**SCHOOL OF COMPUTING (SOC)**

|  |  |
| --- | --- |
| **Date of Submission:** | 23/2/2020 |

|  |  |
| --- | --- |
| **Prepared for:** | Dora Chua Heok Hoon |

|  |  |
| --- | --- |
| **Class:** | DIT/2A/34 |

|  |  |
| --- | --- |
| **Submitted by:** | Briant Toh |

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 1828894  1829080  1829048 | Hyu Tan  Briant Toh  Joseph Chong Teik Yun |

**IOT CA2**

**Step-by-step Tutorial**

**DIPLOMA IN BUSINESS INFORMATION TECHNOLOGY**

**DIPLOMA IN INFORMATION TECHNOLOGY**

**DIPLOMA IN INFOCOMM SECURITY MANAGEMENT**

**ST0324 Internet of Things (IOT)**

**2017/2018 Semester 1**

**Table of Contents**

[Section 1 Overview of project 2](#_Toc29297787)

[A. Where we have uploaded our tutorial 2](#_Toc29297788)

[B. What is the application about? 2](#_Toc29297789)

[C. How does the final RPI set-up looks like? 2](#_Toc29297790)

[D. How does the web or mobile application look like? 2](#_Toc29297791)

[E. System architecture of our system 2](#_Toc29297792)

[F. Evidence that we have met basic requirements 3](#_Toc29297793)

[G. Bonus features on top of basic requirements 3](#_Toc29297794)

[A. Quick-start guide (Readme first) 3](#_Toc29297795)

[Section 2 Hardware requirements 5](#_Toc29297796)

[Hardware checklist 5](#_Toc29297797)

[Hardware setup instructions 5](#_Toc29297798)

[Fritzing Diagram 5](#_Toc29297799)

[Section 3 Software Requirements 6](#_Toc29297800)

[Software checklist 6](#_Toc29297801)

[Software setup instructions 6](#_Toc29297802)

[Section 4 Source codes 6](#_Toc29297803)

[server.py 6](#_Toc29297804)

[index.html 6](#_Toc29297805)

[Section 5 Task List 7](#_Toc29297806)

[Section 6 Any other section you want to add 7](#_Toc29297807)

[Section 7 References 7](#_Toc29297808)

# Section 1 Overview of project

* 1. Where we have uploaded our tutorial

Fill up the Google form here to submit your links and then paste the links here of your Youtube and tutorial document here as well.

<http://bit.ly/1910s2iotca2>

|  |  |
| --- | --- |
| **Youtube** | <https://youtu.be/iIPU5MANqfA> |
| **Public tutorial link** | <https://github.com/racko123/CCK-CA2> |

