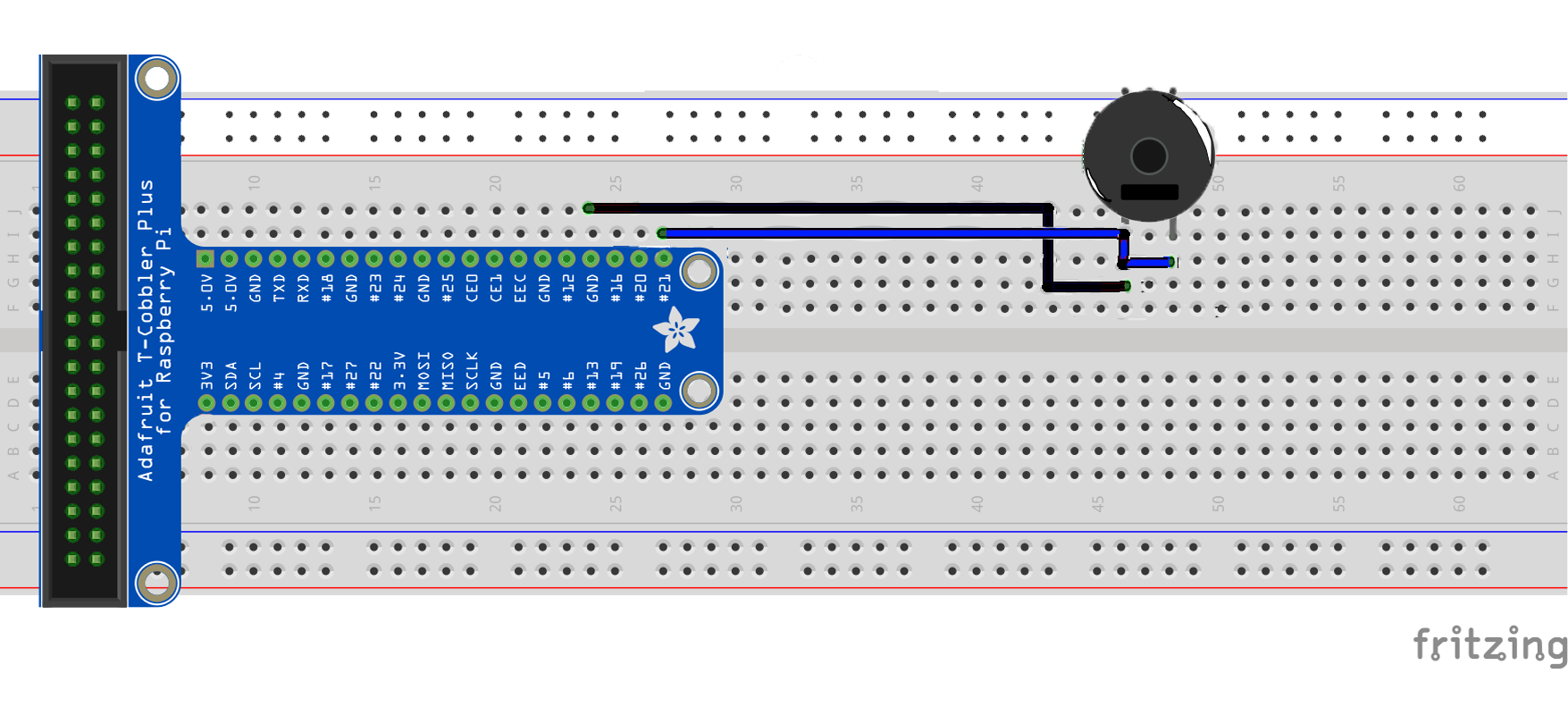
### Completed Fritzing Diagram



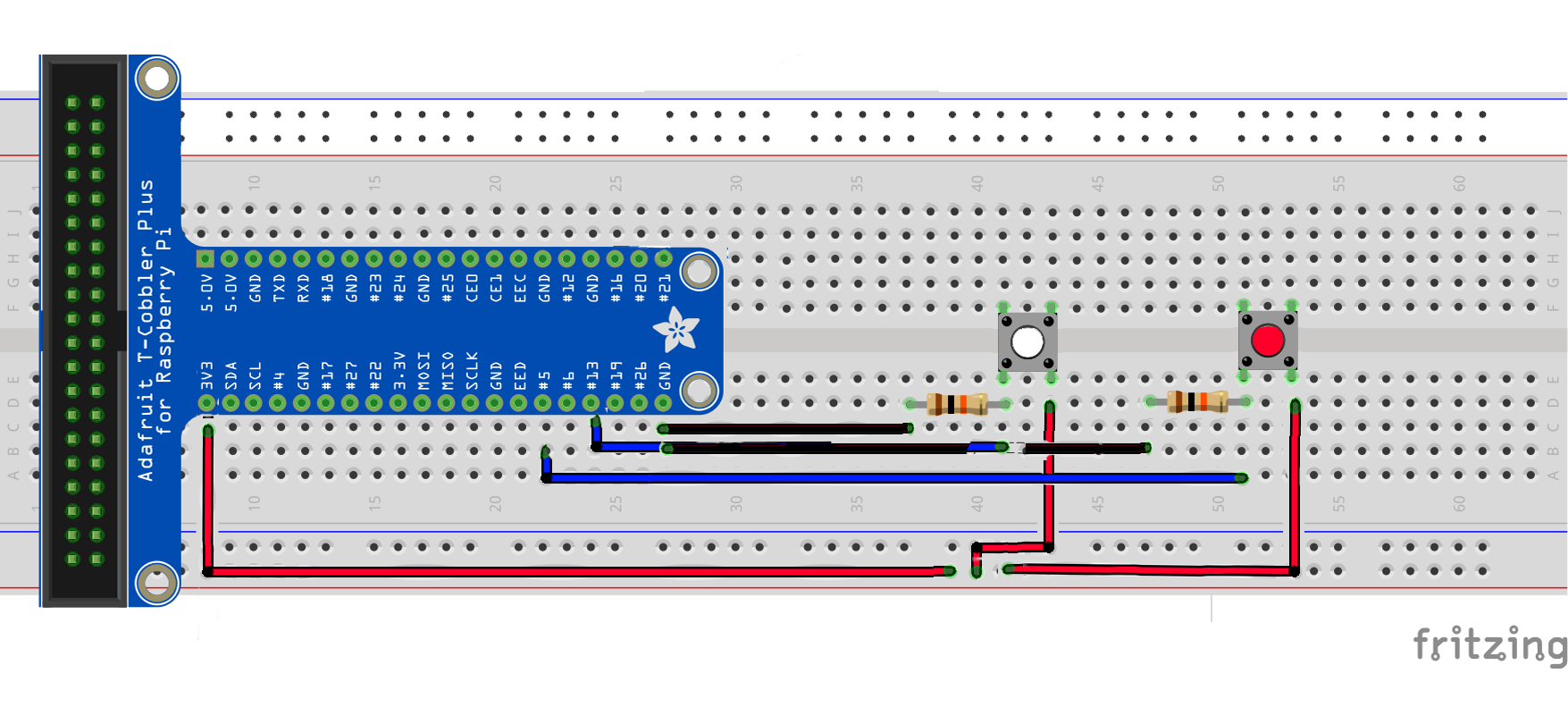
### Connect the LEDs



### Connect the Buzzer



### Connect the Buttons



# Section 4 Appplication Prerequisites

All Packages and Directories Required

|  |  |  |
| --- | --- | --- |
|  | Doorbell RPI Task(s) | |
|  | Make a directory and save ALL files in the directory  mkdir ~/doorbell/ |
|  | Make directories for application to save images  Mkdir -p /home/pi/Desktop/temp /home/pi/Desktop/thieftemp |
|  | Install Botocore and Boto3 (https://github.com/boto/boto)  sudo pip install boto3  sudo pip install botocore |
|  | Install awscli  sudo pip install awscli |
|  | Install the Telegram API on your Raspberry Pi (http://telepot.readthedocs.io/en/latest/)  sudo pip install telepot |
|  | Install threading  sudo pip install threading |

|  |  |  |
| --- | --- | --- |
|  | Shoe Cabinet RPI Task(s) | |
|  | Make a directory and save ALL files in the directory  mkdir ~/shoecabinet/ |

# Section 5 AWS IoTcore (MQTT)

Set-up Amazon Web Service IoTcore service

|  |  |  |
| --- | --- | --- |
|  | Create a Thing in AWS | |
|  | Login to aws using your credentials. |  |
|  | Click on Services and search for IoTcore |  |
|  | Open the navigations on the left. Click on Manage, and then Things. |  |
|  | Click on “Create” on the top right corner. Then, click on Create a single thing |  |
|  | In the field for name, enter whichever name you desire. Leave the other fields as default. Then click “Next” |  |
|  | Proceed to create a certificate using the “One-Click Certificate Creation” |  |
|  | Download the certificates, as well as the rootCA certificate and store them in the Project directory. After which, click on “Activate”. |  |
|  | When all is completed, click on “done” to return to the thing selection page. Then, click on the newly created thing. |  |
|  | On the left hand side, click on interact, then copy the REST API endpoint onto a notepad for use later. |  |

# Section 6 AWS DynamoDB

Set-up Amazon Web Service NoSQL DynamoDB service

|  |  |  |
| --- | --- | --- |
|  | Set up and store data in the light database | |
|  | Login to aws using your credentials. |  |
|  | Open the AWS IOT console, click on “ACT” and click “Create”. |  |
|  | Name it as “light”, “\*” for Attribute and “sensors/light” for Topic filter.  Under “Set one or more actions”, click “Add Action” and select the “Split message into multiple columns of a database table (DynamoDBv2), then click “configure action” |  |
|  | Click on “Create a new resource” to create one. It will open up a new tab on DynamoDB console. Click on “Create table”.  Fill the Table name as “Lights”, and Partition key/Primary key as “date”, and sort as “time”. |  |
|  | Click on “Create”. After create a table, back to your “configure action” and refresh that page.  Select the “LightSensor” under the Table name and choose “my-iot-role” as for the IAM role name. Then, click “Add Action”. |  |

|  |  |  |
| --- | --- | --- |
|  | Set up and store data in the logs database | |
|  | Login to aws using your credentials. |  |
|  | Open the AWS IOT console, click on “ACT” and click “Create”. |  |
|  | Name it as “logs”, “\*” for Attribute and “sensors/facescan” for Topic filter.  As before, under “Set one or more actions”, click “Add Action” and select the “Split message into multiple columns of a database table (DynamoDBv2), then click “configure action” |  |
|  | Click on “Create a new resource” to create one. It will open up a new tab on DynamoDB console. Click on “Create table”.  Fill the Table name as “Logs”, and Partition key/Primary key as “date”, and sort as “event”. |  |
|  | Click on “Create”. After create a table, back to your “configure action” and refresh that page.  Select the “Logs” under the Table name and choose “my-iot-role” as for the IAM role name. Then, click “Add Action”. |  |

