SCHOOL OF COMPUTING (SOC)

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:

Prepared for: Ms Dora Chua

Class: DISM /FT/3A/31

Submitted by:

Student ID	**Name**
1512345	Evander
1703221	Davis Zheng

Table of Contents

Section 1
Overview of project

A. Where we have uploaded our tutorial

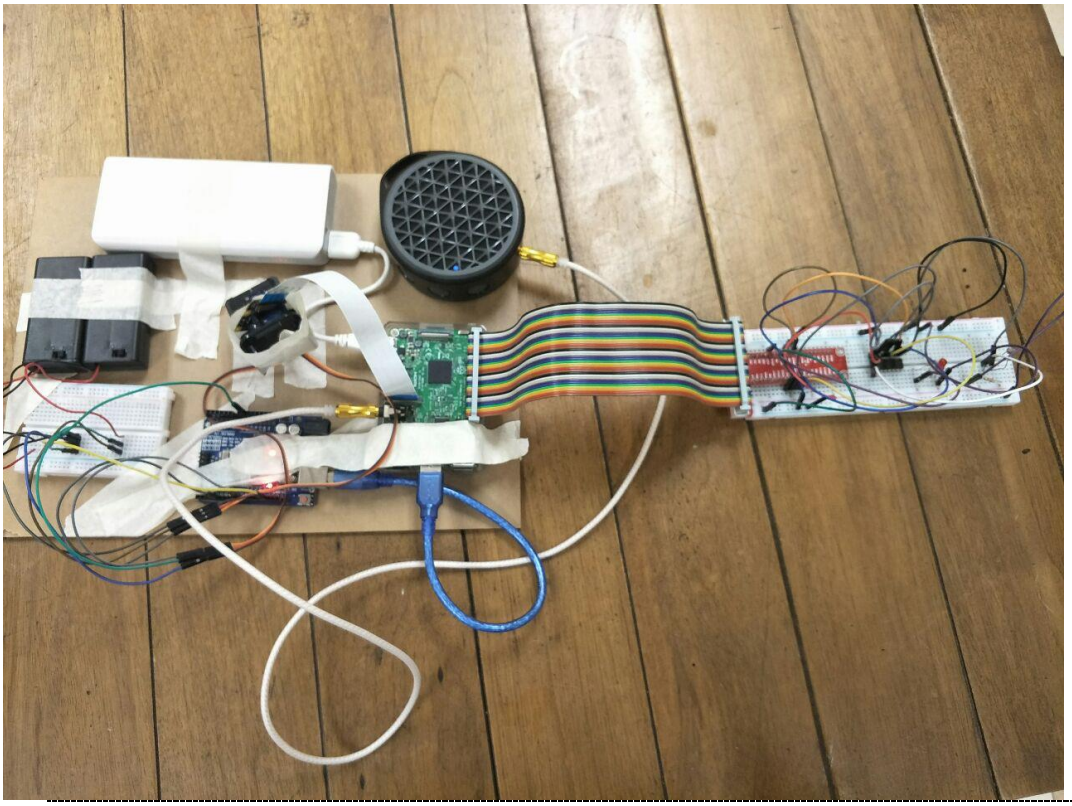
GitHub:https://github.com/evanderleng/IOTv2-Electric-Boogaloo
Youtube: https://youtu.be/j5pM6OR2QoY

B. What is the application about?

Our project, takes images using the camera every predefined seconds, feeds it into a cloud bucket and performs image recognition on it. It has been preconfigured to identify faces and highlight them. Currently it is performing the function of security camera, thus when it recognises a person, it will email the owner and issue a verbal warning to the visitor.It has been attached to servos to allow movement of the camera however this application is infinitely dynamic, allowing it to be used for a variety of other purposes such as as being able to be configured for Optical character recognition and other artificial intelligence purposes.

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

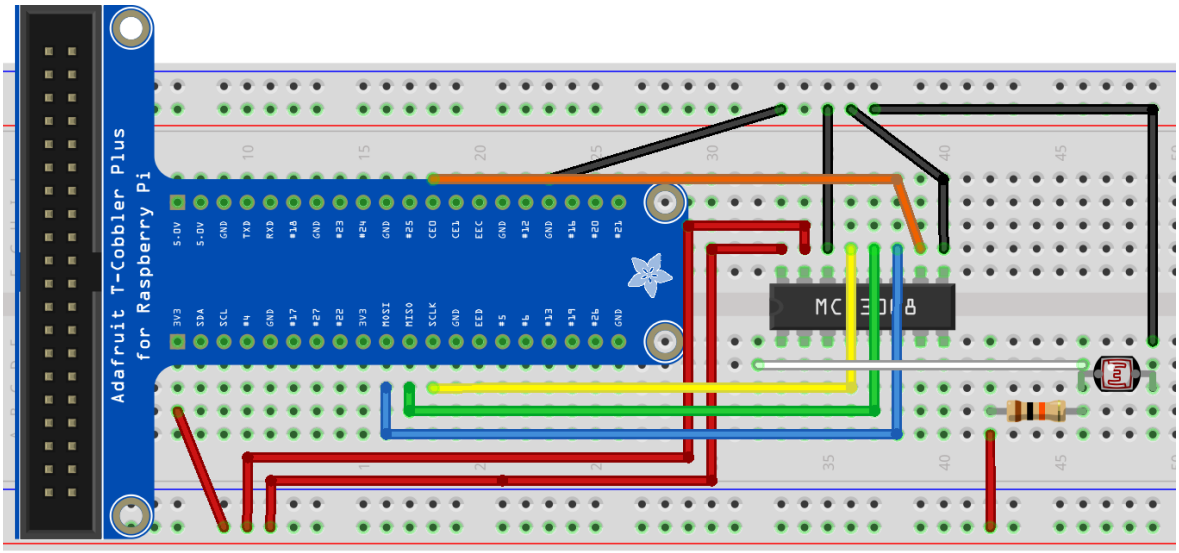
The following images shows how the completed circuit should look and a fritzing diagram for reference when setting up the hardware.



D. Connect the LDR

Task
a) Add the LDR to the breadboard as shown • One end of the LDR should be connected to **Pin 1** of the MCP3008 ADC and the 10k ohms **resistor** • The other end of the LDR should be connected to **GND**