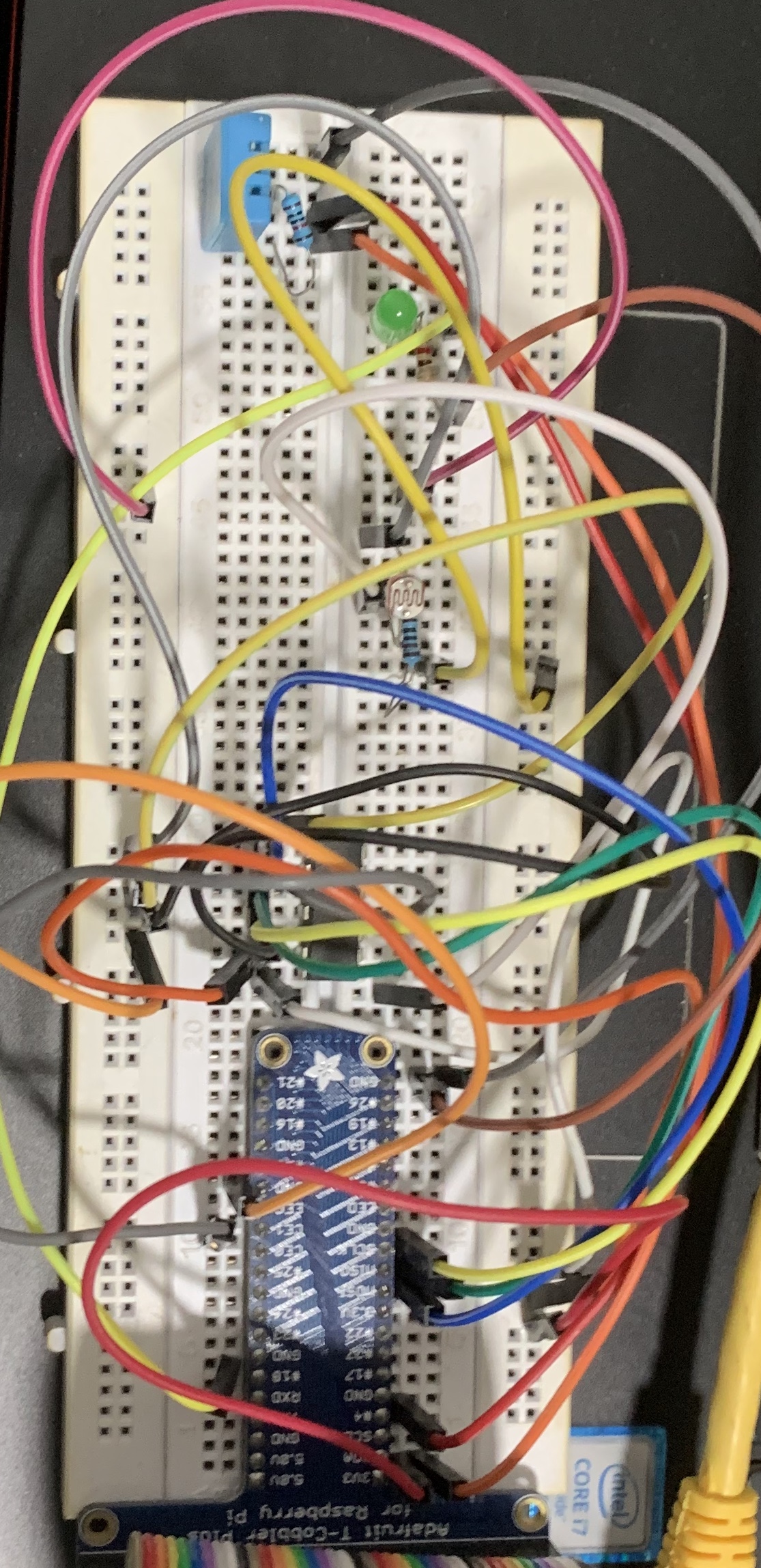
* 1. What is the application about?

Provide a brief description of your application here. Who is the target audience? How can your app help your target audience?

The application is developed to solve a problem of crops not developing as well as they should which will devastate the rate of harvest and affect our food supply as the demand for food is constantly increasing and more crops harvest is needed to meet the increasing demand. The target audience is farmers who grow crops. The app will make use of a dht sensor to allow farmers to detect temperature changes in the soil and store the temperature into the database for them to monitor and a light-dependent resistor to detect ambient light and to automatically turn on a light-emitting diode when the light level is low. This will help the crops develop much better and increase crop harvest rate significantly as soil temperature and light are key factors for crop growth. There is also a function to take a photo of the current crops and store it to the database so that the farmer can monitor the crops growth without physically being there and image recognition to detect intruders who want to cause disruption to their crops.

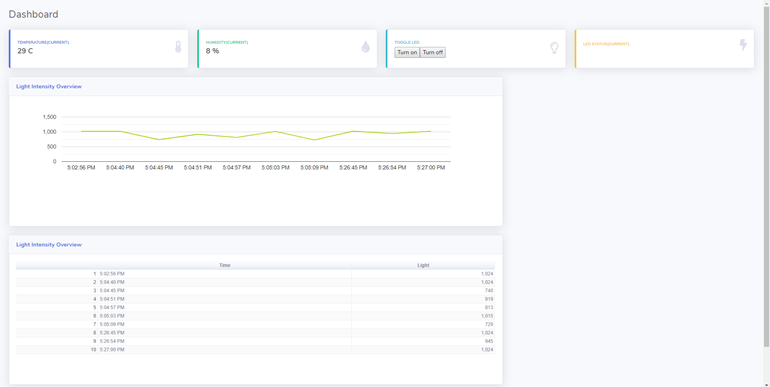
* 1. How does the final RPI set-up looks like?

Provide a photo of your final RPI hardware set-up. You may want to mark-up (annotate or draw arrows) and refer to this in Section F for instance.