|  |  |  |
| --- | --- | --- |
|  | Set up and store data in the passcode database | |
|  | Login to aws using your credentials. |  |
|  | Open the AWS IOT console, click on “ACT” and click “Create”. |  |
|  | Name it as “passcode”, “\*” for Attribute and “doorbell/pass” for Topic filter.  As before, under “Set one or more actions”, click “Add Action” and select the “Split message into multiple columns of a database table (DynamoDBv2), then click “configure action” |  |
|  | Click on “Create a new resource” to create one. It will open up a new tab on DynamoDB console. Click on “Create table”.  Fill the Table name as “Passcode”, and Partition key/Primary key as “passid". |  |
|  | Click on “Create”. After create a table, back to your “configure action” and refresh that page.  Select the “Passcode” under the Table name and choose “my-iot-role” as for the IAM role name. Then, click “Add Action”. |  |

|  |  |  |
| --- | --- | --- |
|  | Set up and store data in the visitors log database | |
|  | Login to aws using your credentials. |  |
|  | Open the AWS IOT console, click on “ACT” and click “Create”. |  |
|  | Name it as “visitors”, “\*” for Attribute and “doorbell/img” for Topic filter.  As before, under “Set one or more actions”, click “Add Action” and select the “Split message into multiple columns of a database table (DynamoDBv2), then click “configure action” |  |
|  | Click on “Create a new resource” to create one. It will open up a new tab on DynamoDB console. Click on “Create table”.  Fill the Table name as “VisitorLogs”, and Partition key/Primary key as “date", and the sort key as “time”. |  |
|  | Click on “Create”. After create a table, back to your “configure action” and refresh that page.  Select the “Passcode” under the Table name and choose “my-iot-role” as for the IAM role name. Then, click “Add Action”. |  |

# Section 7 AWS S3

Set-up Amazon Web Service Image Repository S3

|  |  |  |
| --- | --- | --- |
|  | Get Own Chat ID | |
|  | Under the “Services” navigation, search “s3” and navigate to the AWS S3 |  |
|  | Next, click on “Create Bucket”. |  |
|  | Enter a unique name for your new bucket. (Keep track of the bucket names that you enter.) After which, click on “Create” on the bottom left corner. |  |
|  | Next, repeat from step b) to create another container. (Keep track of both the container names as we will be using it later) |  |
|  | When the buckets are created, enter the bucket and upload any image into the bucket, then click on the image and take note of the “Link” in the overview.  The “Link” without the image name will be your bucket url. For this case, my bucket url will be <https://s3-ap-southeast-1.amazonaws.com/dexjosh-doorcam/>  Repeat this step for both buckets to get the bucket links that will be used later. |  |

# Section 8 Telegram Bot

Set-up Telegram Bot

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Creating a Telegram Bot | | | | |
|  | Open Telegram app in your laptop or mobile and start “BotFather” | | | |  |
|  | Type /newbot to create a new bot | | | |  |
|  | Give a name to your bot that describes its purpose. | |  | | |
|  | Give a username to your bot. Make sure it ends with \_bot | | |  | |
|  | Copy down the access token that is issued by BotFather |  | | | |

|  |  |  |
| --- | --- | --- |
|  | Get Own Chat ID | |
|  | Open Telegram app in your laptop or mobile and search for @get\_id\_bot |  |
|  | Type /start to get your chat id |  |
|  | Record down the chat id for later use. |  |

# Section 9 OpenCV Facial Recognition

Set-up OpenCV Facial Recognition

For the facial recognition software, we will be using OpenCV (Open Source Computer Vision Library) with the PiCam.