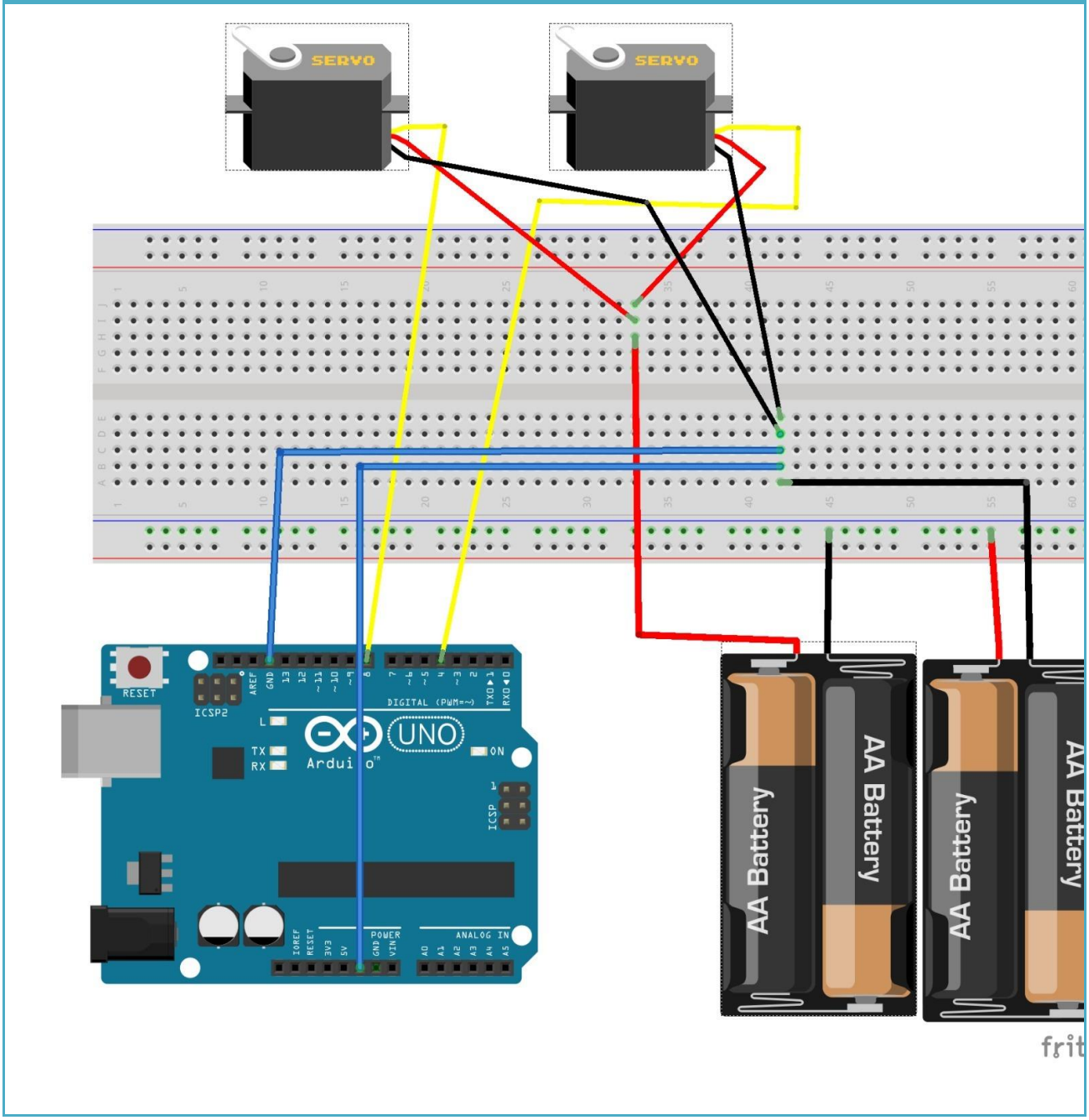
Task



Task

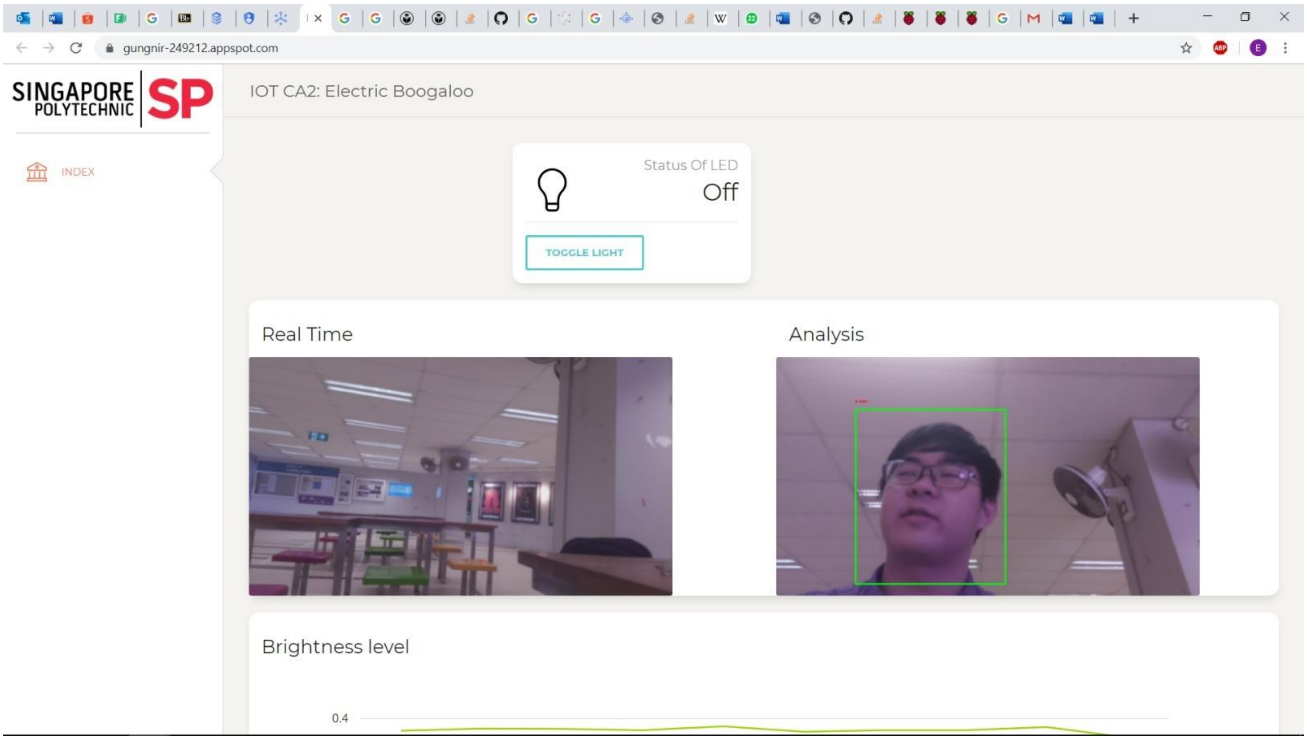
b) Follow the wiring display down below to properly connect the servos to the arduino

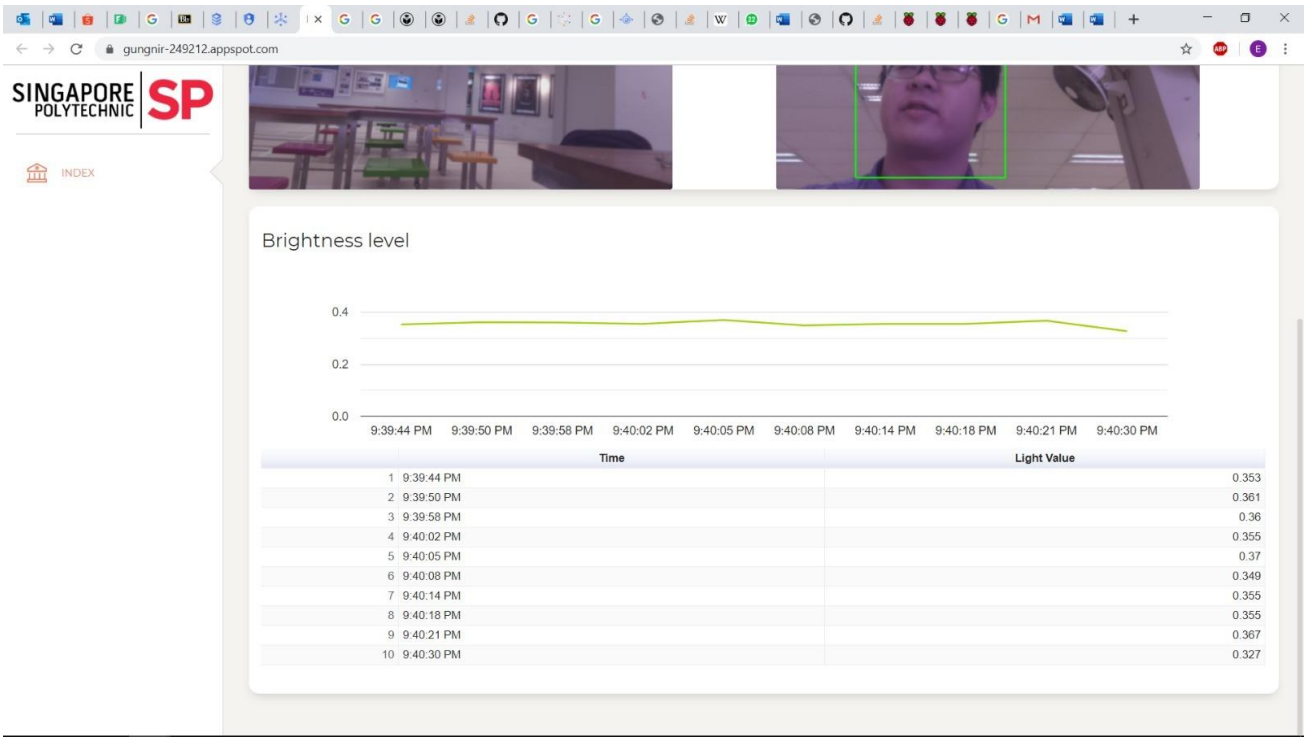
Task



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

The following images are screenshots of how the web application should look if setup was done correctly. All the screenshots are of index.html.