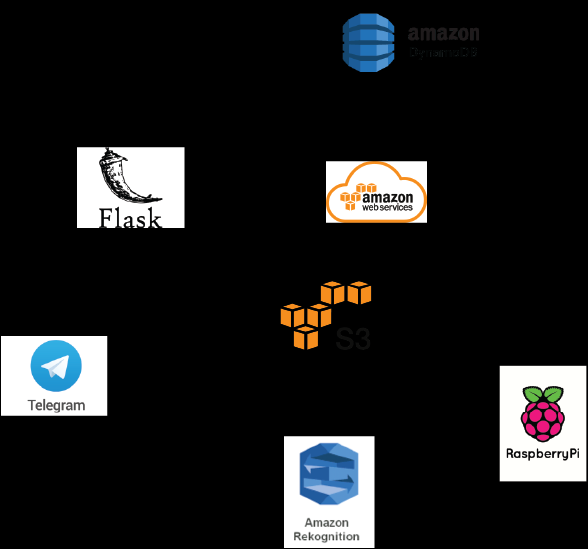
* 1. How does the web or mobile application look like?

Provide at least one screenshot of your web app, and more if your web app consists of more than 1 page. Otherwise, I will assume your webapp only can show 1 page. Label your screenshots so that they may be referenced in Section F.



* 1. System architecture of our system

Provide a hand-drawn or computer-drawn system architecture diagram please. Example given below.



* 1. Evidence that we have met basic requirements

Provide bullet list to describe how your group has met basic requirements

|  |  |
| --- | --- |
| Requirement | Evidence |
| Used three sensors | Used Light Dependent Resistor and Temperature sensor. Also Used LED light as an Acutator. |
| Used MQTT | Our MQTT endpoint --> Node-Red  Example of data sent through MQTT : Humidity,Temperature and Light Values |
| Stored data in cloud | Stored light,Humidity and Temperature data in Cloudant database in IBM cloud |
| Used cloud service | Use AWS Rekognition, hosted web server on EC@ |
| Provide real-time sensor value / status |  |
| Provide historical sensor value/ status |  |
| Control actuator |  |

* 1. Bonus features on top of basic requirements

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

1. Telegram bot
   1. Quick-start guide (Readme first)

Give a few lines of basic instructions on how I need to run your app, e.g

1. First connect hardware as in Section 2 Fritzing Diagram
2. Run database.py to store data in database
3. Run imagerecognition.py for image recognition
4. Run picam.py to take photo
5. For Node-Red First Start it up
6. Import the Flows.txt,Deploy then got to localhost/sensors/CA2

# Section 2 Hardware requirements

Hardware checklist

1. One DHT sensor
2. One LED
3. One Light-Dependent Resistor
4. One MCP3008

Hardware setup instructions

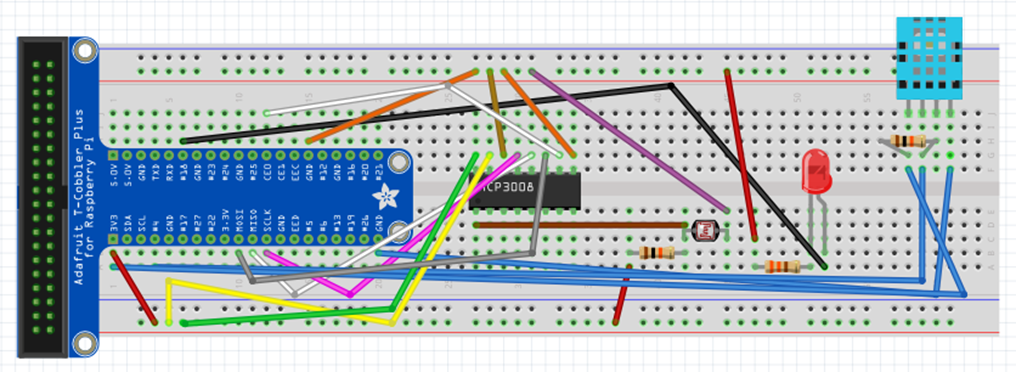
Describe any special setup instructions here

Follow the Fritzing Diagram to setup hardware

Fritzing Diagram

Paste a Fritzing diagram of your setup here

You can get the Fritzing software at Blackboard Labs folder (third link from top)

****

# Section 3 Software Requirements

Software checklist

If your applications needs the user to install additional Python or other libraries, pleasse provide here. A simple one like this is sufficient.