*Note: The first part of this tutorial is based on an online tutorial by Adrian Rosebrock on pyimagesearch, available at: https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/*

Firstly, we would need to expand the filesystem to include all the available space on the microSD card. This is because the software takes up quite a big amount of space.



Select ‘Advanced Options’ in the menu and select ‘Expand filesystem’. You will then be guided alont to run through the process of expanding the filesystem. After which, reboot the system.



After rebooting, we will now install the dependencies. After updating and upgrading all existing packages, we will install several developer tools, such as CMake, and several OpenCV requirements.



After the dependencies have been installed, we can now download the OpenCV source code.



We would also need to download the opencv\_contrib repository as well, as we will want the full install of OpenCV 3 for all of its various features.



We will also install pip, the Python package manager.



We will be using a virtualenv for our program. This is a standard practice for Python programming, and as such, we will be using it as well.



We will need to add the following to our *~/.profile* file.



In our command prompt, we will need to reload the *~/.profile* file for it to take effect. We can do so by executing:



After this, we can now make our virtualenv in python3.



Refresh the *~/.profile* file once again, and we can work on our newly created virtual environment.



You will know if you are in the virtual environment when you see a “(cv)” at the side of your command line.



Now that we are in the virtual environment, we can install NumPy, used for numerical processing.



After completion, we can finally make, compile and install OpenCV.



Before we compile, we will have to expand the swap space size of the raspberry pi, be editing /etc/dphys-swapfile and changing the ‘CONF\_SWAPSIZE’ variable to ‘1024’.

After which, we can restart the swap service by running:



Now, we’re ready to compile OpenCV.



Compiling OpenCV alone will take a very long time, as the Raspberry Pi is not the most powerful of machines. Compiling alone took about 3-4 hours for me personally, while the entire process took me a whole day! After it’s done, we can (finally) install OpenCV 3.



After the install, we should check /usr/local/lib/python3.6/site-packages for the cv2.s file. It should be something along the lines of ‘cv2.cpython-xxxx.so’. However, it should be cv2.so alone instead. Apparently, when installed for python3, it will have this issue. However, it is not difficult to correct that, by just renaming the file.



After renaming, we can sym-link the cv2.so file to our cv virtual environment.



Now, we can change our swapfile back to the original size of 100. Just reverse the process we have done earlier.

*Note: this second part is based off of ‘MJRoBot’/’Mjrovai’s’ tutorial on hackster.io, which is available at:   
https://www.hackster.io/mjrobot/real-time-face-recognition-an-end-to-end-project-a10826*

*However, I modified the code to suit our solution.*

We will also need to download Cascades files, ‘haarcascades’, from the OpenCV Github. This will enable the detection of faces. They are available at:

<https://github.com/opencv/opencv/tree/master/data/haarcascades/>

The specific xml file that we will need is ‘haarcascade\_frontalface\_default.xml’. It will definitely be alright if you just download that. However, I downloaded all of it to experiment around. These XML files will allow the camera to detect various different things, from eyes, to your full body. These files will be required to be in a folder called ‘haarcascades’ in your project directory.

There are three parts to the facial recognition solution:

* + 1. Face Scanning
    2. Face Training
    3. Face Recognition

Face Scanning, uses the OpenCV software to detect and scan for faces, saving multiple images of the face.

It will then use the Face Training code in order to save the faces into a *.yml* file.

This *.yml* file will then be used by the Face Recognition code to identify the face based off of confidence, meaning that the software will give a percentage of confidence that the face matches the one stored in the *.yml* file.

Now that we have installed the OpenCV software, we can now proceed to install the other software in the virtual environment. As the virtual environment is isolated from the Raspberry Pi’s main raspbian installation, we would also be required to install Flask, gevent, gpiozero, MySQL, and several other important software.



MySQL will prompt you to configure the root passwords and the like. Just follow the guide and it should be alright.

Now that we have installed MySQL, we need to setup the database for our project.



After logging in, we shall create an assignment database.



After which, we will create and assignment user to access the database, giving the user the password ‘joshanddexpassword.



Now, we can log in to the database and create the necessary tables, and insert the necessary data.



The data that we inserted into the Security table is firstly the passcode for the button combinations and the confidence level of the facial recognition that will allow the user to login.

# Section 10 Coding

Code the Programs

In this section, we will be creating five python files, one for the doorbell RPI and four for the shoe cabinet RPI.

1. Doorbell.py (used by gate security RPI)
2. Shoecabinet.py (used by shoe cabinet RPI)
3. Faceid.py (used by shoe cabinet RPI; to register faces in the database)
4. Trainer.py (used by shoe cabinet RPI; to train faces into the database)
5. Server.py (used by shoecabinet RPI; to host the web application server)

Take note of all the highlighted codes in the source code. There are the codes that will require you to modify, all the methods to get the codes will be available on the previous sections, so make sure that you complete all previous sections before attempting this section.

