SCHOOL OF DIGITAL MEDIA AND INFOCOMM TECHNOLOGY (DMIT)

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)

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

Table of Contents

Section 1
Overview of project

A. Where we have uploaded our tutorial

https://github.com/joshualeejunyi/KremePi

B. 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.

C. 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.

D. 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.

E. Summary of the steps that will be described

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

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

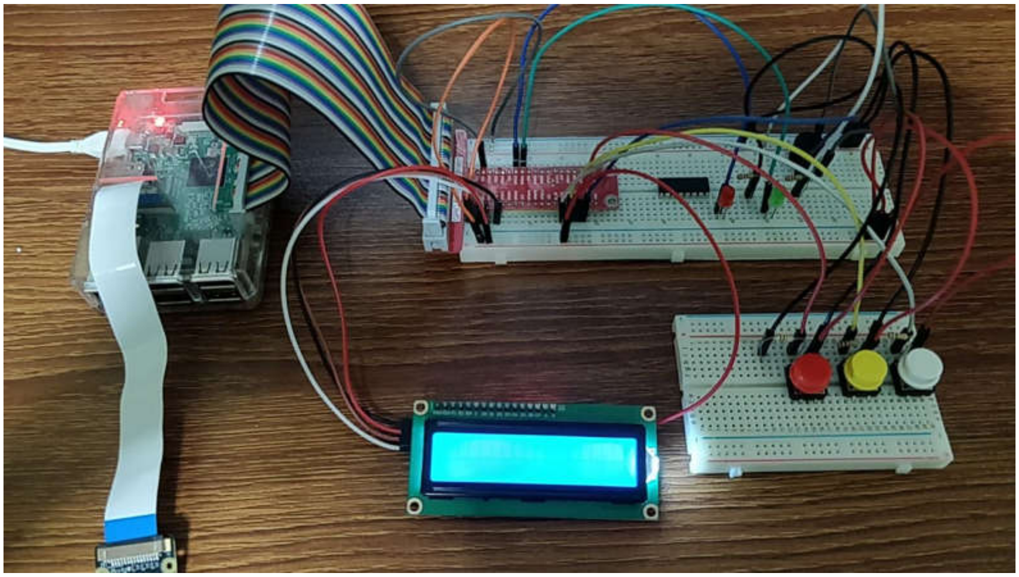


Figure 1: Final product of RPI(doorbell)

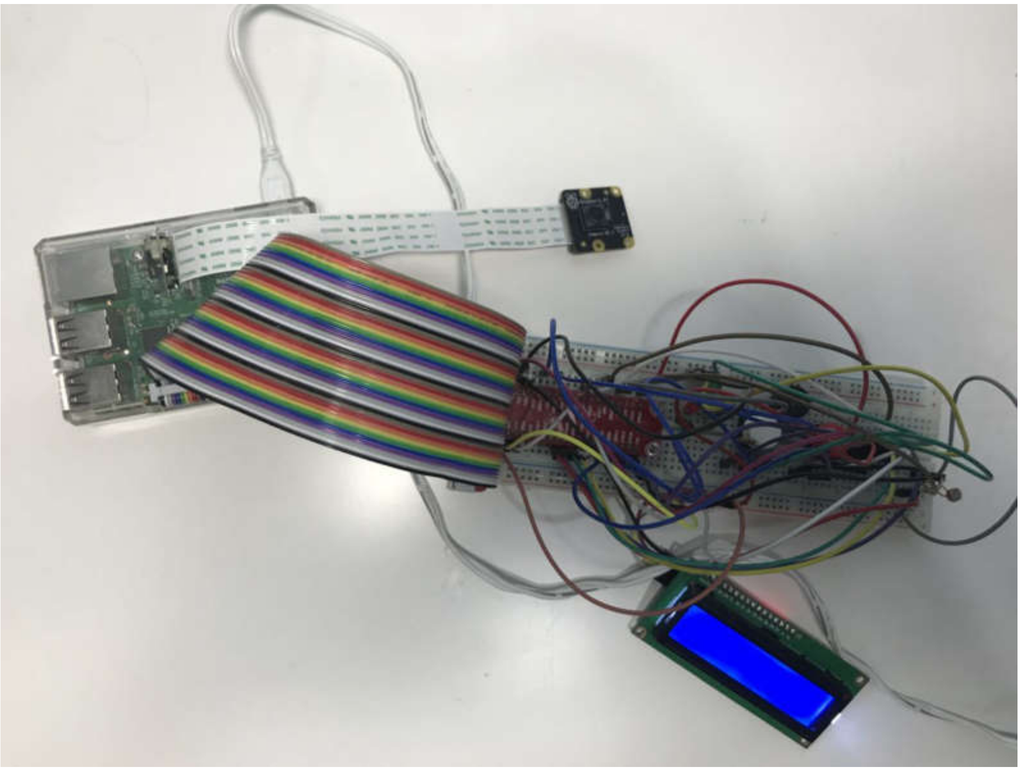
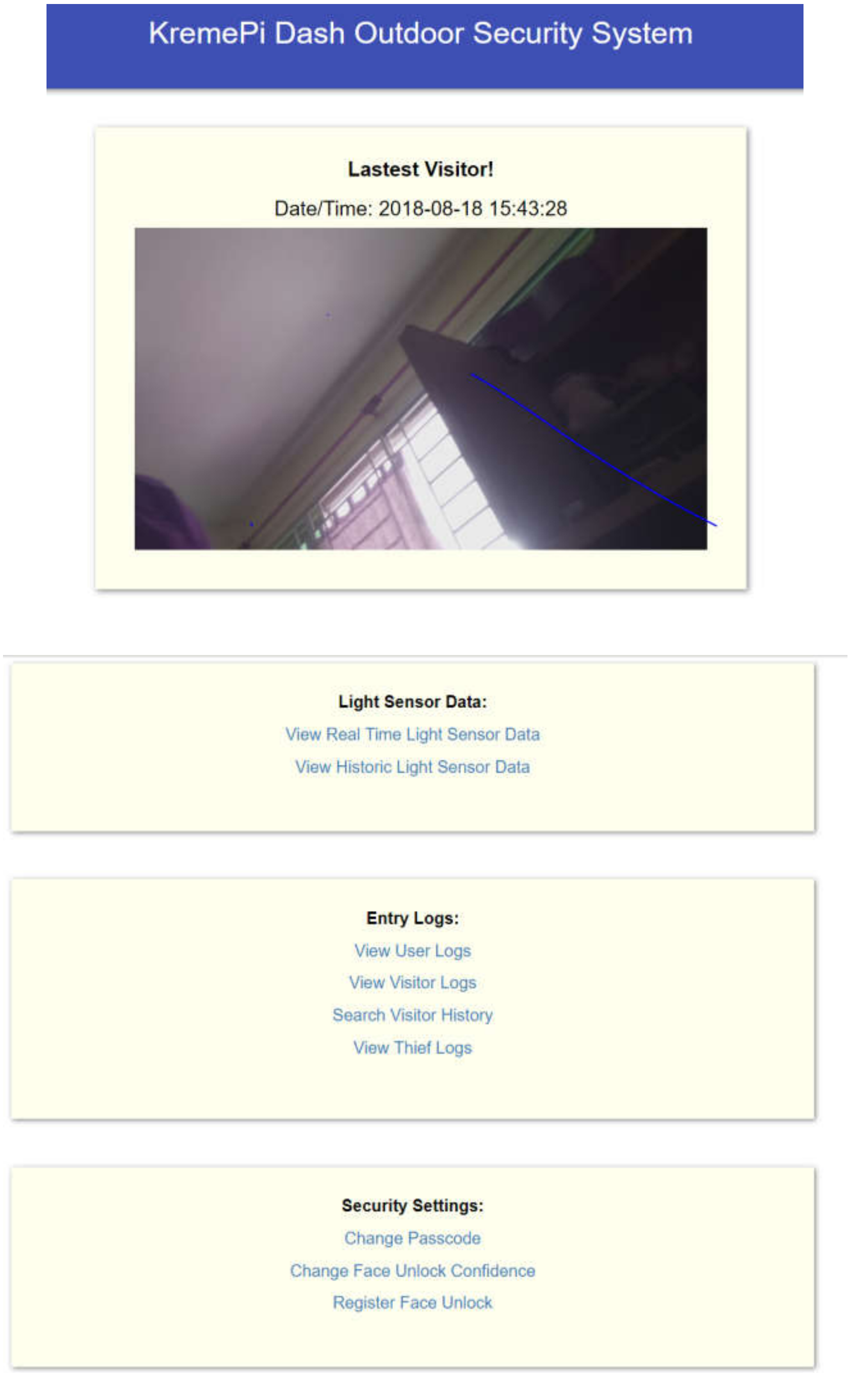


Figure 2: Final product of RPI(shoe cabinet)

G. How does the web or mobile application look like?



KremePi Dash Outdoor Security System

Lastest Visitor!