1. telepot library

Software setup instructions

Describe any special setup instructions here, e.g some libraries you need to pip install or some API key you need to create/request etc

pip install telepot

# Section 4 Source codes

All source codes, including Python, HTML files etc

### Boto\_s3\_1.py

import boto3

import botocore

from time import sleep

# Create an S3 resource

s3 = boto3.resource('s3')

full\_path = '/home/pi/Desktop/image1.jpg'

file\_name = 'image1.jpg'

def takePhotoWithPiCam():

from picamera import PiCamera

camera = PiCamera()

sleep(5)

camera.capture(full\_path)

sleep(3)

# Set the filename and bucket name

bucket = 'sp-p1828894-s3-bucket' # replace with your own unique bucket name

exists = True

try:

s3.meta.client.head\_bucket(Bucket=bucket)

except botocore.exceptions.ClientError as e:

error\_code = int(e.response['Error']['Code'])

if error\_code == 404:

exists = False

if exists == False:

s3.create\_bucket(Bucket=bucket,CreateBucketConfiguration={

'LocationConstraint': 'us-east-1'})

# Take a photo

takePhotoWithPiCam()

# Upload a new file

s3.Object(bucket, file\_name).put(Body=open(full\_path, 'rb'))

print("File uploaded")

### database.py

# Import SDK packages

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

from time import sleep

from gpiozero import MCP3008

import Adafruit\_DHT

adc = MCP3008(channel=0)

# Custom MQTT message callback

def customCallback(client, userdata, message):

print("Received a new message: ")

print(message.payload)

print("from topic: ")

print(message.topic)

print("--------------\n\n")

host = "asmfodsqyf6xn-ats.iot.us-east-1.amazonaws.com"

rootCAPath = "rootca.pem"

certificatePath = "certificate.pem.crt"

privateKeyPath = "private.pem.key"

my\_rpi = AWSIoTMQTTClient("basicPubSub")

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

# Connect and subscribe to AWS IoT

my\_rpi.connect()

my\_rpi.subscribe("sensors/light", 1, customCallback)

sleep(2)

# Publish to the same topic in a loop forever

loopCount = 0

while True:

light = round(1024-(adc.value\*1024))

pin = 4

humidity, temperature = Adafruit\_DHT.read\_retry(11, pin)

temperature=('{}'.format(temperature))

sleep(3)

humidity=round(humidity)

temperature=float(temperature)

loopCount = loopCount+1

message = {}

message["deviceid"] = "deviceid\_CCK"

import datetime as datetime

now = datetime.datetime.now()

message["datetimeid"] = now.isoformat()

message["light"] = light

message["temperature (C)"] = temperature

message["humidity (%)"] = humidity

import json

my\_rpi.publish("sensors/light", json.dumps(message), 1)

sleep(5)

### imagerecognition.py

import boto3

import botocore

from picamera import PiCamera

from time import sleep

import json

# Set the filename and bucket name

BUCKET = 'sp-p1828894-s3-bucket' # replace with your own unique bucket name

location = {'LocationConstraint': 'us-east-1'}

file\_path = "/home/pi/Desktop"

file\_name = "test.jpg"

def takePhoto(file\_path,file\_name):

with PiCamera() as camera:

#camera.resolution = (1024, 768)

full\_path = file\_path + "/" + file\_name

camera.capture(full\_path)

sleep(3)

def uploadToS3(file\_path,file\_name, bucket\_name,location):

s3 = boto3.resource('s3') # Create an S3 resource

exists = True

try:

s3.meta.client.head\_bucket(Bucket=bucket\_name)

except botocore.exceptions.ClientError as e:

error\_code = int(e.response['Error']['Code'])

if error\_code == 404:

exists = False

if exists == False:

s3.create\_bucket(Bucket=bucket\_name,CreateBucketConfiguration=location)