|  |  |
| --- | --- |
|  | Doorbell.py |
|  | Create a python script server.py with the code below  sudo nano ~/doorbell/doorbell.py |
|  | import boto3  import datetime  import botocore  import json  import threading  import time  import telepot  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  from picamera import PiCamera  from time import sleep  from gpiozero import Button, LED, Buzzer  from rpi\_lcd import LCD  from boto3.dynamodb.conditions import Key, Attr  #declare  doorbell = Button(5, pull\_up=False)  btn1 = Button(13, pull\_up=False)  btn2 = Button(6, pull\_up=False)  buzzer = Buzzer(19)  camera = PiCamera()  lcd = LCD()  redled = LED(23)  greenled = LED(24)  s3 = boto3.resource('s3')  tele\_token = 'enter your telegram auth token'  bot = telepot.Bot(tele\_token)  chat\_id = 'enter you chat id'  #connect to aws mqtt broker  try:  host = "enter your amw thing rest api endpoint"  rootCAPath = "rootca.pem" #enter rootca file name  certificatePath = "certificate.pem.crt" #enter certificate file name  privateKeyPath = "private.pem.key" #enter private key file name  my\_rpi = AWSIoTMQTTClient("doorbell")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1)  my\_rpi.configureDrainingFrequency(2)  my\_rpi.configureConnectDisconnectTimeout(10)  my\_rpi.configureMQTTOperationTimeout(5)  my\_rpi.connect()  except Exception as e:  print('Fatal Error %s' % e)  def doorbellpress():  print('Waiting for doorbell press...')  while True:  if doorbell.is\_pressed:  buzzer.on()  sleep(2)  buzzer.off()  ts = time.time()  file\_name = 'img'+datetime.datetime.fromtimestamp(ts).strftime('%H%M%S')+'.jpg'  full\_path = '/home/pi/Desktop/temp/'+file\_name  camera.capture(full\_path)  #upload picture to s3  bucket\_name='dexjosh-doorcam' #enter bucket 1 name  s3.Object(bucket\_name, file\_name).put(Body=open(full\_path, 'rb'),ContentType='image/jpeg',ACL='public-read')  s3link = 'https://s3-ap-southeast-1.amazonaws.com/dexjosh-doorcam/'+file\_name # modify with bucket 1 url  #set message payload  st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')  payload = json.dumps ({  "timestamp": st,  "imglink": s3link  })  #publish to broker  my\_rpi.publish("doorbell/img", payload, 1)  #send to tele  bot.sendPhoto(chat\_id, open(full\_path, 'rb'))  bot.sendMessage(chat\_id, text='Someone is at your door!')  bot.sendMessage(chat\_id, text='Go to website to accept/deny.')  sleep(5)  def waitforreply():  def customCallback(client, userdata, message):  payload\_dict = json.loads(message.payload)  if payload\_dict['message'] == 'granted':  print('guest allowed.')  lcd.clear()  lcd.text('Access granted.', 1)  lcd.text('Come in!', 2)  greenled.on()  sleep(5)  lcd.clear()  greenled.off()  if payload\_dict['message'] == 'denied':  print('guest denied.')  lcd.clear()  lcd.text('You are not', 1)  lcd.text('welcomed.', 2)  redled.on()  sleep(5)  lcd.clear()  redled.off()  print('Waiting for user response...')  while True:  my\_rpi.subscribe("doorbell/entry", 1, customCallback)  def catchthief():  def customCallback2(client, userdata, message):  payload\_dict = json.loads(message.payload)  if payload\_dict:  buzzer.on()  sleep(5)  buzzer.off()  ts = time.time()  file\_name = payload\_dict['date']+payload\_dict['time']+'.jpg'  full\_path = '/home/pi/Desktop/thieftemp/'+file\_name  camera.capture(full\_path)  bucket\_name='dexjosh-thief' #enter bucket 2 name  s3.Object(bucket\_name, file\_name).put(Body=open(full\_path, 'rb'),ContentType='image/jpeg',ACL='public-read')  s3link = 'https://s3-ap-southeast-1.amazonaws.com/dexjosh-thief/'+file\_name #enter bucket 2 url  payload = json.dumps ({  "date": payload\_dict['date'],  "time": payload\_dict['time'],  "event": payload\_dict['event'],  "capture": s3link  })  my\_rpi.publish("sensors/facescan", payload, 1)  bot.sendPhoto(chat\_id, open(full\_path, 'rb'))  bot.sendMessage(chat\_id, text='Someone is stealing your shoes!')  bot.sendMessage(chat\_id, text='We have captured his face, call the police.')  print('Ready for theft capture...')  while True:  my\_rpi.subscribe("doorbell/theft", 1, customCallback2)  sleep(5)  t1 = threading.Thread(target=waitforreply)  t2 = threading.Thread(target=doorbellpress)  t3 = threading.Thread(target=catchthief)  print('starting wait for response...')  t1.start()  print('starting wait for doorbell...')  t2.start()  print('starting catch theif mechanism...')  t3.start()  print('Initializing passcode system...')  # Helper class to convert a DynamoDB item to JSON.  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('Passcode')  Passresponse = table.query(  KeyConditionExpression=Key('passid').eq(1)  )  for i in Passresponse['Items']:  passcode = i.get('passcode')  print('Passcode is: '+ str(passcode))  password = [int(i) for i in str(passcode)]  userpass = []  print('please key the passcode')  def buttonOne():  print("Button 1 pressed")  userPass(1)    def buttonTwo():  print("Button 2 pressed")  userPass(2)    def userPass(number):  print("Number received: " + str(number))  userpass.append(number)  print(userpass)  def checkPass(userpass, password):  if userpass == password:  result = True  else:  result = False    return result    while True:  lcd.text('Please Enter \nPasscode!', 1)  btn1.when\_pressed = buttonOne  btn2.when\_pressed = buttonTwo    if len(userpass) == len(password):  lcd.clear()  lcd.text('Authenticating...', 1)  sleep(1)  result = checkPass(userpass, password)    if result is True:  lcd.text('Passcode', 1)  lcd.text('Correct!', 2)  buzzer.on()  greenled.on()  sleep(2)  greenled.off()  buzzer.off()  lcd.text('Access', 1)  lcd.text('Granted!', 2)  break  else:  lcd.text('Passcode', 1)  lcd.text('Incorrect!', 2)  buzzer.on()  redled.on()  sleep(2)  redled.off()  buzzer.off()  userpass = []  elif len(userpass) > len(password):  lcd.clear()  lcd.text('Please Try Again', 1)  sleep(1)  userpass = [] |