Date/Time: 2018-08-18 15:43:28

Light Sensor Data:

View Real Time Light Sensor Data

View Historic Light Sensor Data

Entry Logs:

View User Logs

View Visitor Logs

Search Visitor History

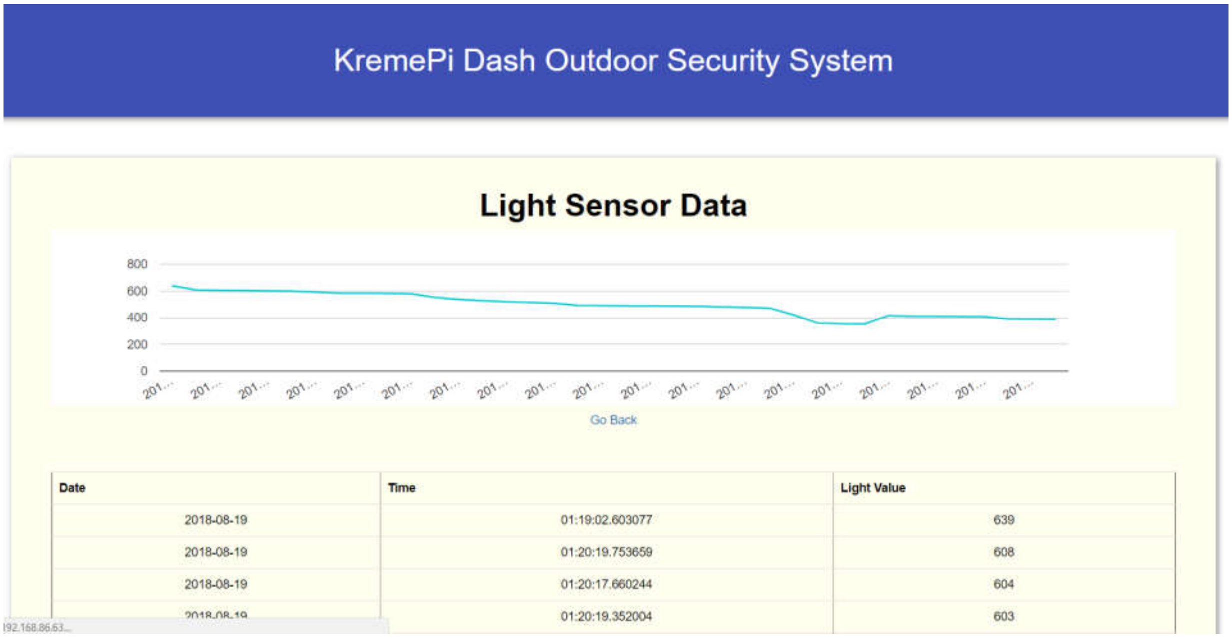
View Thief Logs

Security Settings:

Change Passcode

Change Face Unlock Confidence

Register Face Unlock



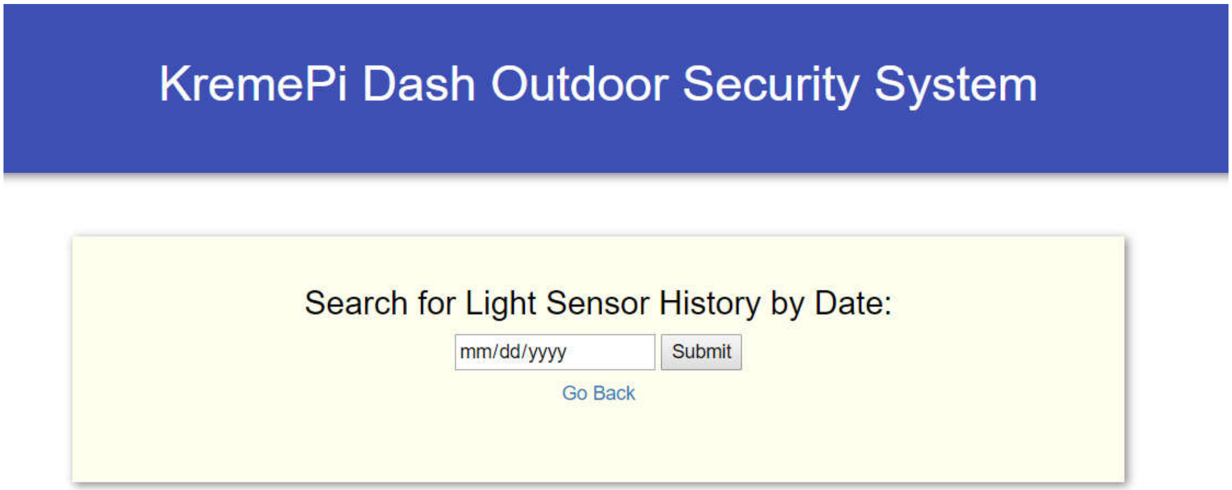
KremePi Dash Outdoor Security System

Light Sensor Data

Go Back

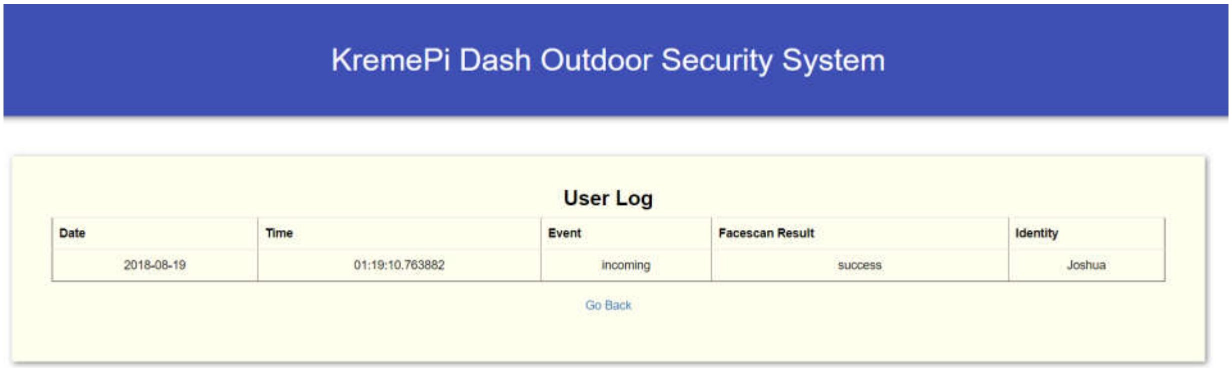
Date	Time	Light Value
2018-08-19	01:19:02.603077	639
2018-08-19	01:20:19.753659	608
2018-08-19	01:20:17.660244	604
2018-08-19	01:20:19.352004	603

192.168.86.63...



KremePi Dash Outdoor Security System

Search for Light Sensor History by Date:

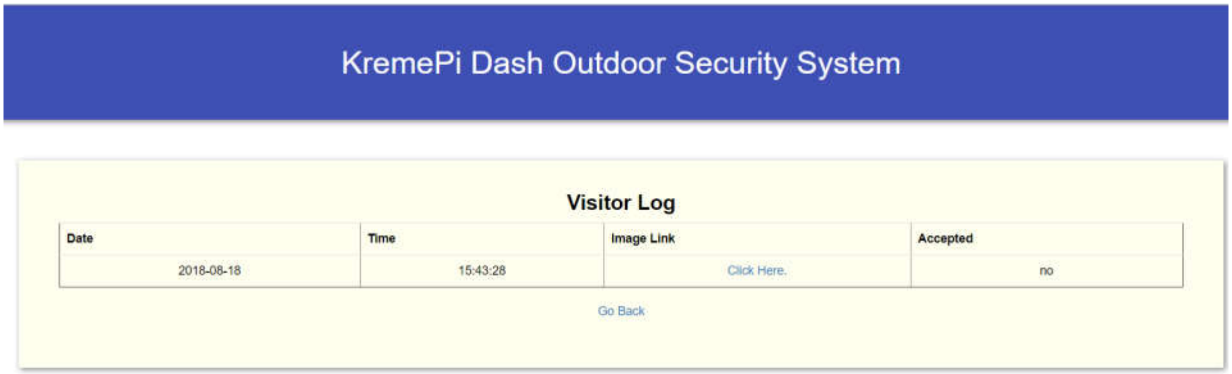


KremePi Dash Outdoor Security System

User Log

Date	Time	Event	Facescan Result	Identity
2018-08-19	01:19:10.763882	incoming	success	Joshua