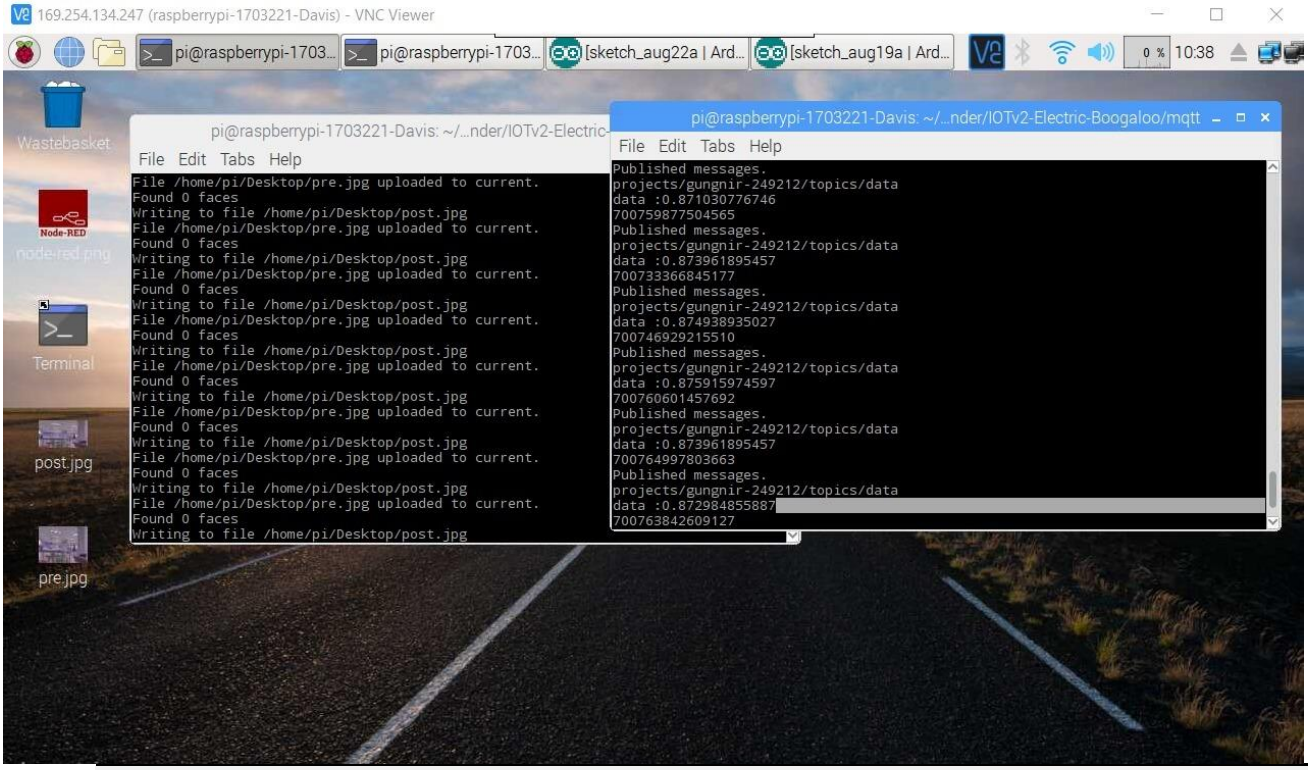




	Time	Light Value
1	9:39:44 PM	0.353
2	9:39:50 PM	0.361
3	9:39:58 PM	0.36
4	9:40:02 PM	0.355
5	9:40:05 PM	0.37
6	9:40:08 PM	0.349
7	9:40:14 PM	0.355
8	9:40:18 PM	0.355
9	9:40:21 PM	0.367
10	9:40:30 PM	0.327

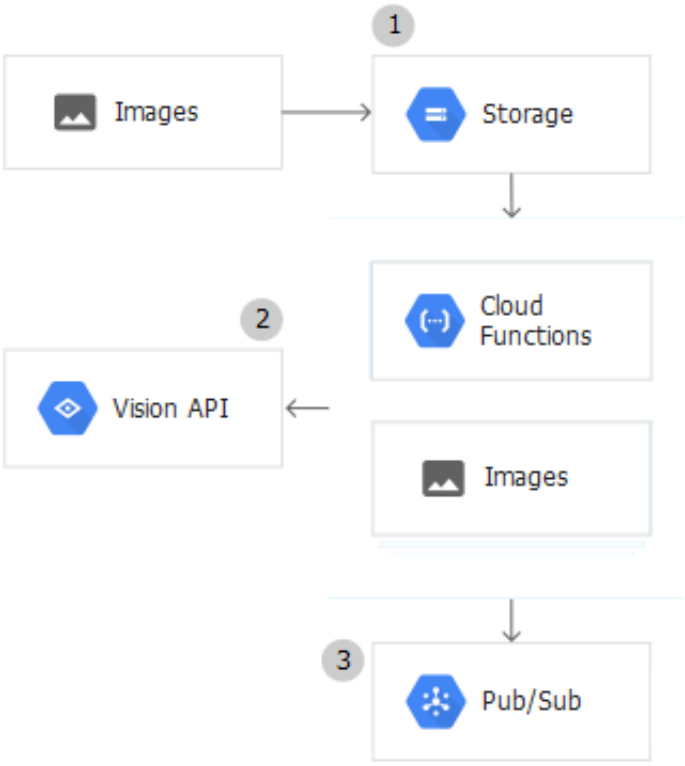
The Email alert will look like this:

The console should look like this

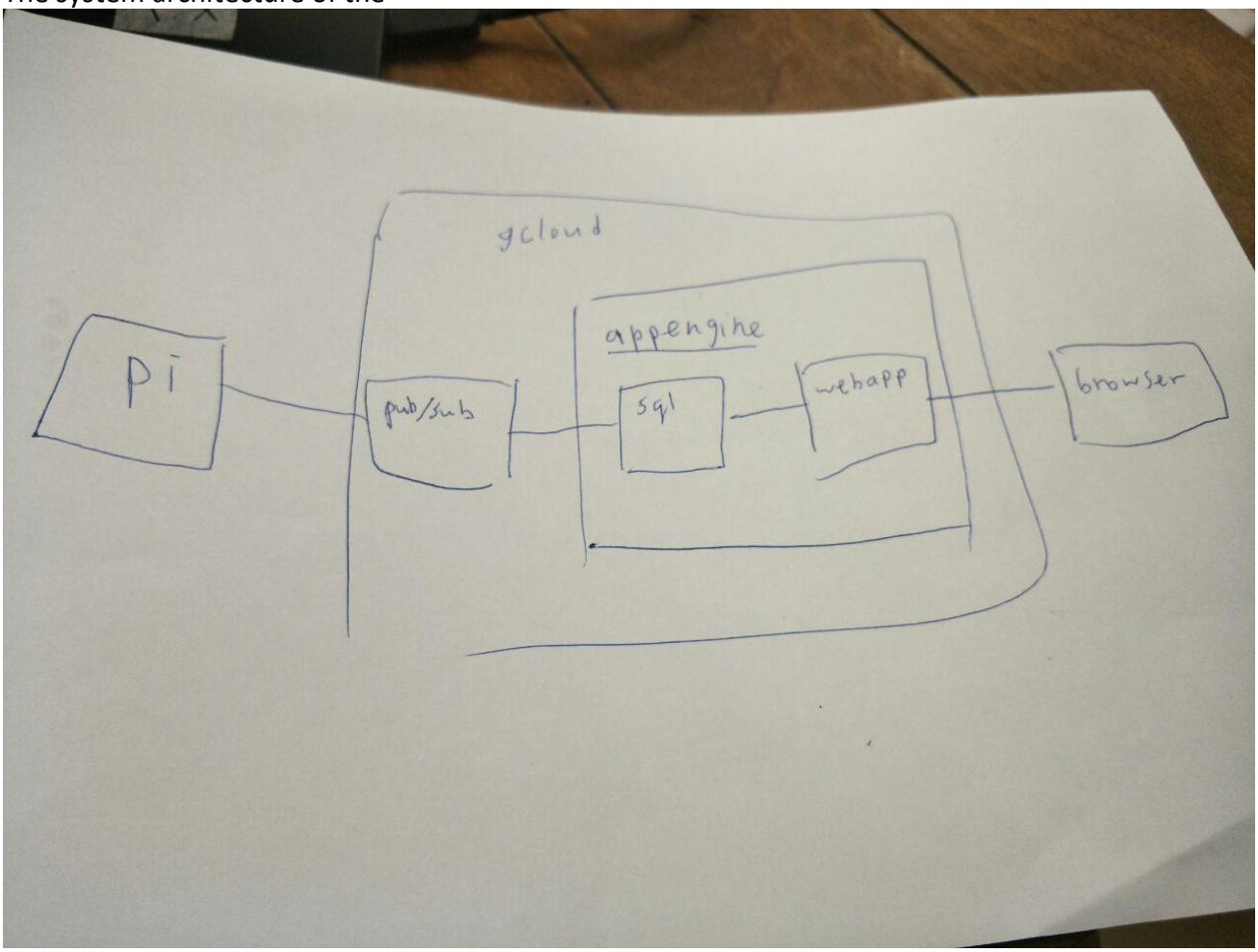


F. System architecture of our system

The System architecture of the image recognition.



The system architecture of the



G. Evidence that we have met basic requirements

Requirement	Evidence
Used three sensors	Used Camera, and light detector as sensors, led and servos as actuator
Used MQTT	Our MQTT endpoint --> Google cloud platform Example of data sent through MQTT : Blob of images, light value : 0.8
Stored data in cloud	Stored People detected data in Google Cloud
Used cloud service	Used Google Cloud Platform with AppEngine, IOTCore, Storage and Vision
Provide real-time sensor value / status	Show historical values of whether theres any body visible and detected by the camera
Provide historical sensor value/ status	Show historical values of whether theres any body visible and detected by the camera
Control actuator	Control Servos through the WebApp

H. Bonus features on top of basic requirements

Provide bullet list of the bonus features you have added on top of basic requirements

a) Image Recognition
b) SSL Email system
c) Audio Alert System
d) Outline Faces
e) Store to Google Cloud
f) Independent Power Source
g) Control the movement of the servos via wasd keys

I. Quick-start guide (Readme first)

The following list is the instruction to set up the web application.