|  |  |
| --- | --- |
|  | shoecabinet.py |
|  | Create a python script shoecabinet.py with the code below  sudo nano ~/shoecabinet/shoecabinet.py |
|  | from gpiozero import Button, MCP3008, LED, Buzzer  from rpi\_lcd import LCD  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  from time import sleep  import json  import cv2  import numpy as np  import os  import MySQLdb  from time import sleep  import string  import time, datetime  from picamera import PiCamera  import subprocess  adc = MCP3008(channel=0)  whitebutton = Button(13, pull\_up=False)  redbutton = Button(5, pull\_up=False)  lcd = LCD()  lcd.clear()  dbaction = None  timeout = None  host = "enter your amw thing rest api endpoint"  rootCAPath = "rootca.pem" #enter rootca file name  certificatePath = "certificate.pem.crt" #enter certificate file name  privateKeyPath = "private.pem.key" #enter private key file name  my\_rpi = AWSIoTMQTTClient("joshpubsub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()  def lightcheck():  lightvalue = adc.value  return lightvalue  def greenLED():  greenled = LED(23)  greenled.on()  sleep(1)  greenled.off()  greenled.close()  def redLED():  redled = LED(18)  redled.on()  sleep(1)  redled.off()  redled.close()  def buzzBuzz():  buzz = Buzzer(21)  buzz.on()  sleep(1)  buzz.off()  buzz.close()    while True:  lcd.text("White: Come Home", 1)  lcd.text("Red: Leave House", 2)  whitepress = whitebutton.is\_pressed  redpress = redbutton.is\_pressed  if whitepress is True:  dbaction = 'incoming'  break  elif redpress is True:  dbaction = 'outgoing'  break  else:  lightvalue = lightcheck()  dbdata = {  "date": str(datetime.date.today()),  "time": str(datetime.datetime.now().time()),  "lightvalue": str(lightvalue)  }  dbsend = json.dumps(dbdata)  my\_rpi.publish("sensors/light", str(dbsend), 1)  print(dbsend)  print('LIGHT')  print(lightvalue)  if lightvalue < 0.5:  timestamp = datetime.datetime.now()  if timeout is not None:  timediff = timestamp - timeout    if timediff > datetime.timedelta(minutes = 3):  logsdata = {  "date": str(datetime.date.today()),  "time": str(datetime.datetime.now().time()),  "event": "thief!"  }  logsdbsend = json.dumps(logsdata)  print(dbsend)  my\_rpi.publish("sensors/facescan", str(logsdbsend), 1)  my\_rpi.publish("doorbell/theft", str(logsdbsend), 1)  timeout = timestamp  else:  buzzBuzz()  logsdata = {  "date": str(datetime.date.today()),  "time": str(datetime.datetime.now().time()),  "event": "thief!"  }  logsdbsend = json.dumps(logsdata)  print(dbsend)  my\_rpi.publish("sensors/facescan", str(logsdbsend), 1)  my\_rpi.publish("doorbell/theft", str(logsdbsend), 1)  timeout = timestamp  lcd.clear()  whitebutton.close()  redbutton.close()  adc.close()  subprocess.call(["sudo", "modprobe", "bcm2835-v4l2"])  lcd = LCD()  lcd.text('Initializing', 1)  lcd.text('Face Scan...', 2)  recognizer = cv2.face.LBPHFaceRecognizer\_create()  recognizer.read('../trainer/trainer.yml')  cascadePath = "../haarcascades/haarcascade\_frontalface\_default.xml"  faceCascade = cv2.CascadeClassifier(cascadePath);  font = cv2.FONT\_HERSHEY\_SIMPLEX  #iniciate id counter  id = 0  try:  db = MySQLdb.connect("localhost", "assignmentuser", "joshanddexpassword", "assignment")  curs = db.cursor()  print("Successfully connected to database!")  except:  print("Error connecting to mySQL database")  sql = "SELECT Username FROM assignment.Users"  curs.execute(sql)  result = curs.fetchall()  userslist = ['None']  for x in range(0, len(result)):  userslist.append(result[x][0])  # Initialize database  sql = "SELECT FaceScanConfidence FROM assignment.Security"  curs.execute(sql)  result = curs.fetchall()  setconfidence = int(result[0][0])  # Initialize and start realtime video capture  cam = cv2.VideoCapture(0)  cam.set(3, 640) # set video widht  cam.set(4, 480) # set video height  # Define min window size to be recognized as a face  minW = 0.1\*cam.get(3)  minH = 0.1\*cam.get(4)  lcd.clear()  lcd.text('Scanning...', 1)  starttime = time.time()  while True:  confidentint = 0  ret, img =cam.read()  img = cv2.flip(img, -1) # Flip vertically    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  faces = faceCascade.detectMultiScale(  gray,  scaleFactor = 1.2,  minNeighbors = 5,  minSize = (int(minW), int(minH)),  )    for(x,y,w,h) in faces:  cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)  id, confidence = recognizer.predict(gray[y:y+h,x:x+w])    # Check if confidence is less them 100 ==> "0" is perfect match  if (confidence < 100):  id = userslist[id]  confidentint = round(100 - confidence)  print(confidentint)  confidence = " {0}%".format(round(100 - confidence))  else:  id = "unknown"  confidence = " {0}%".format(round(100 - confidence))    cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255), 2)  cv2.putText(img, str(confidence), (x+5,y+h-5), font, 1, (255,255,0), 1)  #cv2.imshow('camera',img)  if confidentint > setconfidence:  lcd.text('Identity', 1)  lcd.text('Confirmed!', 2)  buzzBuzz()  greenLED()  lcd.clear()  if dbaction == 'incoming':  lcd.text('Welcome Home,', 1)  lcd.text(str(id) + '!', 2)  else:  lcd.text('Goodbye,' + str(id), 1)    logsdata = {  "date": str(datetime.date.today()),  "time": str(datetime.datetime.now().time()),  "event": dbaction,  "facescan result": "success",  "identity": id  }  dbsend = json.dumps(logsdata)  print(dbsend)  my\_rpi.publish("sensors/facescan", str(dbsend), 1)  sleep(5)  break    nowtime = time.time()  timediff = nowtime - starttime  print(timediff)  if timediff > 30:  redLED()  buzzBuzz()  cam.release()  cv2.destroyAllWindows()  lcd.text('Identity', 1)  lcd.text('Unconfirmed!', 2)  logsdata = {  "date": str(datetime.date.today()),  "time": str(datetime.datetime.now().time()),  "event": dbaction,  "facescan result": "fail"  }  dbsend = json.dumps(logsdata)  print(dbsend)  my\_rpi.publish("sensors/facescan", str(dbsend), 1)  sleep(5)  break  sleep(60)  lcd.clear()  import program |

|  |  |
| --- | --- |
|  | faceid.py |
|  | Create a python script faceid.py with the code below  sudo nano ~/shoecabinet/faceid.py |
|  | import cv2, os, string, MySQLdb  def face(userid):  cam = cv2.VideoCapture(0)  cam.set(3, 640) # set video width  cam.set(4, 480) # set video height  facedetector = cv2.CascadeClassifier('../haarcascades/haarcascade\_frontalface\_default.xml')  print("\nInitializing Face Capture. Please look at the camera.")  facecount = 0  while(True):  ret, img = cam.read()  img = cv2.flip(img, -1) # flip video image vertically  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  faces = facedetector.detectMultiScale(gray, 1.3, 5)  for (x,y,w,h) in faces:  cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)  facecount += 1  progress = facecount / 30 \* 100  print("Progress: " + '{0:.2f}'.format(progress) +"%")  # Save the captured image into the datasets folder  cv2.imwrite("../data/UserID-" + str(userid) + '-' + str(facecount) + ".jpg", gray[y:y+h,x:x+w])    k = cv2.waitKey(100) & 0xff  if k == 27:  break  elif facecount >= 30:  break  print("\nCapture Complete")  cam.release()  cv2.destroyAllWindows()  print("\nTraining Faces")  import trainer |