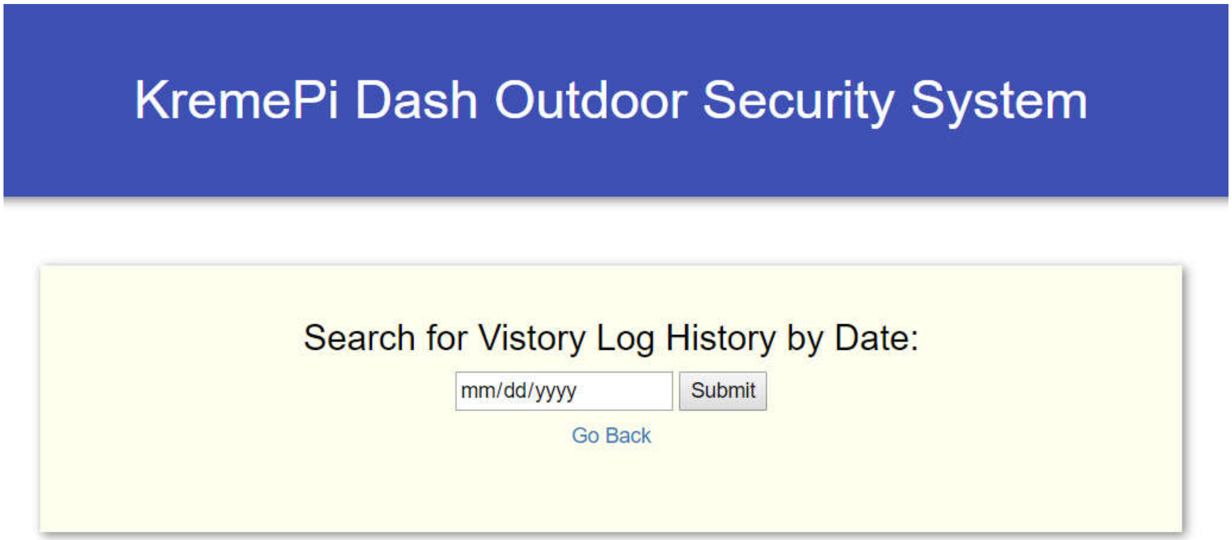
Go Back



KremePi Dash Outdoor Security System

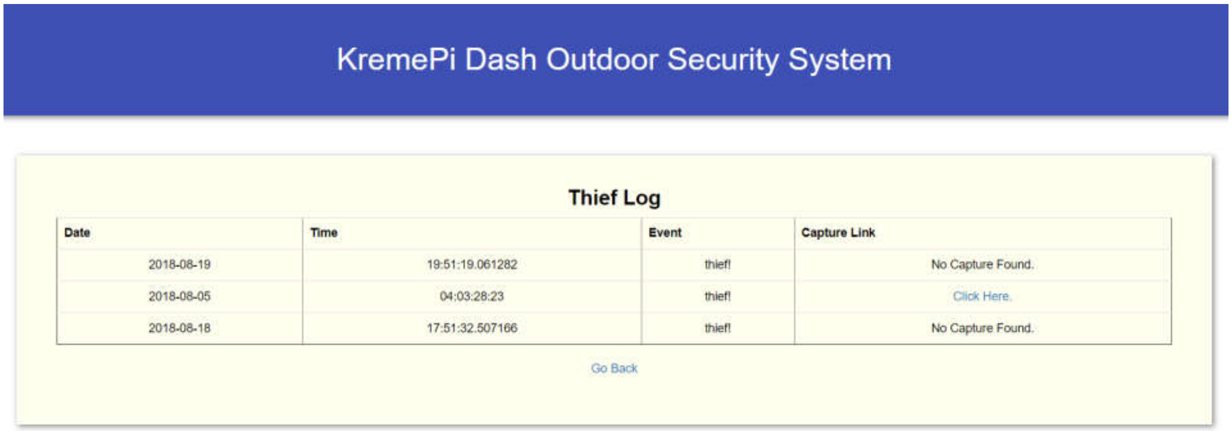
Visitor Log

Date	Time	Image Link	Accepted
2018-08-18	15:43:28	Click Here.	no



KremePi Dash Outdoor Security System

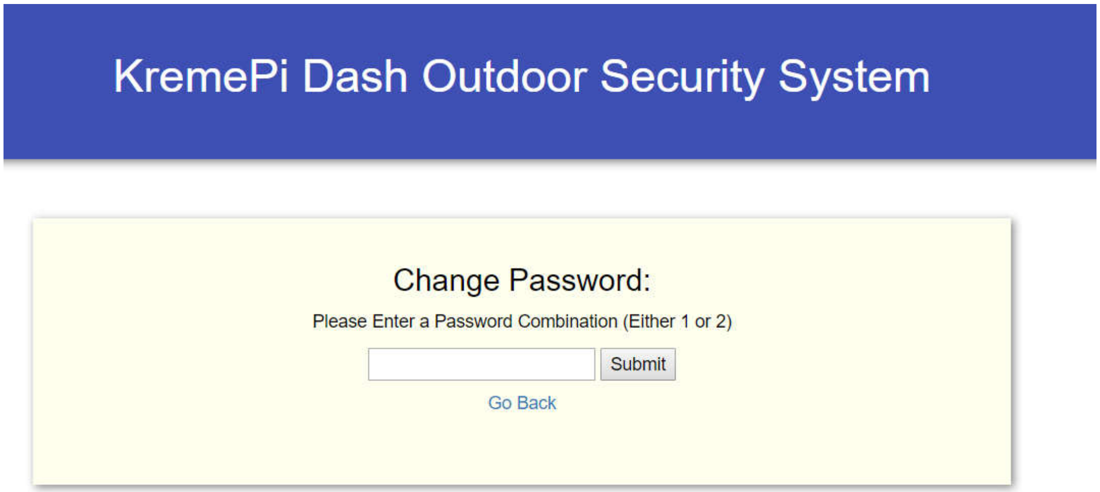
Search for Vistory Log History by Date:



KremePi Dash Outdoor Security System

Thief Log

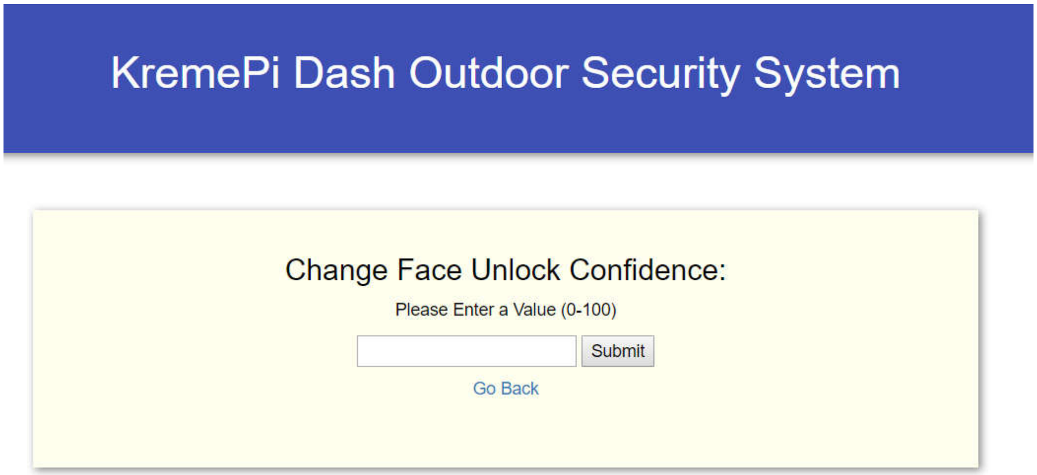
Date	Time	Event	Capture Link
2018-08-19	19:51:19.061282	thief!	No Capture Found.
2018-08-05	04:03:28:23	thief!	Click Here.
2018-08-18	17:51:32.507166	thief!	No Capture Found.



KremePi Dash Outdoor Security System

Change Password:

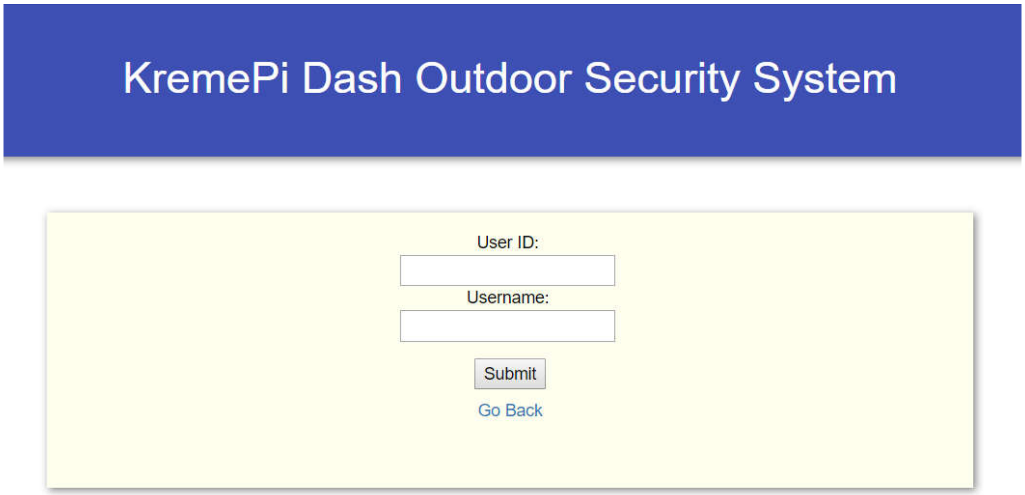
Please Enter a Password Combination (Either 1 or 2)



KremePi Dash Outdoor Security System

Change Face Unlock Confidence:

Please Enter a Value (0-100)



KremePi Dash Outdoor Security System

User ID:

Username:

Section 2
Hardware requirements

Hardware checklist

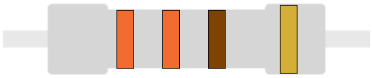
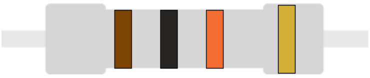
Button

	Task	
a)	- 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	
a)	- 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	
a)	- 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*
b)	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

a)
- 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

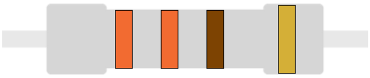


Figure 4: Orange,Orange,Brown

Buzzer

Task

a) 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

a)

- 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

a)

- 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!