1) First connect the hardware using the fritzing diagram located in Section B as a guide
2) Git Clone the Repository from git hub
3) Select or create a Cloud Platform project.
4) Enable billing for your project.
5) Pip install the libraries to install all the packages required
6) Run Main.py for all the functions
7) Run dataup.py to publish data from the light sensor to the IOT Topic

Section 2
Hardware requirements

Hardware checklist

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
		 

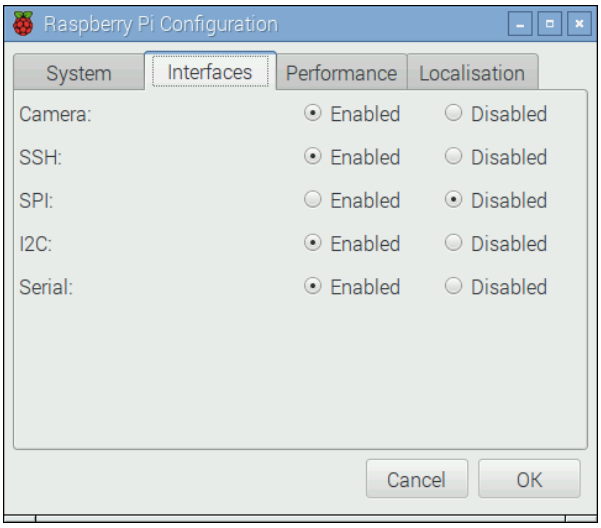
SG90 Servo Motor

	Task	

b)	• Servo motors (or servos) are self-contained electric devices that rotate or push parts of a machine with great precision.
	• The heart of a servo is a small direct current (DC) motor, similar to what you might find in an inexpensive toy.
	• These motors run on electricity from a battery and spin at high RPM (rotations per minute) but put out very low torque
	• The specifications of the motor are the following:
	  ○ Speed = 0.32 oz
	  ○ Torque = 4.8V: 25.0 oz-in (1.80 kg-cm)
		

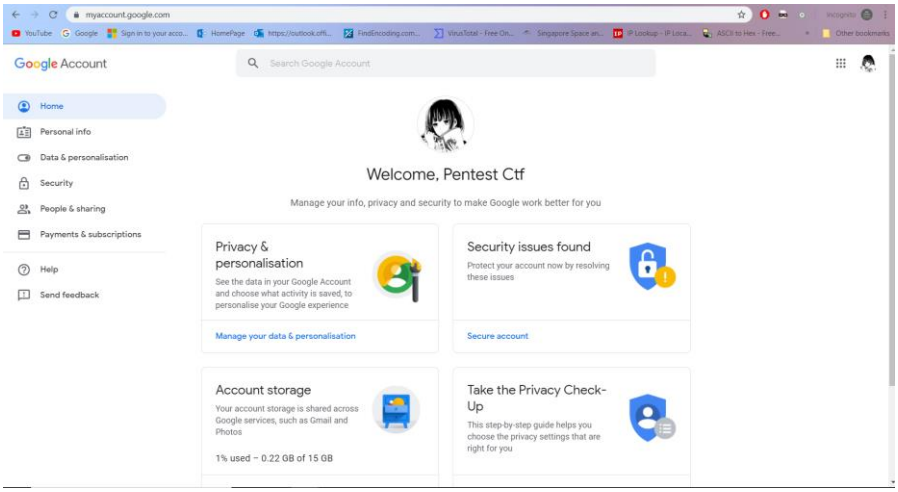
Section 3
Configuration

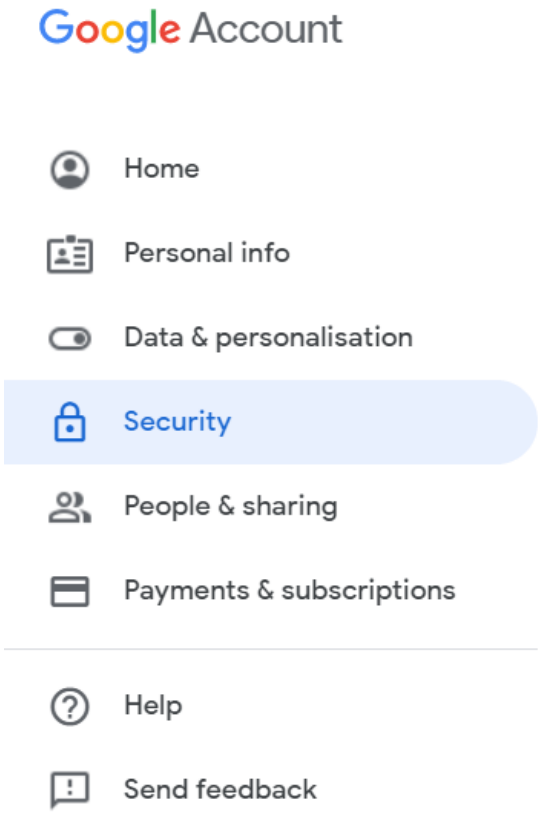
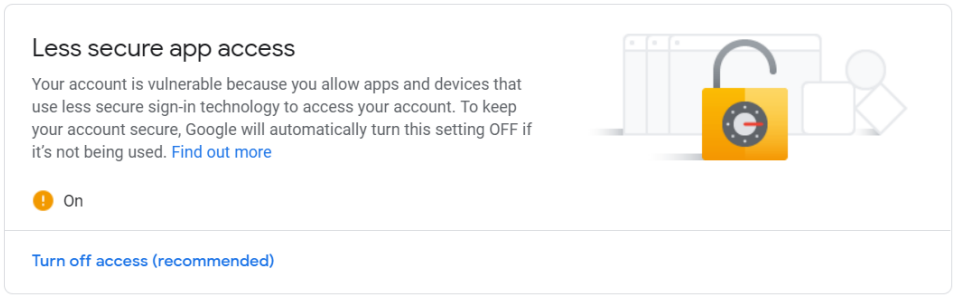
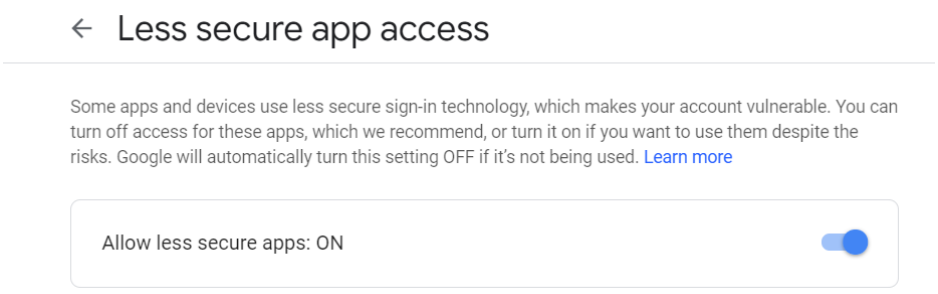
A. Enable camera with raspi-config

No.	Task	
a)	• Open the Raspberry Pi Configuration Tool from the main menu:	
b)	• Ensure the camera software is enabled. If it's not enabled, enable it and reboot your Pi to begin.	

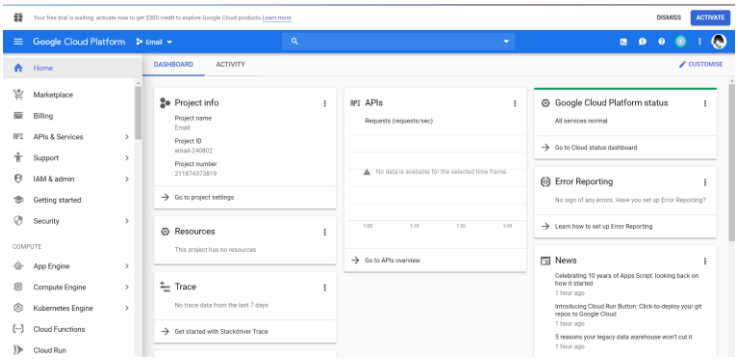
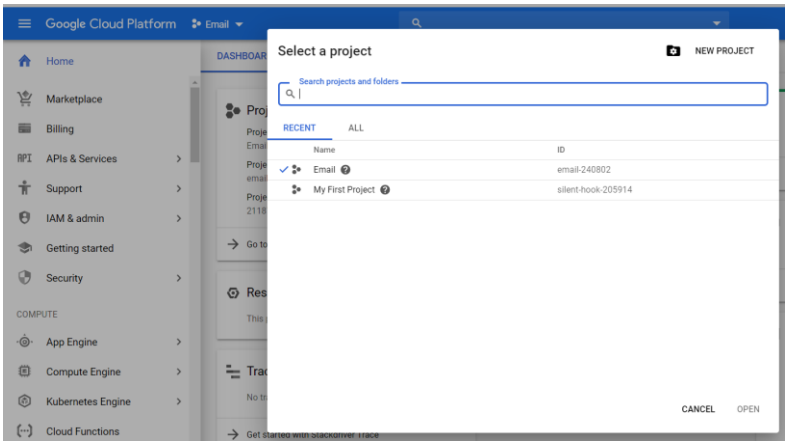
b)Configure Gmail Account