# 

|  |  |
| --- | --- |
|  | trainer.py |
|  | Create a python script trainer.py with the code below  sudo nano ~/shoecabinet/trainer.py |
|  | import cv2  import numpy as np  from PIL import Image  import os  path = '../data'  recognizer = cv2.face.LBPHFaceRecognizer\_create()  detector = cv2.CascadeClassifier("../haarcascades/haarcascade\_frontalface\_default.xml");  # function to get the images and label data  def getImagesAndLabels(path):  imagePaths = [os.path.join(path,f) for f in os.listdir(path)]  faceSamples=[]  ids = []  for imagePath in imagePaths:  PIL\_img = Image.open(imagePath).convert('L') # convert it to grayscale  img\_numpy = np.array(PIL\_img,'uint8')  id = int(os.path.split(imagePath)[-1].split("-")[1])  faces = detector.detectMultiScale(img\_numpy)    for (x,y,w,h) in faces:  faceSamples.append(img\_numpy[y:y+h,x:x+w])  ids.append(id)  return faceSamples,ids  print ("\nTraining Faces. Please wait...")  faces,ids = getImagesAndLabels(path)  recognizer.train(faces, np.array(ids))  # Save the model into trainer/trainer.yml  recognizer.write('../trainer/trainer.yml') # recognizer.save() worked on Mac, but not on Pi  # Print the numer of faces trained and end program  print("\n{0} faces trained. Bye".format(len(np.unique(ids)))) |

|  |  |
| --- | --- |
|  | server.py |
|  | Create a python script server.py with the code below  sudo nano ~/shoecabinet/server.py |
|  | import datetime  import gevent  import gevent.monkey  from gevent.pywsgi import WSGIServer  import MySQLdb  import boto3  import json  import decimal  from boto3.dynamodb.conditions import Key, Attr  from flask import Flask, request, Response, render\_template  from flask\_bootstrap import Bootstrap  from gpiozero import LED  from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  app = Flask(\_\_name\_\_)  Bootstrap(app)  @app.route("/")  def index():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('VisitorLogs')  response = table.scan()  data = []  final = None  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  timestamp = str(str(date) + ' ' + str(time))  comparetime = datetime.datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')  if final is not None:  result = comparetime > final  if result is True:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')  else:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')    d.append(final)  d.append(imglink)  d.append(accepted)  data.append(d)  return render\_template('index.html', data=data[0])  @app.route("/accept/")  def acceptVisitor():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('visitor\_log')  response = table.scan()  data = []  final = None  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  timestamp = str(str(date) + ' ' + str(time))  comparetime = datetime.datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')  if final is not None:  result = comparetime > final  if result is True:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')  else:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')    d.append(final)  d.append(imglink)  d.append(accepted)  data.append(d)    host = "a33pwtpx7h9igb.iot.us-west-2.amazonaws.com"  rootCAPath = "../keys/rootca.pem"  certificatePath = "../keys/certificate.pem.crt"  privateKeyPath = "../keys/private.pem.key"  my\_rpi = AWSIoTMQTTClient("joshpubsub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()    dbdata = {  "date": str(date),  "time": str(time),  "imglink": imglink,  "accepted": "yes"  }  dbsend = json.dumps(dbdata)  my\_rpi.publish("doorbell/img", str(dbsend), 1)    my\_rpi.publish("doorbell/entry", "granted", 1)    return render\_template('index.html', data=data[0])    @app.route("/reject/")  def rejectVisitor():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('visitor\_log')  response = table.scan()  data = []  final = None  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  timestamp = str(str(date) + ' ' + str(time))  comparetime = datetime.datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')  if final is not None:  result = comparetime > final  if result is True:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')  else:  final = comparetime  imglink = i.get('imglink')  accepted = i.get('accepted')    d.append(final)  d.append(imglink)  d.append(accepted)  data.append(d)    host = "a33pwtpx7h9igb.iot.us-west-2.amazonaws.com"  rootCAPath = "../keys/rootca.pem"  certificatePath = "../keys/certificate.pem.crt"  privateKeyPath = "../keys/private.pem.key"  my\_rpi = AWSIoTMQTTClient("joshpubsub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()    dbdata = {  "date": str(date),  "time": str(time),  "imglink": imglink,  "accepted": "no"  }  dbsend = json.dumps(dbdata)  my\_rpi.publish("doorbell/img", str(dbsend), 1)  my\_rpi.publish("doorbell/entry", "denied", 1)  return render\_template('index.html', data=data[0])    @app.route("/viewLight/")  @app.route("/viewLight/realtime/")  def viewLightRT():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('Lights')  response = table.scan()  data = []  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  lightvalue = i.get('lightvalue')  d.append(date)  d.append(time)  d.append(int(float(lightvalue) \* 1024))  data.append(d)  data\_reversed = data[::-1]  return render\_template('lights.html', data=data\_reversed)    @app.route("/viewLight/historic/")  def viewLightHistoricRouter():  return render\_template('router.html')  @app.route("/viewLight/historic/<date>")  def viewLightHistoric(date):  date = str(date)  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('Lights')  data = []  response = table.query(  KeyConditionExpression=Key('date').eq(date)  )  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  lightvalue = i.get('lightvalue')  d.append(date)  d.append(time)  d.append(int(float(lightvalue) \* 1024))  data.append(d)  data\_reversed = data[::-1]  return render\_template('lights.html', data=data\_reversed)    @app.route("/viewLogs/")  def viewUserLogs():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('Logs')  response = table.scan()  data = []  for i in response['Items']:  currentval = i.get('event')  if currentval == 'incoming' or currentval == 'outgoing':  d = []  date = i.get('date')  time = i.get('time')  event = i.get('event')  facescan = i.get('facescan result')  identity = i.get('identity')  d.append(date)  d.append(time)  d.append(event)  d.append(facescan)  d.append(identity)  data.append(d)    data\_reversed = data[::-1]  print(data\_reversed)  return render\_template('userlog.html', data = data\_reversed)  @app.route("/viewThief/")  def viewThief():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('Logs')  response = table.scan()  data = []  for i in response['Items']:  currentval = i.get('event')  if currentval == 'thief!':  d = []  date = i.get('date')  time = i.get('time')  event = i.get('event')  capture = i.get('capture')  d.append(date)  d.append(time)  d.append(event)  d.append(capture)  data.append(d)    data\_reversed = data[::-1]  print(data\_reversed)  return render\_template('thief.html', data = data\_reversed)  @app.route("/viewVisitorLogs/")  def visitorLog():  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('VisitorLogs')  response = table.scan()  data = []  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  imglink = i.get('imglink')  accepted = i.get('accepted')  d.append(date)  d.append(time)  d.append(imglink)  d.append(accepted)  data.append(d)  print(data)  data\_reversed = data[::-1]  return render\_template('visitorlog.html', data=data\_reversed)  @app.route("/viewVisitorLogs/search/")  def visitorLogsSearchRouter():  return render\_template('visitorrouter.html')    @app.route("/viewVisitorLogs/search/<date>/")  def visitorLogSearch(date):  date = str(date)  dynamodb = boto3.resource('dynamodb')  table = dynamodb.Table('VisitorLogs')  response = table.scan()  data = []  response = table.query(  KeyConditionExpression=Key('date').eq(date)  )  for i in response['Items']:  d = []  date = i.get('date')  time = i.get('time')  imglink = i.get('imglink')  accepted = i.get('accepted')  d.append(date)  d.append(time)  d.append(imglink)  d.append(accepted)  data.append(d)  data\_reversed = data[::-1]  return render\_template('visitorlog.html', data=data\_reversed)  @app.route("/changePassword/")  def changePassword():  return render\_template('changepassword.html')      @app.route("/changePassword/<passcode>/")  def changePasswordCommit(passcode):  passcode = str(passcode)  host = "a33pwtpx7h9igb.iot.us-west-2.amazonaws.com"  rootCAPath = "../keys/rootca.pem"  certificatePath = "../keys/certificate.pem.crt"  privateKeyPath = "../keys/private.pem.key"  my\_rpi = AWSIoTMQTTClient("joshpubsub")  my\_rpi.configureEndpoint(host, 8883)  my\_rpi.configureCredentials(rootCAPath, privateKeyPath, certificatePath)  my\_rpi.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing  my\_rpi.configureDrainingFrequency(2) # Draining: 2 Hz  my\_rpi.configureConnectDisconnectTimeout(10) # 10 sec  my\_rpi.configureMQTTOperationTimeout(5) # 5 sec  my\_rpi.connect()    dbdata = {  "passid": 1,  "passcode": passcode  }  dbsend = json.dumps(dbdata)  my\_rpi.publish("doorbell/pass", str(dbsend), 1)    return render\_template('passwordchanged.html')  @app.route("/registerFace/")  def registerFaceForm():  return render\_template('facescan.html')  @app.route('/registerFace/<userid>/<username>')  def registerFace(userid, username):  from faceid import face  face(userid)    try:  db = MySQLdb.connect("localhost", "assignmentuser", "joshsmartroom", "assignment")  curs = db.cursor()  print("Successfully connected to database!")  except:  print("Error connecting to mySQL database")  try:  sql = "INSERT into Users(UserID, Username) VALUES ('%d', '%s')" % (int(userid), str(username))  curs.execute(sql)  db.commit()  print('\nDatabase Modified')  except MySQLdb.Error as e:  print(e)    return render\_template('faceregistered.html')    @app.route("/changeFaceUnlockConfidence/")  def changeConfidence():  return render\_template('changeconfidence.html')    @app.route("/changeFaceUnlockConfidence/<value>")  def changeConfidenceDB(value):  value = int(value)  try:  db = MySQLdb.connect("localhost", "assignmentuser", "joshsmartroom", "assignment")  curs = db.cursor()  print("Successfully connected to database!")  except:  print("Error connecting to mySQL database")  try:  sql = "UPDATE Security SET FaceScanConfidence = %d WHERE ID = 1;" % (value)  print(sql)  curs.execute(sql)  db.commit()  print('\nDatabase Modified')  except MySQLdb.Error as e:  print(e)  return render\_template('confidencechanged.html')    if \_\_name\_\_ == '\_\_main\_\_':  try:  http\_server = WSGIServer(('0.0.0.0', 8001), app)  app.debug = True  http\_server.serve\_forever()  except:  print("Exception") |