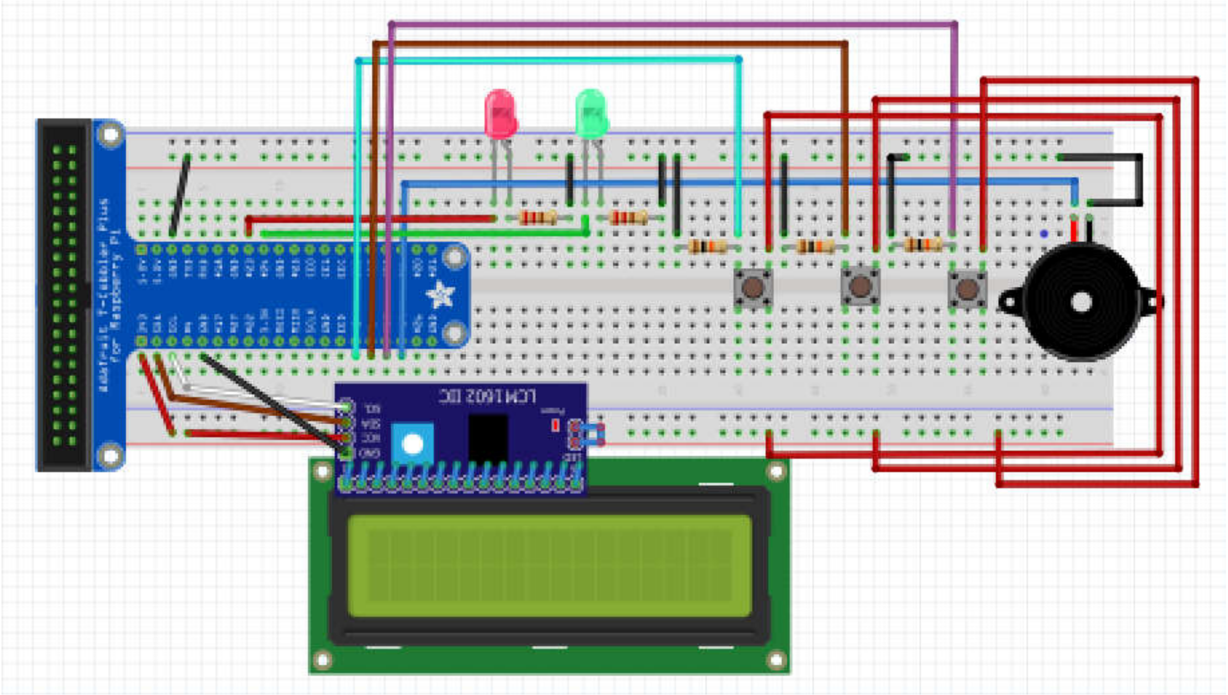
i2c LCD Screen

Section 3
Hardware Set-up

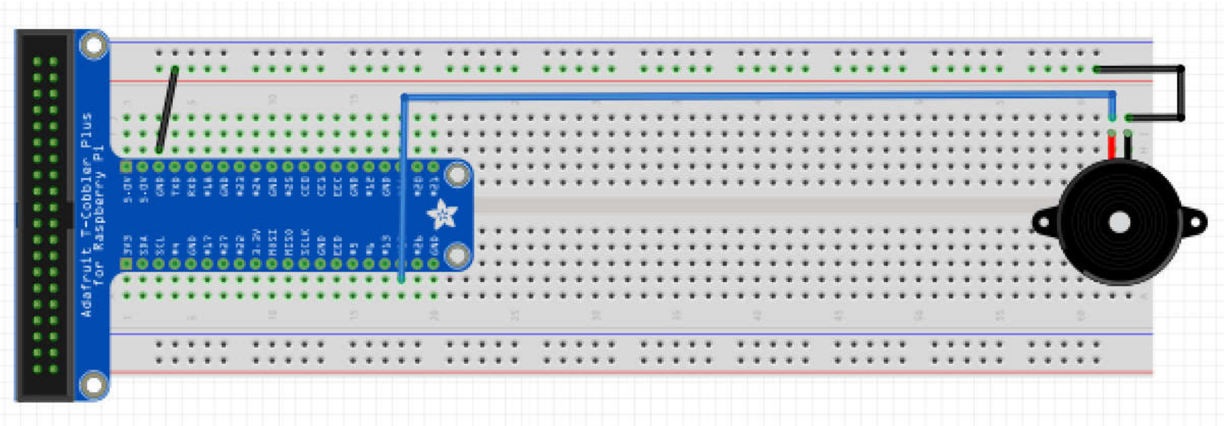
Set-up the Hardware Required

a) Gate Security RPI

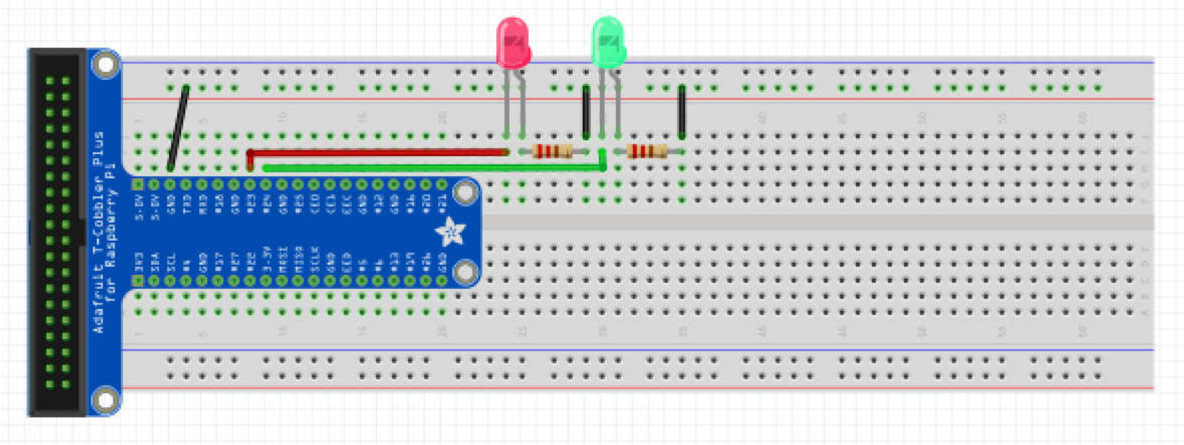
Completed Fritzing Diagram



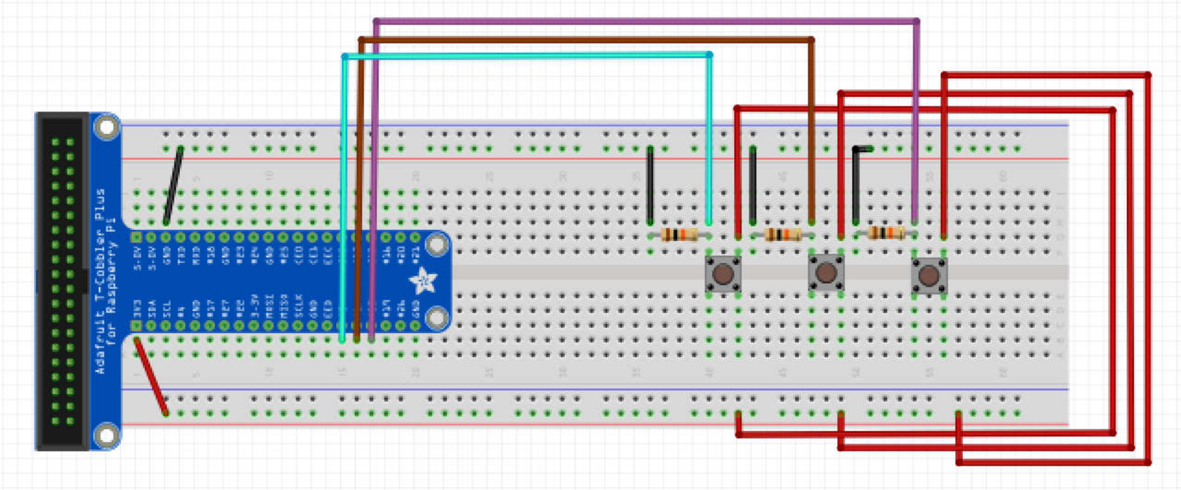
Connect the Buzzer



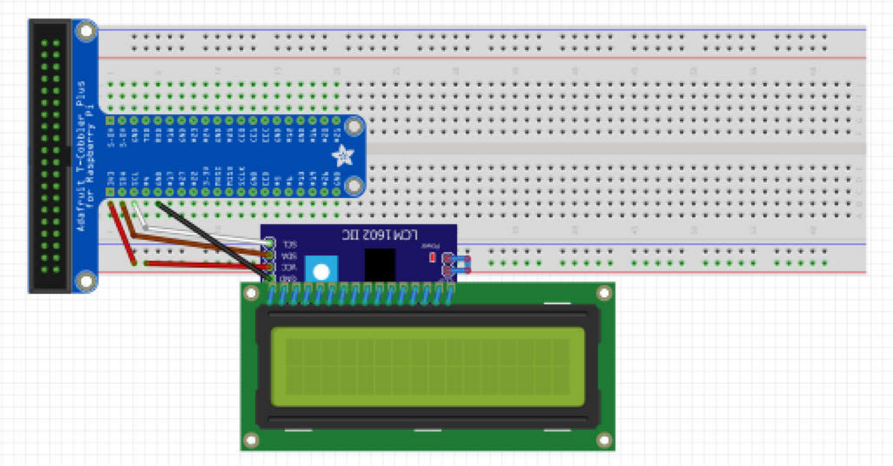
Connect the LEDs



Connect the Buttons

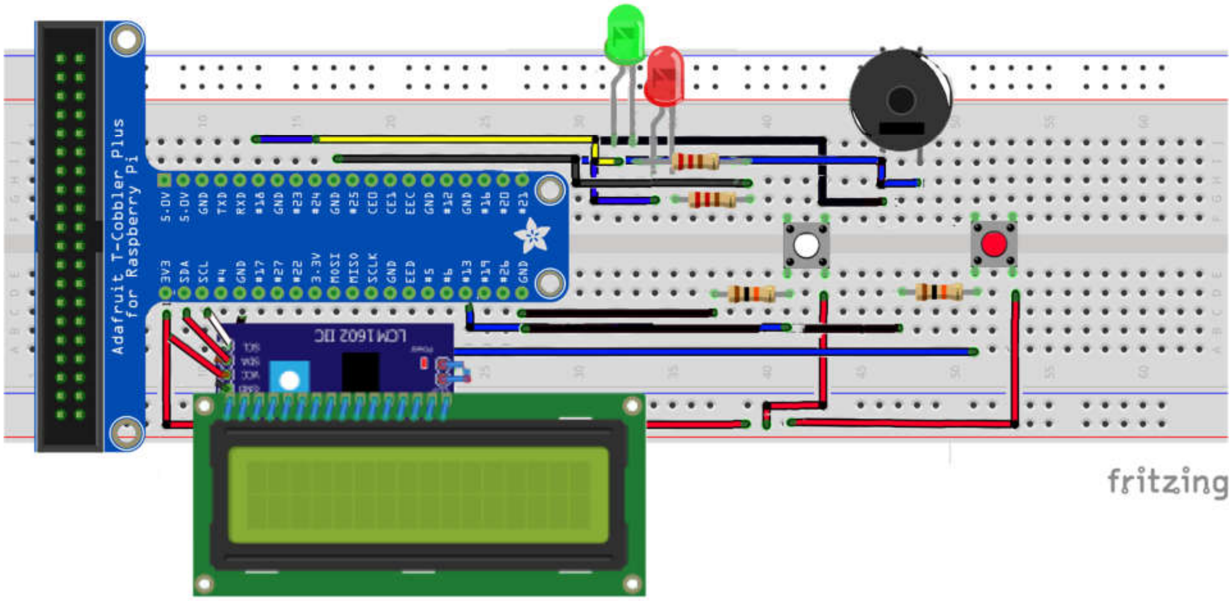


Connect the LCD

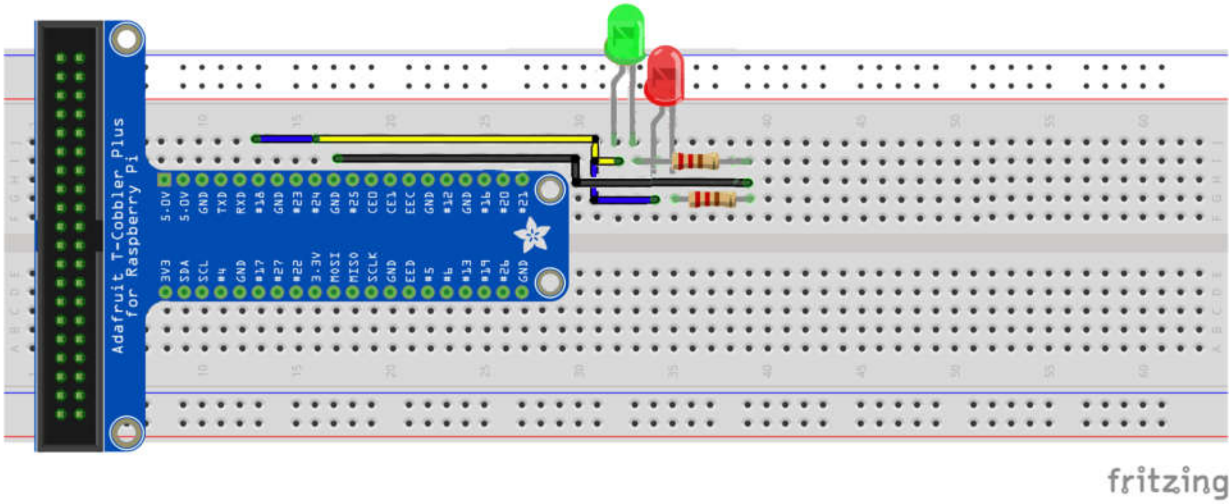


b)Shoe Cabinet RPI

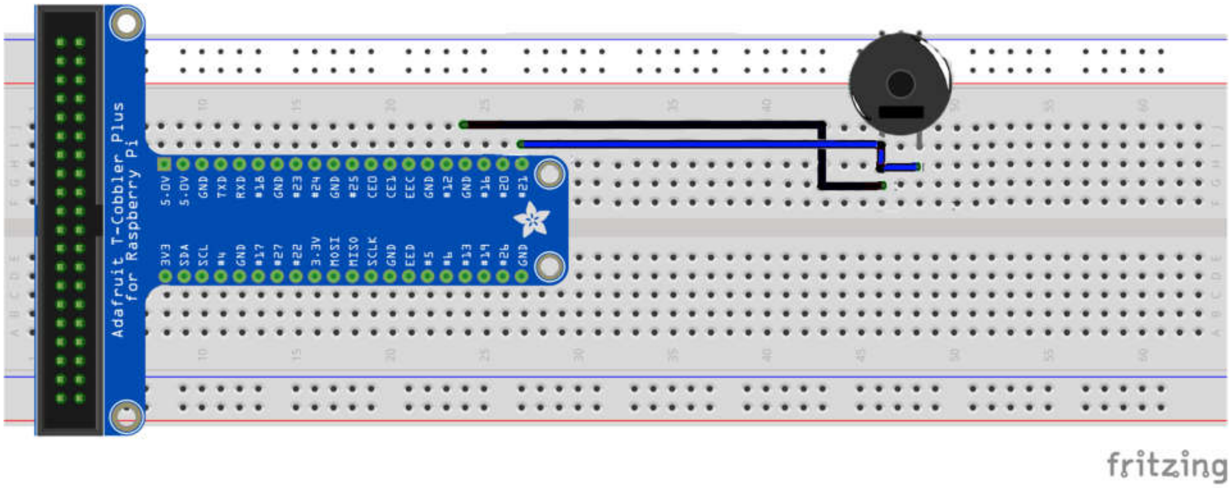
Completed Fritzing Diagram


Adafruit T-Cobbler Plus
for Raspberry Pi
5.0V 5.0V GND TXD RXD #18 GND #23 #24 GND #25 CE0 CE1 EEC GND #12 GND #16 #20 #21
3V3 SDA SCL #4 GND #17 #27 #22 3.3V MOSI MISO SCLK GND EED #5 #6 #13 #19 #26 GND
LCM1602 IIC
SCL
SDA
VCC
GND
fritzing

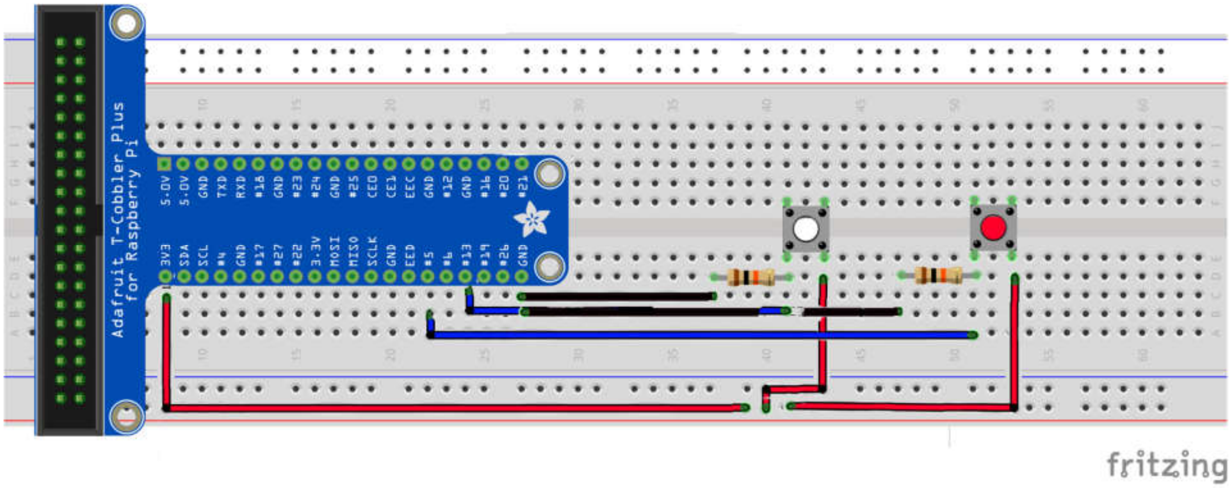
Connect the LEDs


Adafruit T-Cobbler Plus
for Raspberry Pi
5.0V 5.0V GND TXD RXD #18 GND #23 #24 GND #25 CE0 CE1 EEC GND #12 GND #16 #20 #21
3V3 SDA SCL #4 GND #17 #27 #22 3.3V MOSI MISO SCLK GND EED #5 #6 #13 #19 #26 GND