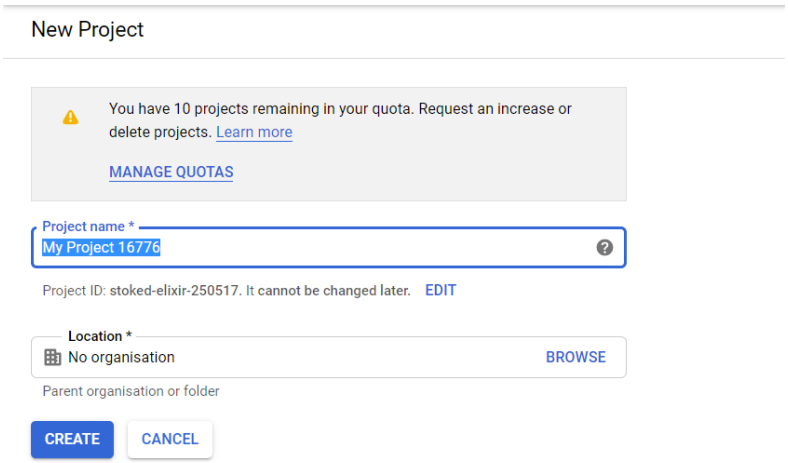
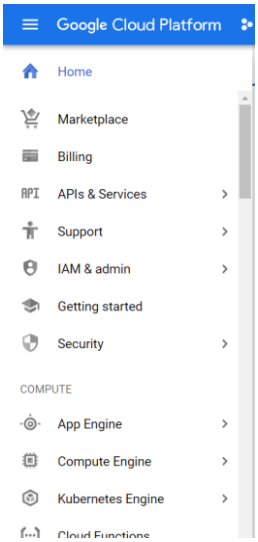
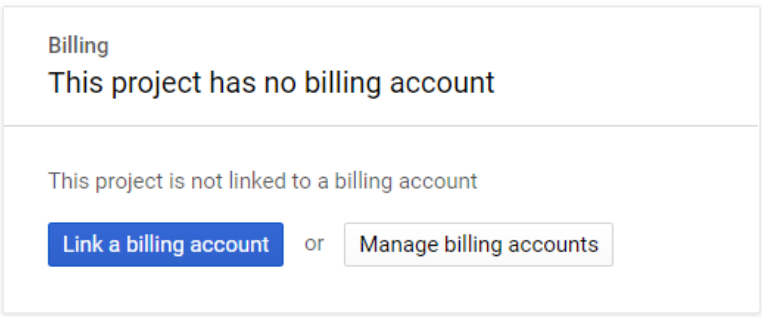
	Task
a)	Go to https://www.google.com/gmail/ and log in with your credentials

Task	
b)	If you don't have one, you can create a gmail account for free easily following this guide (https://support.google.com/mail/answer/56256?hl=en). Follow the instructions to apply for the account so that you can access the Google APIs. You can block sign-in attempts from some apps or devices that are less secure. Apps that are less secure don't use modern security standards, such as OAuth. Using apps and devices that don't use modern security standards increases the risk of accounts being compromised. Blocking these apps and devices helps keep your users and data safe.
c)	Sign in to your Google Admin console. https://myaccount.google.com/ 

Task	
d) From the Admin console Home page, go to Security and then Basic settings. To see Security on the Home page, you might have to click More controls at the bottom.	
e) Scroll down until you see the tab on the left saying less secure app access	
f) Press the button and allow less secure app access	

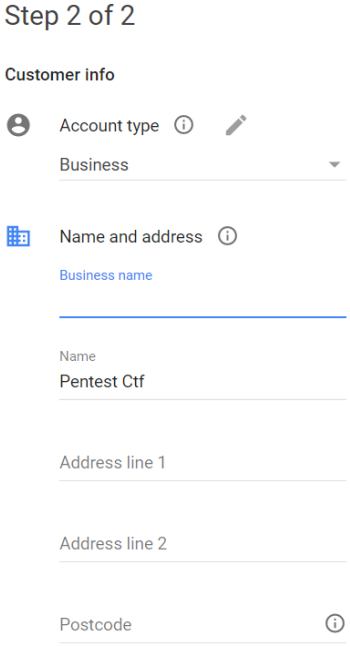
c) Configure Google Cloud

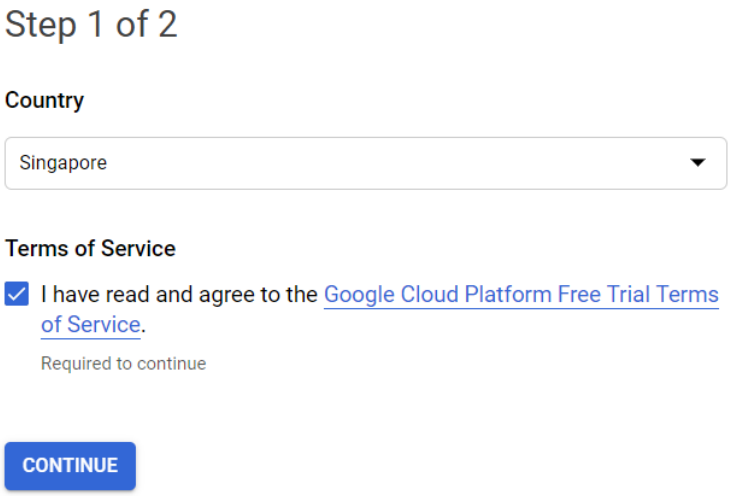
	Task
a)	Go to https://console.cloud.google.com and log in with your credentials
b)	If you don't have one, you can create a gmail account for free easily following this guide (https://support.google.com/mail/answer/56256?hl=en). Follow the instructions to apply for the account so that you can access the Google APIs.
c)	You will see this Dashboard 
d)	Click on the project selection and click new project 

Task	
e) You will be faced with this selection next, name your project what you wishbut keep track of your project ID for later	
f) After Creation, head to the drop down bar and set up your billing account as it will be needed to enable the additional features. You can follow the billing guide here (https://cloud.google.com/billing/docs/)	
g) Click Link a billing account and create a billing account	

Task

h) Select Singapore, after which fill up contact and credit card details pubsub mqtt data.





d)Configure Google Bucket

Task

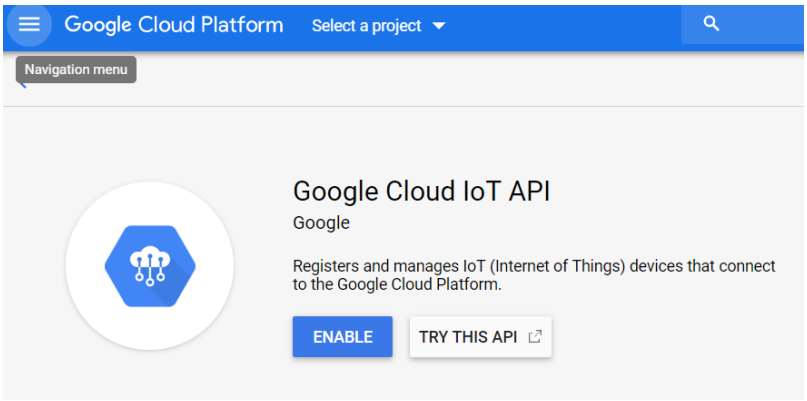
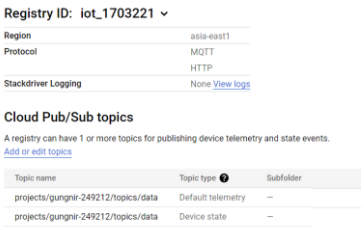
i) In your terminal window, create a Cloud Storage bucket, where [YOUR_BUCKET_NAME] represents the name of your bucket:

```
gsutil mb gs://[YOUR_BUCKET_NAME]
```

j) To view uploaded images in the Bookshelf app, set the bucket's default access control list (ACL) to public-read:

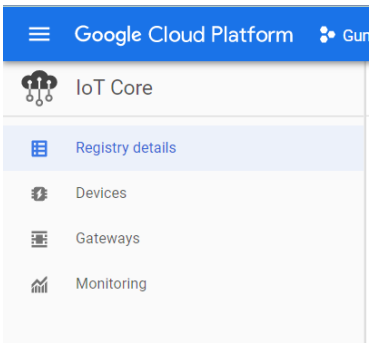
```
gsutil defacl set public-read gs://[YOUR_BUCKET_NAME].
```

e)Configure Google Cloud IOT CORE

	Task
g)	Go to https://console.cloud.google.com and log in with your credentials
h)	If you don't have one, you can create a gmail account for free easily following this guide (https://support.google.com/mail/answer/56256?hl=en).
i)	Go your project and enable the various apis that you might use including but not limited to google iot core, vision and more. 
j)	a) Click **Create registry.** b) Enter `my-registry` for the **Registry ID**. c) If you're in the US, select **us-central1** for the **Region**. If you're outside the US, select your preferred region. d) Select **MQTT** for the **Protocol**. e) In the **Default telemetry topic** dropdown list, select **Create a topic.** f) In the **Create a topic** dialog, enter `my-device-events` in the **Name** field. g) Click **Create** in the **Create a topic** dialog. h) The **Device state topic** and **Certificate value** fields are optional, so leave them blank. i) Click **Create** on the Cloud IoT Core page. You've just created a device registry with a Cloud Pub/Sub topic for publishing device telemetry events. 