Apart from that, we will need to have the HTML templates for the server.py file. All the templates are to be put in *~/shoecabinet/templates/.*

|  |  |
| --- | --- |
|  | Index.html |
|  | Create a python script index.html with the code below  sudo nano ~/shoecabinet/templates/index.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  KremePi Home  {% endblock %}  <head>  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>  </head>  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3 class="card-title"><b>Lastest Visitor!</b></h3>  <h3>Date/Time: {{data.0}} </h3>  <img src="{{data.1}}" style="width:100%;"></img>  <br>  {% if data.2 is none %}  <h3 style="border: 1px solid black; padding: 1%; background-color: white;">  <a href="/accept/" style="">Accept</a>  </h3>  <h3 style="border: 1px solid black; padding: 1%; background-color: white;">  <a href="/reject/">Reject</a>  </h3>  {% endif %}  </div>    <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3><b>Light Sensor Data: </b></h3>  <h3>  <a href="/viewLight/">View Real Time Light Sensor Data</a>  </h3>  <h3>  <a href="/viewLight/historic/">View Historic Light Sensor Data</a>  </h3>  </div>    <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3><b>Entry Logs: </b></h3>  <h3>  <a href="/viewLogs/">View User Logs</a>  </h3>  <h3>  <a href="/viewVisitorLogs/">View Visitor Logs</a>  </h3>  <h3>  <a href="/viewVisitorLogs/search/">Search Visitor History</a>  </h3>  <h3>  <a href="/viewThief/">View Thief Logs</a>  <h3>  </div>    <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%; margin-bottom: 5%;">  <h3><b>Security Settings: </b></h3>  <h3>  <a href="/changePassword/">Change Passcode</a>  </h3>  <h3>  <a href="/changeFaceUnlockConfidence/">Change Face Unlock Confidence</a>  </h3>  <h3>  <a href="/registerFace/">Register Face Unlock</a>  </h3>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | lights.html |
|  | Create a python script lights.py with the code below  sudo nano ~/shoecabinet/templates/lights.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Light Sensor Data  {% endblock %}  {% block scripts %}  <script type="text/javascript" src="https://code.jquery.com/jquery-3.2.1.js"></script>  <script type="text/javascript" src="https://www.google.com/jsapi"></script>  <script type="text/javascript">  google.load('visualization', '1', {'packages':['corechart']});  google.setOnLoadCallback(drawChart);  function drawChart() {  var data = new google.visualization.DataTable();  data.addColumn('string', 'Date/Time');  data.addColumn('number', 'Light Value');  data.addRows([  {%- for date, time, lightvalue in data %}  ['{{ date }} {{time}}', {{ lightvalue }}],  {%- endfor %}  ]);  var chart = new google.visualization.LineChart(  document.getElementById('chart\_div'));  chart.draw(data, {legend: 'none', vAxis: {baseline: 0},  colors: ['#00C7CE', '#2200BC']});  }  </script>  <script>  $(document).ready(function () {  setInterval(function () {  location.reload();  //drawChart();  }, 3000);  });  </script>  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div id="content" style="width: 90%; text-align: center; margin: 0 auto; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h1 style="text-align: center;" ><b>Light Sensor Data</b></h1>  <div id="chart\_div"></div>  <h5>  <a href="/">Go Back</a>  </h5>  <div class="box-body" style="margin-top: 4%; height: 600px; overflow: auto;">  <table class="table table-hover dataTable" role="grid" border="1">  <tr>  <th>Date</th>  <th>Time</th>  <th>Light Value</th>  </tr>  {%- for date, time, lightvalue in data %}  <tr>  <td>{{ date }}</td>  <td>{{ time }}</td>  <td>{{ lightvalue }}</td>  </tr>  {%- endfor %}  </table>  </div>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | router.html |
|  | Create a python script router.py with the code below  sudo nano ~/shoecabinet/templates/router.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Light Sensor History  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3>Search for Light Sensor History by Date:</h3>  <form id="routeform" onSubmit="route();">  <input type="date" name="date" id="date"/>  <input type="button" id="submit" value="Submit"/>  </form>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  {% block scripts %}  <script>  var submit = document.getElementById('submit');  submit.addEventListener('click', function() {  //console.log("clicked")  var date = document.getElementById('date').value;  console.log(date);  var url = window.location.href.concat(date);  window.location = url;  console.log(url);  })  </script>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | userlog.html |
|  | Create a python script userlog.py with the code below  sudo nano ~/shoecabinet/templates/userlog.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  User Logs  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div id="content" style="width: 90%; text-align: center; margin: 0 auto; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3 style="text-align: center;" ><b>User Log</b></h3>  <table class="table table-hover dataTable" role="grid" border="1">  <tr>  <th>Date</th>  <th>Time</th>  <th>Event</th>  <th>Facescan Result</th>  <th>Identity</th>  </tr>  {%- for date, time, event, facescan, identity in data %}  <tr>  <td>{{ date }}</td>  <td>{{ time }}</td>  <td>{{ event }}</td>  <td>{{ facescan }}</td>  <td>{{ identity }}</td>  </tr>  {%- endfor %}  </table>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | visitorlog.html |
|  | Create a python script visitorlog.py with the code below  sudo nano ~/shoecabinet/templates/visitorlog.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Visitor Logs  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div id="content" style="width: 90%; text-align: center; margin: 0 auto; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3 style="text-align: center;" ><b>Visitor Log</b></h3>  <table class="table table-hover dataTable" role="grid" border="1">  <tr>  <th>Date</th>  <th>Time</th>  <th>Image Link</th>  <th>Accepted</th>  </tr>  {%- for date, time, imglink, accepted in data %}  <tr>  <td>{{ date }}</td>  <td>{{ time }}</td>  <td><a href="{{ imglink }}">Click Here.</a></td>  <td>{{ accepted }}</td>  </tr>  {%- endfor %}  </table>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  </body>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | visitorrouter.html |