fritzing

Connect the Buzzer



Connect the Buttons



Section 4
Appplication Prerequisites

All Packages and Directories Required

Doorbell RPI Task(s)

a) Make a directory and save ALL files in the directory

```
mkdir ~/doorbell/
```

b) Make directories for application to save images

```
Mkdir -p /home/pi/Desktop/temp /home/pi/Desktop/thieftemp
```

c) Install Botocore and Boto3 (https://github.com/boto/boto)

```
sudo pip install boto3
sudo pip install botocore
```

d) Install awscli

```
sudo pip install awscli
```

e) Install the Telegram API on your Raspberry Pi (http://telepot.readthedocs.io/en/latest/)

```
sudo pip install telepot
```

f) Install threading

```
sudo pip install threading
```

Shoe Cabinet RPI Task(s)

a) Make a directory and save ALL files in the directory

```
mkdir ~/shoecabinet/
```

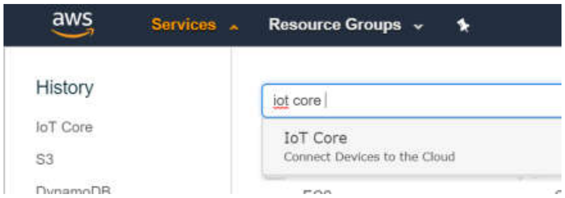
Section 5
AWS IoTcore (MQTT)

Set-up Amazon Web Service IoTcore service

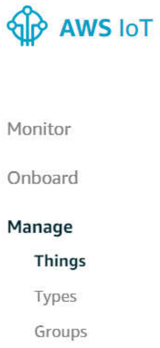
Create a Thing in AWS

a) Login to aws using your credentials.