Task

k) You can create gateways and more registries here as well as send commands and configure the mqttt protocol. Here you can also add devices that will be required later



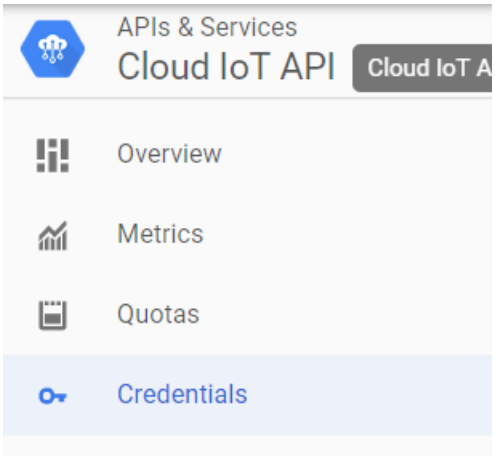
f) Configure Google Credentials

Task

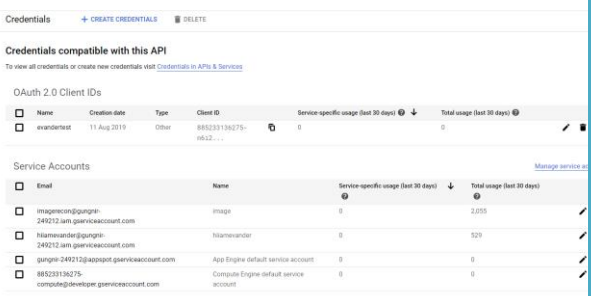
a) Go to https://console.cloud.google.com and log in with your credentials

b) If you don't have one, you can create a gmail account for free easily following this guide (https://support.google.com/mail/answer/56256?hl=en).

c) Go to IOT code and select credentials by the side



d) Select Create credentials, follow the instrutions provided and in the end it should generate a json file that you can then use as your credentials



Section 4
Software Requirements

a)Software checklist

Below is a list of libraries that I have imported and used for each of the python script.

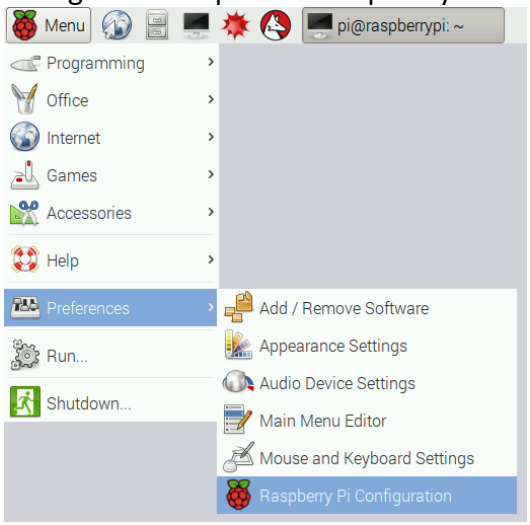
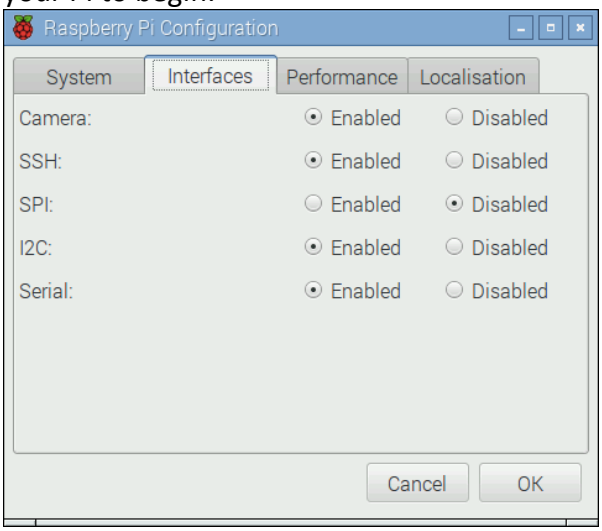
main.py	Faces_detection.py	publish.py	receive.py
1. base64 2. json 3. os 4. smtplib 5. time 6. sys 7. random:randrange 8. pygame:mixer 9. google.cloud: pubsub_v1 10. google.cloud: storage 11. google.cloud:vision 12. time:sleep 13. datetime : datetime 14. picamera: PiCamera 15. google.cloud.vision: types 16. face_detection:highlight_faces 17. PIL: Image, ImageDraw 18. email.mime.text:MIMEText 19. email.mime.image: MIMEImage 20. email.mime.multipart:MIMEMultipart	1. PIL :Image, ImageDraw	1. receive :receive_messages 2. publish: publish_messages 3. google.oauth2 : service_account 4. gpiozero : MCP3008 5. time :sleep 6. RPi.GPIO : GPIO	1. mysql.connector

Section 5
Software SetUp

b)Software setup instructions

No.	Task

1)	To implement the google cloud functions, the libraries need to be installed on the Raspberry Pi.
	
	Install Google Cloud Storage (https://pypi.org/project/google-cloud-storage/)
	`sudo pip install google-cloud-storage`
	
	Install Google Cloud Vision (https://pypi.org/project/google-cloud-vision/)
	`sudo  install google-cloud-vision`
	
	Install Pillow (https://pypi.org/project/Pillow/2.2.1)
	`sudo  pip install Pillow`

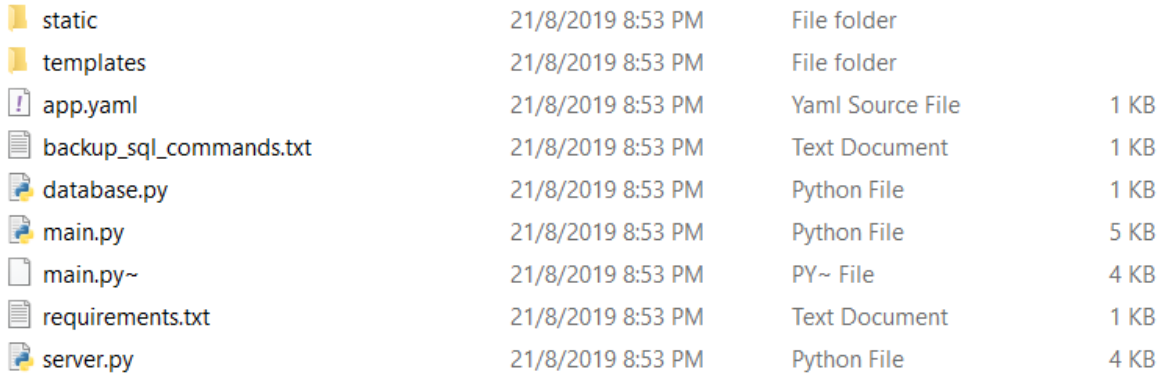
2)	To implement the Pi Camera, the camera has to be enable inside the raspberry pi configuration. Open the Raspberry Pi Configuration Tool from the main menu:  • Ensure the camera software is enabled. If it's not enabled, enable it and reboot your Pi to begin. 
4)	To implement MySQL database, the MySQL libraries need to be installed. If you are using the ST0324 IoT Raspbian image, you might encounter an error as the libraries have

	already been pre-installed. If you are using a fresh image, you would need to run the commands below to install the MySQL libraries.
	
	`sudo pip install mysql-connector-python`
	`sudo pip install mysql-connector`

Section 6
Source codes

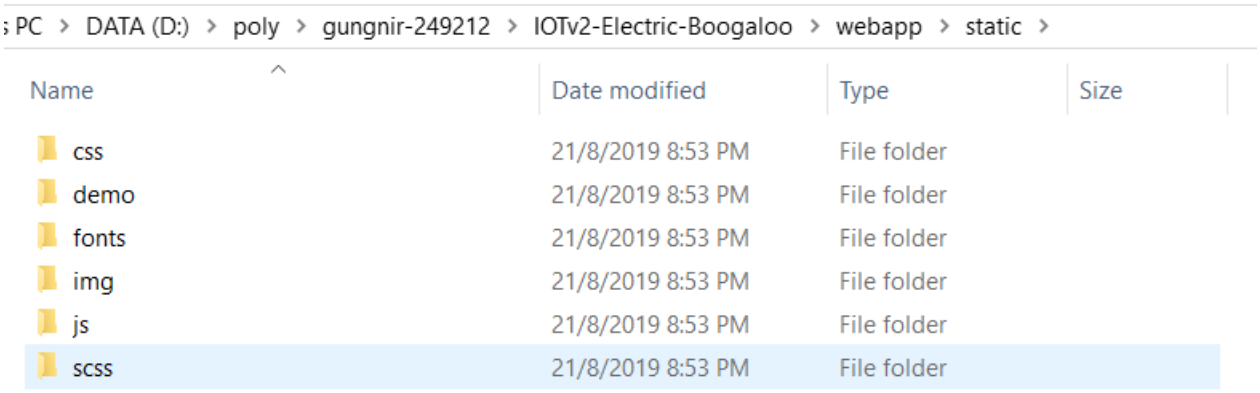
a)Bootstrap

1. After you have git cloned the repository, change directory into WebApp
2. Run the command export GOOGLE_APPLICATION_CREDENTIALS="../credentials/credentials_davis.json" to export the credentials necessary to access google applications
3. Run the command gcloud app deploy to deploy the app to the app engine



static	21/8/2019 8:53 PM	File folder	
templates	21/8/2019 8:53 PM	File folder	
app.yaml	21/8/2019 8:53 PM	Yaml Source File	1 KB
backup_sql_commands.txt	21/8/2019 8:53 PM	Text Document	1 KB
database.py	21/8/2019 8:53 PM	Python File	1 KB
main.py	21/8/2019 8:53 PM	Python File	5 KB
main.py~	21/8/2019 8:53 PM	PY~ File	4 KB
requirements.txt	21/8/2019 8:53 PM	Text Document	1 KB
server.py	21/8/2019 8:53 PM	Python File	4 KB

4. For this web application, We will be using a free bootstrap template called Paper Dashboard 2 that has already been downloaded. It is also available here at (https://www.creative-tim.com/product/paper-dashboard-2)
5. The asset and css folders have been already copied into the static folder



; PC › DATA (D:) › poly › gungnir-249212 › IOTv2-Electric-Boogaloo › webapp › static ›