|  | Create a python script visitorrouter.py with the code below  sudo nano ~/shoecabinet/templates/ visitorrouter.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Vistor Log History  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3>Search for Visitor Log History by Date:</h3>  <form id="routeform" onSubmit="route();">  <input type="date" name="date" id="date"/>  <input type="button" id="submit" value="Submit"/>  </form>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  {% block scripts %}  <script>  var submit = document.getElementById('submit');  submit.addEventListener('click', function() {  //console.log("clicked")  var date = document.getElementById('date').value;  console.log(date);  var url = window.location.href.concat(date);  window.location = url;  console.log(url);  })  </script>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | thief.html |
|  | Create a python script thief.py with the code below  sudo nano ~/shoecabinet/templates/ thief.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Thief Logs  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div id="content" style="width: 90%; text-align: center; margin: 0 auto; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3 style="text-align: center;" ><b>Thief Log</b></h3>  <table class="table table-hover dataTable" role="grid" border="1">  <tr>  <th>Date</th>  <th>Time</th>  <th>Event</th>  <th>Capture Link</th>  </tr>  {%- for date, time, event, capture in data %}  <tr>  <td>{{ date }}</td>  <td>{{ time }}</td>  <td>{{ event }}</td>  <td>  {% if capture is not none %}  <a href="{{ capture }}">  Click Here.  </a>  {% else %}  No Capture Found.  {% endif %}  </td>  </tr>  {%- endfor %}  </table>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | changepassword.html |
|  | Create a python script changepassword.py with the code below  sudo nano ~/shoecabinet/templates/ changepassword.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Change Password  {% endblock %}    {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3>Change Password: </h3>  <p>Please Enter a Password Combination (Either 1 or 2)</p>  <form id="routeform" onSubmit="route();">  <input type="number" name="password" id="password"/>  <input type="button" id="submit" value="Submit"/>  </form>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  {% block scripts %}  <script>  var submit = document.getElementById('submit');  submit.addEventListener('click', function() {  console.log("clicked")  var password = document.getElementById('password').value;  var url = window.location.href.concat(password);  window.location = url;  console.log(url);  })  </script>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | passwordchanged.html |
|  | Create a python script passwordchanged.py with the code below  sudo nano ~/shoecabinet/templates/ passwordchanged.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Password Changed  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h2>Password Changed</h2>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | changeconfidence.html |
|  | Create a python script changeconfidence.py with the code below  sudo nano ~/shoecabinet/templates/ changeconfidence.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Change Face Unlock Confidence  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h3>Change Face Unlock Confidence: </h3>  <p>Please Enter a Value (0-100)</p>  <form id="routeform" onSubmit="route();">  <input type="number" name="number" id="number"/>  <input type="button" id="submit" value="Submit"/>  </form>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  </body>  {% endblock %}  {% block scripts %}  <script>  var submit = document.getElementById('submit');  submit.addEventListener('click', function() {  console.log("clicked")  var number = document.getElementById('number').value;  var url = window.location.href.concat(number);  window.location = url;  console.log(url);  })  </script>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | confidencechanged.html |
|  | Create a python script confidencechanged.py with the code below  sudo nano ~/shoecabinet/templates/ confidencechanged.html |
|  | { {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Password Changed  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h2>Confidence Changed</h2>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | facescan.html |
|  | Create a python script facescan.py with the code below  sudo nano ~/shoecabinet/templates/ facescan.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Face Scan  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <form id="routeform" onSubmit="route();">  User ID:  <br>  <input type="number" name="id" id="id"/>  <br>  Username:  <br>  <input type="text" name="username" id="username"/>  <br>  <input type="button" style="margin-top: 2%;" id="submit" value="Submit"/>  </form>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  {% block scripts %}  <script>  var submit = document.getElementById('submit');  submit.addEventListener('click', function() {  //console.log("clicked")  var userid = document.getElementById('id').value;  var username = document.getElementById('username').value;  var url = window.location.href.concat(userid).concat('/').concat(username);  window.location = url;  console.log(url);  })  </script>  {% endblock %}  </html> |

|  |  |
| --- | --- |
|  | faceregistered.html |
|  | Create a python script faceregistered.py with the code below  sudo nano ~/shoecabinet/templates/ faceregistered.html |
|  | {% extends "bootstrap/base.html" %}  <!DOCTYPE html>  {% block title %}  Password Changed  {% endblock %}  {% block content %}  <h1 style="margin: 0 auto; text-align: center; background-color: #3E50B4; color: white; padding: 3%; box-shadow: 2px 2px 10px #888888;">  KremePi Dash Outdoor Security System  </h1>  <div style="margin: 0 auto; text-align: center; width: 50%; background-color: #FFFFF0; color: black; padding: 3%; padding-top: 1%; box-shadow: 2px 2px 10px #888888; margin-top: 3%;">  <h2>Face Registered</h2>  <h5>  <a href="/">Go Back</a>  </h5>  </div>  {% endblock %}  </html> |

# Section 11 Testing

Test the finished Programs

When both RPI are set-up properly, we can proceed to testing the system. The testing should be followed along with the video that we have created. The video can be found here: <https://youtu.be/Z7kAxq_26Ik>

**-- End of CA2 Step-by-step tutorial --**