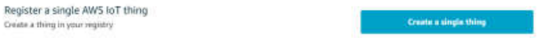
b) Click on Services and search for IoTcore



c) Open the navigations on the left. Click on Manage, and then Things.



d) Click on "Create" on the top right corner. Then, click on Create a single thing

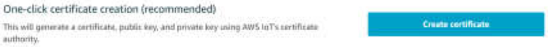


Register a single AWS IoT thing

Create a thing in your registry

e) In the field for name, enter whichever name you desire. Leave the other fields as default. Then click "Next"

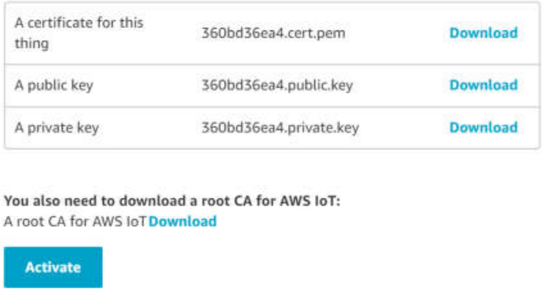
f) Proceed to create a certificate using the "One-Click Certificate Creation"



One-click certificate creation (recommended)

This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

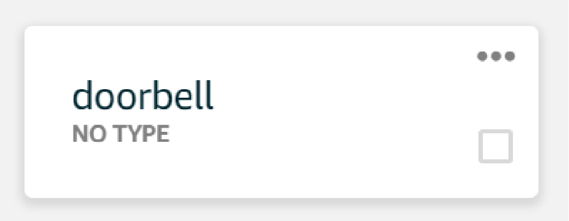
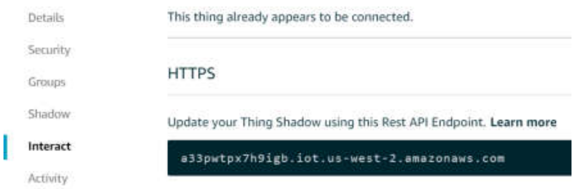
g) Download the certificates, as well as the rootCA certificate and store them in the Project directory. After which, click on "Activate".



A certificate for this thing	360bd36ea4.cert.pem	Download
A public key	360bd36ea4.public.key	Download
A private key	360bd36ea4.private.key	Download

You also need to download a root CA for AWS IoT:

A root CA for AWS IoT **Download**

h) When all is completed, click on “done” to return to the thing selection page. Then, click on the newly created thing.	
i) On the left hand side, click on interact, then copy the REST API endpoint onto a notepad for use later.	 Details Security Groups Shadow Interact Activity HTTPS Learn more This thing already appears to be connected. Update your Thing Shadow using this Rest API Endpoint. `a33pwtpx7h9igb.iot.us-west-2.amazonaws.com`

Section 6
AWS DynamoDB

Set-up Amazon Web Service NoSQL DynamoDB service

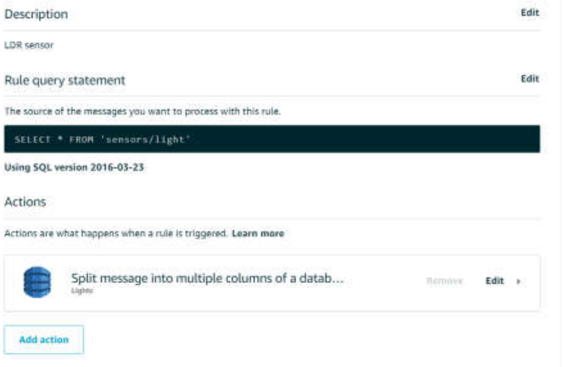
Set up and store data in the light database

a) Login to aws using your credentials.

b) Open the AWS IOT console, click on "ACT" and click "Create".

c) 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"



Description

LDR sensor

Rule query statement

The source of the messages you want to process with this rule.

```
SELECT * FROM 'sensors/light'
```

Using SQL version 2016-03-23

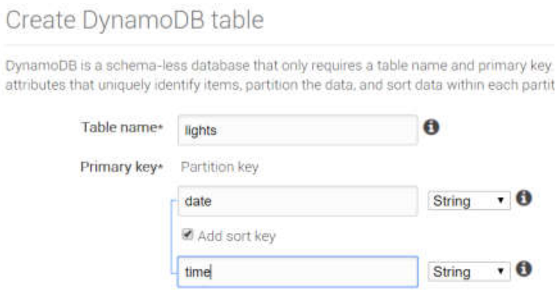
Actions

Actions are what happens when a rule is triggered. Learn more

Split message into multiple columns of a datab...
Lights

d) 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".



Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key attributes that uniquely identify items, partition the data, and sort data within each partit

Table name* lights

Primary key* Partition key

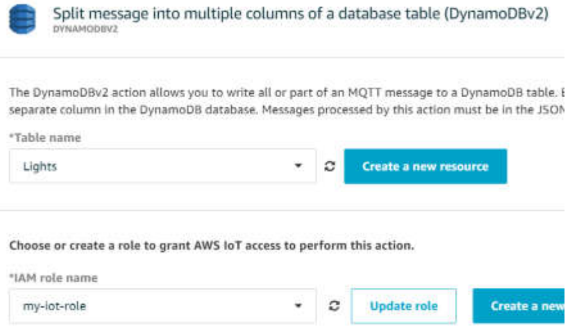
date String

☑ Add sort key

time String

e) 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".



Split message into multiple columns of a database table (DynamoDBv2)
DYNAMODBV2

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. E
separate column in the DynamoDB database. Messages processed by this action must be in the JSON

*Table name

Lights

Choose or create a role to grant AWS IoT access to perform this action.

*IAM role name

my-iot-role

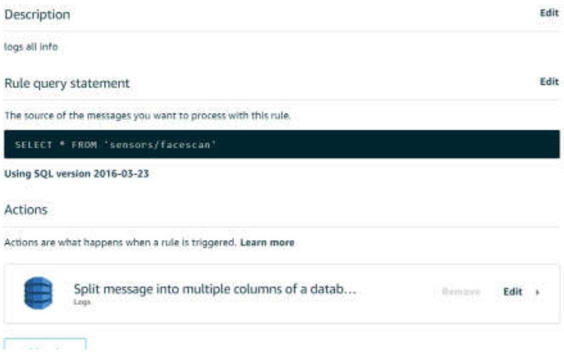
Set up and store data in the logs database

a) Login to aws using your credentials.

b) Open the AWS IOT console, click on "ACT" and click "Create".

c) 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"



Description

logs all info

Rule query statement

The source of the messages you want to process with this rule.

```
SELECT * FROM 'sensors/facescan'
```

Using SQL version 2016-03-23

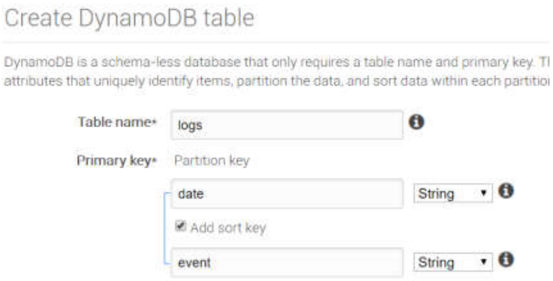
Actions

Actions are what happens when a rule is triggered. Learn more

Split message into multiple columns of a datab...
Logs

d) 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".



Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. Tl
attributes that uniquely identify items, partition the data, and sort data within each partitio

Table name* logs

Primary key* Partition key

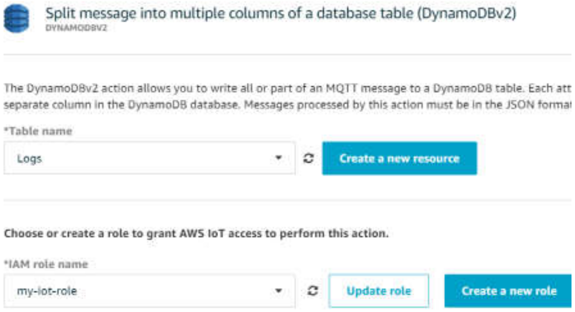
date String

☑ Add sort key

event String

e) 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".



Split message into multiple columns of a database table (DynamoDBv2)
DYNAMODBV2

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each att
separate column in the DynamoDB database. Messages processed by this action must be in the JSON format

*Table name

Logs

Choose or create a role to grant AWS IoT access to perform this action.