Name	Date modified	Type	Size
css	21/8/2019 8:53 PM	File folder	
demo	21/8/2019 8:53 PM	File folder	
fonts	21/8/2019 8:53 PM	File folder	
img	21/8/2019 8:53 PM	File folder	
js	21/8/2019 8:53 PM	File folder	
scss	21/8/2019 8:53 PM	File folder	

b)index.html

Create a *index.html* file in the template folder with the following code below

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="apple-touch-icon" sizes="76x76" href="../static/img/apple-icon.png">
  <link rel="icon" type="image/png" href="../static/img/favicon.png">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
  <title>
    Paper Dashboard 2 by Creative Tim
  </title>
  <meta content='width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
scalable=0, shrink-to-fit=no' name='viewport' />
  <!--     Fonts and icons     -->
  <link href="https://fonts.googleapis.com/css?family=Montserrat:400,700,200"
rel="stylesheet" />
  <link href="https://maxcdn.bootstrapcdn.com/font-awesome/latest/css/font-
awesome.min.css" rel="stylesheet">
  <!-- CSS Files -->
  <link href="../static/css/bootstrap.min.css" rel="stylesheet" />
  <link href="../static/css/paper-dashboard.css?v=2.0.0" rel="stylesheet" />
  <!-- CSS Just for demo purpose, don't include it in your project -->
  <link href="../static/demo/demo.css" rel="stylesheet" />
</head>

<body class="">
  <div class="wrapper ">
    <div class="sidebar" data-color="white" data-active-color="danger">
      <!--
        Tip 1: You can change the color of the sidebar using: data-color="blue |
green | orange | red | yellow"
    -->
      <div class="logo">
         <img src="../static/img/sp.png">
      </div>
      <div class="sidebar-wrapper">
        <ul class="nav">
          <li class="active ">
            <a href="./index.html">
              <i class="nc-icon nc-bank"></i>
              <p>Index</p>
            </a>
          </li>
        </ul>
      </div>
    </div>

    <div class="main-panel">
      <!-- Navbar -->
      <nav class="navbar navbar-expand-lg navbar-absolute fixed-top navbar-
transparent">
        <div class="container-fluid">
          <div class="navbar-wrapper">
            <div class="navbar-toggle">
```

```
              <button type="button" class="navbar-toggler">
                <span class="navbar-toggler-bar bar1"></span>
                <span class="navbar-toggler-bar bar2"></span>
                <span class="navbar-toggler-bar bar3"></span>
              </button>
            </div>
            <a class="navbar-brand" href="#pablo">IOT CA1: Resorts World Sentosa
v1</a>
          </div>

        </div>
      </nav>
      <!-- End Navbar -->
      <!-- <div class="panel-header panel-header-lg">

  <canvas id="bigDashboardChart"></canvas>


</div> -->
      <div class="content">
        <div class="row">
          <div class="col-lg-3 col-md-0 col-sm-0">
          </div>
          <div class="col-lg-3 col-md-6 col-sm-6">
            <div class="card card-stats">
              <div class="card-body ">
                <div class="row">
                  <div class="col-5 col-md-4">
                    <div class="icon-big text-center icon-warning">
                  <img id="light-bulb" src="../static/img/icons/light-bulb-
off.png">
                    </div>
                  </div>
                  <div class="col-7 col-md-8">
                    <div class="numbers">
                      <p class="card-category">Status of LED</p>
                      <p class="card-title" id="lightStatus">Off<p>
                    </div>
                  </div>
                </div>
              </div>
              <div class="card-footer ">
                <hr>
                <div class="stats">
                  <button type="button" class="btn btn-outline-primary"
id="toggle">Toggle Light</button>
                </div>
              </div>
            </div>
          </div>
        </div>
            <div class="row">
                  <div class="col-md-12">
                        <div class="card ">
                              <div class="row">
                                    <div class="col-md-6">
                                          <div class="card-header ">
                                                <h5 class="card-title">Real
Time</h5>
                                          </div>
```

```
                                        <img id="rt" width="500px"
src="https://storage.cloud.google.com/images1703221/current?authuser=1"/>

                                    </div>
                                    <div class="col-md-6">
                                        <div class="card-header ">
                                            <h5 class="card-
title">Analysis</h5>
                                        </div>
                                        <img id="an" width="500px"
src="https://storage.cloud.google.com/images1703221/visitor?authuser=1"/>
                                    </div>
                                </div>
                            </div>
                        </div>
            </div>
          <div class="row">
            <div class="col-md-12">
              <div class="card ">
                <div class="card-header ">
                  <h5 class="card-title">Number of people</h5>
                  <p class="card-category">powered by yolo</p>
                </div>
                <div class="card-body ">
              <div id="chart_div" style="width:100%"></div>
              <div id="table_div" style="width:100%"></div>
                </div>
                <div class="card-footer ">
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>

<script type="text/javascript" src="https://code.jquery.com/jquery-
3.2.1.js"></script>
    <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
         google.charts.load('current', {'packages':['corechart','table']});
        // Set a callback to run when the Google Visualization API is loaded.
        google.charts.setOnLoadCallback(googlecharts_is_ready);

        var chart;
        var graphdata;

        function reset_status_messages(){
            $("#status").html("")
        }

        function googlecharts_is_ready(){
            $("#buttonloadchart").show()
            $("#buttonloadchart").click()
            $("#status").html("Google charts is ready")
        }

        function loadChart(){
               getData_and_drawChart()
        }
```

```
        function getData_and_drawChart(){
            getNewData()
        }


        function getNewData(){
            $("#status").html("Fetching data to plot graph...");

            jQuery.ajax({
                url: "/api/getdata" ,
                type: 'POST',
                success: function(ndata, textStatus, xhr){
                    console.log(ndata.chart_data.data)
                    $("#status").html("Data fetched! Now plotting graph!");
                    chartdata = ndata.chart_data.data
                    graphdata = createDataTable(chartdata)
                    drawLineChart(graphdata)
                    drawDataTable(graphdata)
                    $("#status").html("Graph plotted");
                }//end success
            });//end ajax
          } //end getNewData

        function createDataTable(newdata){
            graphdata = new google.visualization.DataTable();
            graphdata.addColumn('string', 'Time');
            graphdata.addColumn('number', 'Light Value');
            for (i in newdata) {
                datetime = newdata[i].datetime_value;
                jsdatetime = new Date(Date.parse(datetime));
                jstime = jsdatetime.toLocaleTimeString();
                light = newdata[i].light;
                graphdata.addRows([[jstime,light]]);
            }//end for
            return graphdata
        }

        function drawDataTable(graphdata){
            var table = new
google.visualization.Table(document.getElementById('table_div'));
            table.draw(graphdata, {showRowNumber: true, width: '100%', height:
'100%'});

        }//end drawTable

        function drawLineChart(graphdata) {
            chart = new google.visualization.LineChart(
            document.getElementById('chart_div'));
            chart.draw(graphdata, {legend: 'none', vAxis: {baseline: 0},
                colors: ['#A0D100']});
            return
        } //end drawChart

        $(document).ready(function(){
            reset_status_messages()

            setInterval(function () {
                loadChart()
            }, 3000);
        });
```

```
</script>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>



<script><!--code for light toggle-->


var imgOne=true;
var URLimgOne="../static/img/icons/light-bulb-off.png";
var URLimgTwo="../static/img/icons/light-bulb-on.png";

function updateImg(){
      var currUrl;
      if(imgOne){
            imgOne=false;
            currUrl=URLimgTwo;
      }else{
            imgOne=true;
            currUrl=URLimgOne;
      }
      document.getElementById("light-bulb").src=currUrl;
}

                function toggle(){
                  $.ajax({url: "toggle",
                          success: function(result){
                                        $("#lightStatus").html(result);
                          }

                  })
            }


                $(document).ready(function(){
                    $("#toggle").click(function(){
                              toggle();
                        updateImg();
                    });
            });
</script>
<script><!--refresh for yolo image-->
window.onload = function() {
    var image = document.getElementById("rt");
    var image2 = document.getElementById("an");
    function updateImage() {
        image.src = image.src.split("?")[0] + "?" + new Date().getTime();
        image2.src = image2.src.split("?")[0] + "?" + new Date().getTime();
    }

    setInterval(updateImage, 5000);
}
</script>


<script>
document.onkeydown = function(e) {
```

```
    switch (e.keyCode) {
        case 37:
            moveL();
            break;
        case 38:
            moveU();
            break;
        case 39:
            moveR();
            break;
        case 40:
            moveD();
            break;
    }
};


function moveL(){
      $.ajax({url: "moveL",
      success: function(result){
      }

})
function moveR(){
      $.ajax({url: "moveR",
      success: function(result){
      }

})
function moveU(){
      $.ajax({url: "moveU",
      success: function(result){
      }

})
function moveD(){
      $.ajax({url: "moveD",
      success: function(result){
      }

})

</script>


  <!--   Core JS Files   -->
  <script src="../static/js/core/jquery.min.js"></script>
  <script src="../static/js/core/popper.min.js"></script>
  <script src="../static/js/core/bootstrap.min.js"></script>
  <script src="../static/js/plugins/perfect-scrollbar.jquery.min.js"></script>
  <!--  Google Maps Plugin    -->
  <script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY_HERE"></script>
  <!-- Chart JS -->
  <script src="../static/js/plugins/chartjs.min.js"></script>
  <!--  Notifications Plugin    -->
  <script src="../static/js/plugins/bootstrap-notify.js"></script>
  <!-- Control Center for Now Ui Dashboard: parallax effects, scripts for the
example pages etc -->
  <script src="../static/js/paper-dashboard.min.js?v=2.0.0"
type="text/javascript"></script>
```

```
  <!-- Paper Dashboard DEMO methods, don't include it in your project! -->
  <script src="../static/demo/demo.js"></script>
  <script>
    $(document).ready(function() {
      // Javascript method's body can be found in assets/assets-for-
demo/js/demo.js
      demo.initChartsPages();
    });
  </script>
</body>

</html>
```

c) main.py

To upload the images and perform cloud functions and recognition as well as other functions that have been included, the following code has been provided.