*IAM role name

my-iot-role

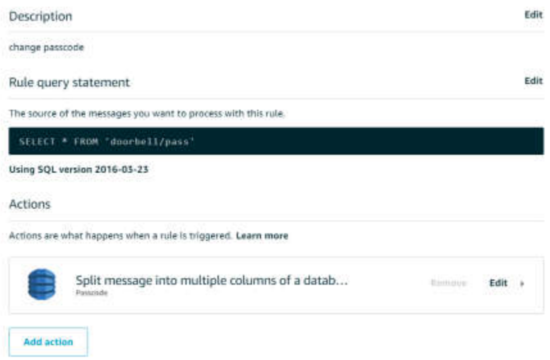
Set up and store data in the passcode database

a) Login to aws using your credentials.

b) Open the AWS IOT console, click on "ACT" and click "Create".

c) 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"



Description

change passcode

Rule query statement

The source of the messages you want to process with this rule.

```
SELECT * FROM 'doorbell/pass'
```

Using SQL version 2016-03-23

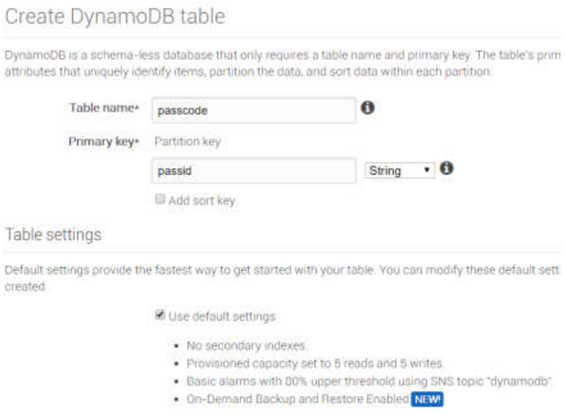
Actions

Actions are what happens when a rule is triggered. **Learn more**

Split message into multiple columns of a datab...
Passcode

d) 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".



Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's prim attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* passcode

Primary key* Partition key

passid String

□ Add sort key

Table settings

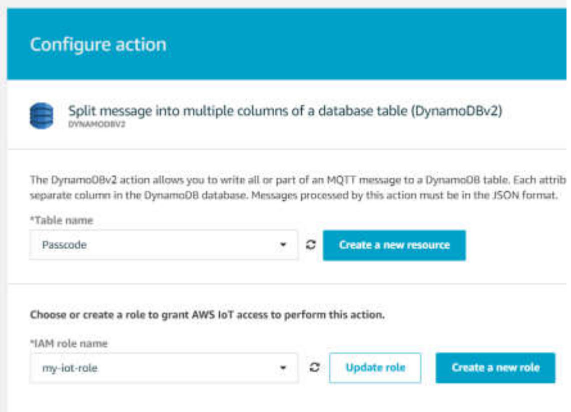
Default settings provide the fastest way to get started with your table. You can modify these default sett created

☑ Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- On-Demand Backup and Restore Enabled NEW!

e) 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".



Configure action

Split message into multiple columns of a database table (DynamoDBv2)
DYNAMODBV2

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each attrib separate column in the DynamoDB database. Messages processed by this action must be in the JSON format.

*Table name

Passcode

Choose or create a role to grant AWS IoT access to perform this action.

*IAM role name

my-iot-role

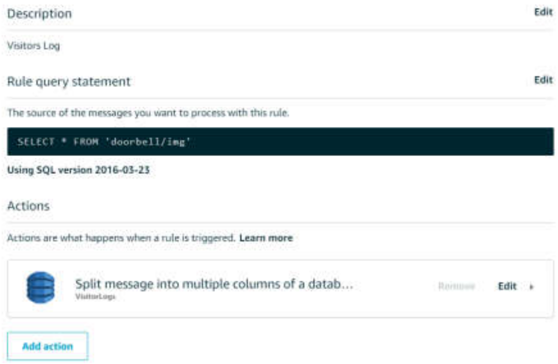
Set up and store data in the visitors log database

a) Login to aws using your credentials.

b) Open the AWS IOT console, click on "ACT" and click "Create".

c) 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"



Description

Visitors Log

Rule query statement

The source of the messages you want to process with this rule.

```
SELECT * FROM 'doorbell/img'
```

Using SQL version 2016-03-23

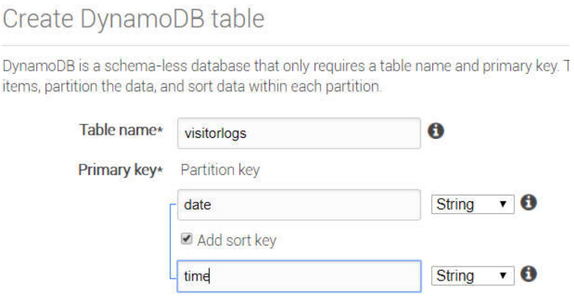
Actions

Actions are what happens when a rule is triggered. Learn more

Split message into multiple columns of a datab...
VisitorLogs

d) 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".



Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. T
items, partition the data, and sort data within each partition.

Table name* visitorlogs

Primary key* Partition key

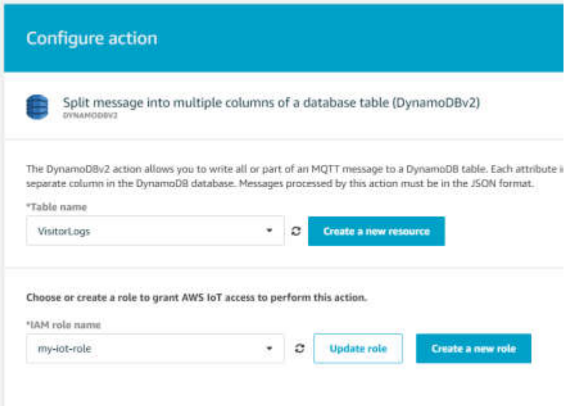
date String

☑ Add sort key

time String

e) 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".



Configure action

Split message into multiple columns of a database table (DynamoDBv2)
DYNAMODBV2

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each attribute i
separate column in the DynamoDB database. Messages processed by this action must be in the JSON format.

*Table name
VisitorLogs

Choose or create a role to grant AWS IoT access to perform this action.

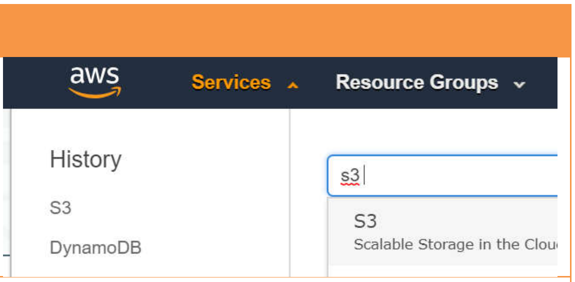
*IAM role name
my-iot-role

Section 7
AWS S3

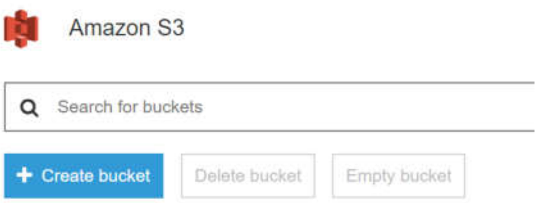
Set-up Amazon Web Service Image Repository S3

Set-up AWS S3

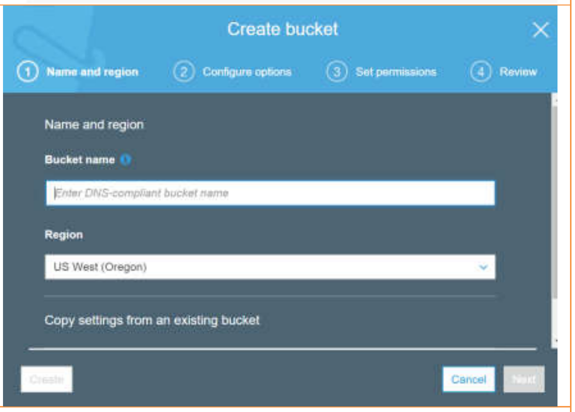
1) Under the "Services" navigation, search "s3" and navigate to the AWS S3



2) Next, click on "Create Bucket".



3) 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.



Name and region

Bucket name: Enter DNS-compliant bucket name

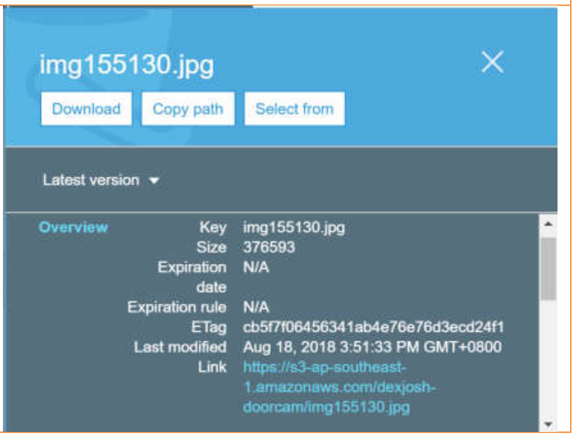
Region: US West (Oregon)

Copy settings from an existing bucket

4) Next, repeat from step b) to create another container. (Keep track of both the container names as we will be using it later)

5) 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/

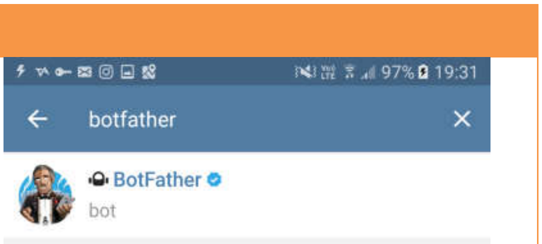
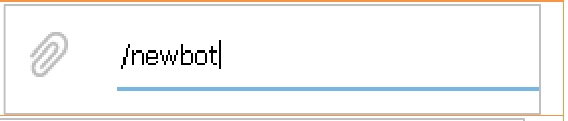
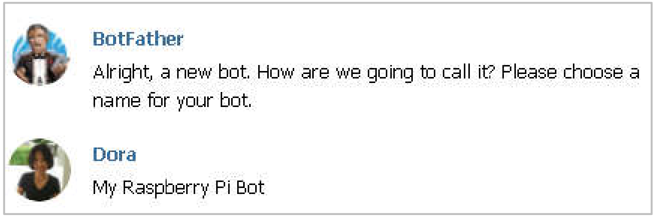
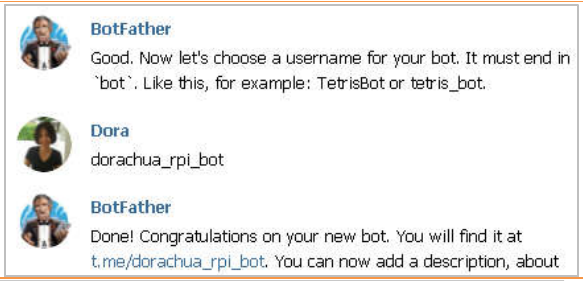
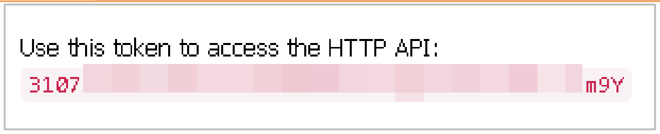
Repeat this step for both buckets to get the bucket links that will be used later.

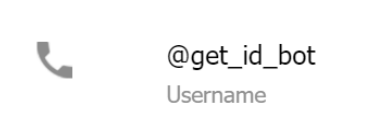
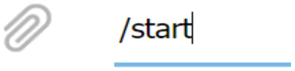
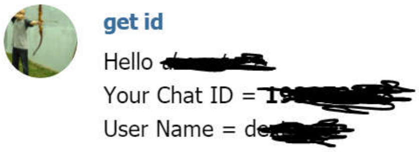


Key img155130.jpg
Size 376593
Expiration date N/A
Expiration rule N/A
ETag cb5f7f06456341ab4e76e76d3ecd24f1
Last modified Aug 18, 2018 3:51:33 PM GMT+0800
Link https://s3-ap-southeast-1.amazonaws.com/dexjosh-doorcam/img155130.jpg

Section 8
Telegram Bot

Set-up Telegram Bot

	Creating a Telegram Bot	
a)	Open Telegram app in your laptop or mobile and start "BotFather"	 97% 19:31 botfather BotFather bot
b)	Type /newbot to create a new bot	 /newbot
c)	Give a name to your bot that describes its purpose.	 BotFather Alright, a new bot. How are we going to call it? Please choose a name for your bot. Dora My Raspberry Pi Bot
d)	Give a username to your bot. Make sure it ends with _bot	 BotFather Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot. Dora dorachua_rpi_bot BotFather Done! Congratulations on your new bot. You will find it at t.me/dorachua_rpi_bot. You can now add a description, about
e)	Copy down the access token that is issued by BotFather	 Use this token to access the HTTP API: 3107[redacted]m9Y

	Get Own Chat ID	
1)	Open Telegram app in your laptop or mobile and search for @get_id_bot	 @get_id_bot Username
2)	Type /start to get your chat id	 /start
3)	Record down the chat id for later use.	 get id Hello [redacted] Your Chat ID = [redacted] User Name = [redacted]

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.

```
> sudo raspi-config
```

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.

```
> init 6
```

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.

```
> sudo apt-get update && sudo apt-get upgrade
> sudo apt-get install build-essential cmake pkg-config
> sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev
> sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
> sudo apt-get install libxvidcore-dev libx264-dev
> sudo apt-get install libgtk2.0-dev libgtk-3-dev
> sudo apt-get install libatlas-base-dev gfortran
> sudo apt-get install python2.7-dev python3-dev
```

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

```
> mkdir /home/pi/assignment
> cd /home/pi/assignment
> wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.4.1.zip
> unzip opencv.zip
```

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.

```
> wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.4.1.zip
> unzip opencv_contrib.zip
```

We will also install pip, the Python package manager.

```
> wget https://bootstrap.pypa.io/get-pip.py
> sudo python get-pip.py
> sudo python3 get-pip.py
```

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.

```
> sudo pip install virtualenv virtualenvwrapper
> sudo rm -rf ~/.cache/pip
```

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

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
export VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3.6
# I had to add this in to work
```

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

```
> source ~/.profile
```

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

```
> mkvirtualenv cv -p python3
```

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

```
> source ~/.profile
> workon cv
```

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

```
(cv) > #this is the command line
```

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

```
(cv) > pip install numpy
```

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

```
(cv) > cd ~/opencv-3.3.0/
(cv) > mkdir build
(cv) > cd build
(cv) > cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D INSTALL_C_EXAMPLES=OFF\
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
    -D BUILD_EXAMPLES=ON ..
```

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:

```
(cv) > sudo /etc/init.d/dphys-swapfile stop
(cv) > sudo /etc/init.d/dphys-swapfile start
```

Now, we're ready to compile OpenCV.

```
(cv) > make -j4
```

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.

```
(cv) > sudo make install
(cv) > sudo ldconfig
```

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.

```
(cv) > cd /usr/local/lib/python3.5/site-packages/
(cv) > sudo mv cv2.cpython-xxxx.so cv2.so # adjust the cv2 filename accordingly
```

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

```
(cv) > cd ~/.virtualenvs/cv/lib/python3.6/site-packages/
(cv) > ln -s /usr/local/lib/python3.6/site-packages/cv2.so cv2.so
```

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.

```
(cv) > pip install Flask
(cv) > pip install gevent
(cv) > pip install smbus2
(cv) > pip install rpi-lcd
(cv) > pip install gpiozero
(cv) > pip install Adafruit_Python_DHT==1.1.2
(cv) > pip install mysqlclient
(cv) > pip install flask-bootstrap
```

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.

```
(cv) > mysql -u root -p dmitiot
```

After logging in, we shall create an assignment database.

```
mysql> create database assignment;
```

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

```
mysql> CREATE USER 'assignmentuser'@'localhost' IDENTIFIED by 'joshanddexpassword;
mysql> GRANT ALL PRIVILEGES ON assignment.* TO 'assignmentuser'@'localhost';
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

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

```
(cv) > mysql -u assignmentuser -p
mysql> USE assignment;
mysql> CREATE TABLE Users (UserID INT, Username VARCHAR(45), DateRegistered TIMESTAMP,
PRIMARY KEY (UserID));
mysql> CREATE TABLE Security (ID INT AUTO_INCREMENT, Passcode INT, FaceScanConfidence INT,
PRIMARY KEY(ID));
mysql.> INSERT INTO Security (FaceScanConfidence) VALUES (40);
```

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

a) Create a python script **server.py** with the code below

```
sudo nano ~/doorbell/doorbell.py
```

b)

```
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

a) Create a python script **shoecabinet.py** with the code below

```
sudo nano ~/shoecabinet/shoecabinet.py
```

b)

```
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

a) Create a python script **faceid.py** with the code below

```
sudo nano ~/shoecabinet/faceid.py
```

b)

```
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

a) Create a python script **trainer.py** with the code below

```
sudo nano ~/shoecabinet/trainer.py
```

b)

```
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

a) Create a python script **server.py** with the code below

```
sudo nano ~/shoecabinet/server.py
```

b)

```
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

a) Create a python script **index.html** with the code below

```
sudo nano ~/shoecabinet/templates/index.html
```

b)

```
{% 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"></s
cript>
   </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

a) Create a python script **lights.py** with the code below

```
sudo nano ~/shoecabinet/templates/lights.html
```

b)

```
{% 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

a) Create a python script **router.py** with the code below

```
sudo nano ~/shoecabinet/templates/router.html
```

b)

```
{% 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

a) Create a python script **userlog.py** with the code below

```
sudo nano ~/shoecabinet/templates/userlog.html
```

b)

```
{% 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

a) Create a python script **visitorlog.py** with the code below

```
sudo nano ~/shoecabinet/templates/visitorlog.html
```

b)

```
{% 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

a) Create a python script **visitorrouter.py** with the code below

```
sudo nano ~/shoecabinet/templates/ visitorrouter.html
```

b)

```
{% 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

a) Create a python script **thief.py** with the code below

```
sudo nano ~/shoecabinet/templates/ thief.html
```

b)

```
{% 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

a) Create a python script **changepassword.py** with the code below

```
sudo nano ~/shoecabinet/templates/ changepassword.html
```

b)

```
{% 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

a) Create a python script **passwordchanged.py** with the code below

```
sudo nano ~/shoecabinet/templates/ passwordchanged.html
```

b)

```
{% 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

a) Create a python script **changeconfidence.py** with the code below

```
sudo nano ~/shoecabinet/templates/ changeconfidence.html
```

b)

```
{% 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

a) Create a python script **confidencechanged.py** with the code below

```
sudo nano ~/shoecabinet/templates/ confidencechanged.html
```

b)

```
{ {% 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

a) Create a python script **facescan.py** with the code below

```
sudo nano ~/shoecabinet/templates/ facescan.html
```

b)

```
{% 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

a) Create a python script **faceregistered.py** with the code below

```
sudo nano ~/shoecabinet/templates/ faceregistered.html
```

b)

```
{% 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 --