```
import base64
import json
import os
import smtplib
import time, sys
from random import randrange
from pygame import mixer
from google.cloud import pubsub_v1
from google.cloud import storage
from google.cloud import vision
from time import sleep
from datetime import datetime
from picamera import PiCamera
from google.cloud.vision import types
from face_detection import highlight_faces
from PIL import Image, ImageDraw
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

gmail_user = 'pentestnil@gmail.com'
gmail_password = '1qwe$#@!'

credential_path = "../credentials/credentials_davis.json"
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = credential_path

sent_from = 'pentestnil@gmail.com'
to = ['t0016029b@gmail.com']
From = "pentestnil@gmail.com"
To = "t0016029b@gmail.com"
delay=2
client = vision.ImageAnnotatorClient()

def detect_face(face_file, max_results=4):
```

```
    """Uses the Vision API to detect faces in the given file.

    Args:
        face_file: A file-like object containing an image with faces.

    Returns:
        An array of Face objects with information about the picture.
    """
    client = vision.ImageAnnotatorClient()

    content = face_file.read()
    image = types.Image(content=content)

    return client.face_detection(image=image,
max_results=max_results).face_annotations
def warning(option):
    mixer.init()
    if option == 1:
        sound = mixer.Sound('target.wav')
    else:
        sound = mixer.Sound('seeyou.wav')
    sound.play()
    time.sleep(sound.get_length()+1)

def emailalert():
    try:
        img_data = open(full_path_post, 'rb').read()
        msg = MIMEMultipart()
        msg['Subject'] = 'Alert!!'
        text = MIMEText("Hey, you have a visitor\n\n -Davis ")
        msg.attach(text)
        image = MIMEImage(img_data, name=os.path.basename(full_path_post))
        msg.attach(image)
        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.ehlo()
        server.login(gmail_user, gmail_password)
        server.sendmail(From,To, msg.as_string())
        server.close()
        print ('Email sent!')
    except:
        print ('Something went wrong...')


def takePhotoWithPiCam():
    camera.capture(full_path)
    sleep(delay)

def faces(input_filename, output_filename, max_results):
    with open(input_filename, 'rb') as image:
```

```
        faces = detect_face(image, max_results)
        print('Found {} face{}'.format(
            len(faces), '' if len(faces) == 1 else 's'))

        print('Writing to file {}'.format(output_filename))
        # Reset the file pointer, so we can read the file again
        image.seek(0)
        highlight_faces(image, faces, output_filename)
    if len(faces) == 1:
        print("hi")
        emailalert()
        warning(randrange(2))
        upload_blob("images1703221",full_path_post,"visitor")
        raise SystemExit

def upload_blob(bucket_name, source_file_name, destination_blob_name):
    """Uploads a file to the bucket."""
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print('File {} uploaded to {}.'.format(
        source_file_name,
        destination_blob_name))

# Set the filename and bucket name
bucket = 'images1703221' # replace with your own unique bucket name
with PiCamera() as camera:
    while True:
        full_path = ('/home/pi/Desktop/pre.jpg')
        full_path_post = ('/home/pi/Desktop/post.jpg')
        file_name = (str(datetime.now()) + ' pre.jpg')
        takePhotoWithPiCam()
        faces(full_path,full_path_post,4)
        upload_blob("images1703221",full_path,"current")
```

d)face_detection.py

Create a python script using the code below called *face_detection.py* that will use the image recognition returned and highlight the people identified in the image and output the image which will be uploaded online

```
from PIL import Image, ImageDraw
def highlight_faces(image, faces, output_filename):
    """Draws a polygon around the faces, then saves to output_filename.
```

```
    Args:
      image: a file containing the image with the faces.
      faces: a list of faces found in the file. This should be in the format
          returned by the Vision API.
      output_filename: the name of the image file to be created, where the
          faces have polygons drawn around them.
    """
    im = Image.open(image)
    draw = ImageDraw.Draw(im)
    # Sepecify the font-family and the font-size
    for face in faces:
        box = [(vertex.x, vertex.y)
               for vertex in face.bounding_poly.vertices]
        draw.line(box + [box[0]], width=5, fill='#00ff00')
        # Place the confidence value/score of the detected faces above the
        # detection box in the output image
        draw.text(((face.bounding_poly.vertices)[0].x,
                   (face.bounding_poly.vertices)[0].y - 30),
                  str(format(face.detection_confidence, '.3f')) + '%',
                  fill='#FF0000')
im.save(output_filename)
```

e) publish.py

Create a python script called *publish.py* using the code below. It will retreive the data from the light sensor and publish via mqtt.

```
def publish_messages(project_id, topic_name,data):
    """Publishes multiple messages to a Pub/Sub topic."""
    # [START pubsub_quickstart_publisher]
    # [START pubsub_publish]
    from google.cloud import pubsub_v1
    data = str(data)
    publisher = pubsub_v1.PublisherClient()
    # The `topic_path` method creates a fully qualified identifier
    # in the form `projects/{project_id}/topics/{topic_name}`
    topic_path = publisher.topic_path(project_id, topic_name)
    print(topic_path)
    # Data must be a bytestring
    print("data :" + data)
    data = data.encode('utf-8')
    # When you publish a message, the client returns a future.
    future = publisher.publish(topic_path, data=data)
    print(future.result())
    print('Published messages.')
```

f) receive.py

Create a python script called *retreive.py* using the code below which allows for subscribing data from mqtt.

```
def receive_messages():
    """Receives messages from a pull subscription."""
    # [START pubsub_subscriber_async_pull]
    # [START pubsub_quickstart_subscriber]
    import time

    from google.cloud import pubsub_v1

    project_id = "gungnir-249212"
    topic_name = "data"
    subscription_name = "test_sub"

    subscriber = pubsub_v1.SubscriberClient()
    # The `subscription_path` method creates a fully qualified identifier
    # in the form `projects/{project_id}/subscriptions/{subscription_name}`
    subscription_path = subscriber.subscription_path(
        project_id, subscription_name)

    def callback(message):
        print('Received message: {}'.format(message))
        message.ack()

    subscriber.subscribe(subscription_path, callback=callback)

    # The subscriber is non-blocking. We must keep the main thread from
    # exiting to allow it to process messages asynchronously in the background.
    print('Listening for messages on {}'.format(subscription_path))
    while True:
        time.sleep(5)
```

g)sketch_aug19a.ino

Create a arduino sketch using the code below which allows the servos to move.

```
#include <Servo.h>
Servo base;
Servo plane;

void setup(){
  Serial.begin(9600);
  plane.attach(8);
  plane.write(22);
  base.attach(4);
  base.write(75);
```

```
  Serial.println("testing...");
}

void loop(){
  if(Serial.available()){
    Serial.print("loops ");
    int i = Serial.parseInt();
    Serial.println(i);
    delay(10);
    switch (i){
      case 1:
        base.write(base.read()+1);
        break;
      case 2:
        base.write(base.read()-1);
        break;
      case 3:
        plane.write(plane.read()+1);
        break;
      case 4:
        plane.write(plane.read()-1);
        break;
    }

    Serial.flush();
  }
}

© 2019 GitHub, Inc.
```

h)dataup.py

It retreives the data from the light sensor and uses publish.py to publish the information via mqtt

```
from receive import receive_messages
from publish import publish_messages
from google.oauth2 import service_account
from time import sleep
from gpiozero import MCP3008
import RPi.GPIO as GPIO
from random import randrange


adc = MCP3008(channel=0)
while True:
      light_value = adc.value
      publish_messages("gungnir-249212", "data",light_value)
      GPIO.setwarnings(False)
      if light_value > 0.9:
            print("LightOn")
```

```
            GPIO.output(18,GPIO.HIGH)
      else:
            GPIO.output(18,GPIO.LOW)
      sleep(9)
```

i) control.py

This is the python code to control the servos via serial

```
import sys, termios, tty, os, time
import serial
import time

port ="/dev/ttyUSB0"

s1 = serial.Serial(port,9600)
s1.flushInput()


def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)

    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

button_delay = 0.2

while True:
    time.sleep(.03)
    char = getch()

    if (char == "w"):
        s1.write('3\r')

    if (char == "s"):
        s1.write('4\r')

    if (char == "a"):
        s1.write('1\r')

    if (char == "d"):
        s1.write('2\r')

    if (char == "p"):
        break
```

```
    s1.flushInput()
© 2019 GitHub, Inc.
```

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