# Upload the file

full\_path = file\_path + "/" + file\_name

s3.Object(bucket\_name, file\_name).put(Body=open(full\_path, 'rb'))

print("File uploaded")

def detect\_labels(bucket, key, max\_labels=10, min\_confidence=90, region="us-east-1"):

rekognition = boto3.client("rekognition", region)

response = rekognition.detect\_labels(

Image={

"S3Object": {

"Bucket": bucket,

"Name": key,

}

},

MaxLabels=max\_labels,

MinConfidence=min\_confidence,

)

return response['Labels']

def detect\_faces(bucket, key, max\_labels=10, min\_confidence=90, region="us-east-1"):

rekognition = boto3.client("rekognition", region)

response = rekognition.detect\_faces(

Image={

"S3Object": {

"Bucket": bucket,

"Name": key,

}

},

Attributes=['ALL']

)

return response['FaceDetails']

takePhoto(file\_path, file\_name)

uploadToS3(file\_path,file\_name, BUCKET,location)

print('Detected faces for')

for faceDetail in detect\_faces(BUCKET, file\_name):

ageLow = faceDetail['AgeRange']['Low']

ageHigh = faceDetail['AgeRange']['High']

print('Age between {} and {} years old'.format(ageLow,ageHigh))

print('Here are the other attributes:')

print(json.dumps(faceDetail, indent=4, sort\_keys=True))

### picam.py

import boto3

import botocore

from picamera import PiCamera

from time import sleep

# Set the filename and bucket name

BUCKET = 'sp-p1828894-s3-bucket' # replace with your own unique bucket name

location = {'LocationConstraint': 'us-east-1'}

file\_path = "/home/pi/Desktop"

file\_name = "test.jpg"

def takePhoto(file\_path,file\_name):

with PiCamera() as camera:

#camera.resolution = (1024, 768)

full\_path = file\_path + "/" + file\_name

camera.capture(full\_path)

sleep(3)

def uploadToS3(file\_path,file\_name, bucket\_name,location):

s3 = boto3.resource('s3') # Create an S3 resource

exists = True

try:

s3.meta.client.head\_bucket(Bucket=bucket\_name)

except botocore.exceptions.ClientError as e:

error\_code = int(e.response['Error']['Code'])

if error\_code == 404:

exists = False

if exists == False:

s3.create\_bucket(Bucket=bucket\_name,CreateBucketConfiguration=location)

# Upload the file

full\_path = file\_path + "/" + file\_name

s3.Object(bucket\_name, file\_name).put(Body=open(full\_path, 'rb'))

print("File uploaded")

def detect\_labels(bucket, key, max\_labels=10, min\_confidence=90, region="us-east-1"):

rekognition = boto3.client("rekognition", region)

response = rekognition.detect\_labels(

Image={

"S3Object": {

"Bucket": bucket,

"Name": key,

}

},

MaxLabels=max\_labels,

MinConfidence=min\_confidence,

)

return response['Labels']

takePhoto(file\_path, file\_name)

uploadToS3(file\_path,file\_name, BUCKET,location)

highestconfidence = 0

best\_bet\_item = "Unknown"

for label in detect\_labels(BUCKET, file\_name):

print("{Name} - {Confidence}%".format(\*\*label))

if label["Confidence"] >= highestconfidence:

highestconfidence = label["Confidence"]

best\_bet\_item = label["Name"]

if best\_bet\_item!= "Unknown":

print("This should be a {} with confidence {}".format(best\_bet\_item, highestconfidence))

### index.html

<!doctype html>

<head>

<style> #chartDiv {width:100%;}</style>

<title>Google Charts with Flask</title>

<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');

for (i in newdata) {

datetime = newdata[i].datetime\_value;

jsdatetime = new Date(Date.parse(datetime));

jstime = jsdatetime.toLocaleTimeString();

light = newdata[i].light\_value;

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>

function turnon(){

$.ajax({url: "writeLED/On",

success: function(result){

$("#status").html(result);

}

})

}

function turnoff(){

$.ajax({url: "writeLED/Off",

success: function(result){

$("#status").html(result);

}

})

}

$(document).ready(function(){

$("#b1").click(function(){

turnon();

});

$("#b2").click(function(){

turnoff();

});

});

</script>

</head>

<body>

<input id="buttonloadchart" type="button" onclick="loadChart()" value="Update graph">

<div id="status"></div>

<div id="chart\_div" style="width:100%"></div>

<div id="table\_div" style="width:100%"></div>

<button id="b1">Turn on</button>

<button id="b2">Turn off</button>

</body>

</html>

# Section 5 Task List

A table listing members names and the parts of the assignment they worked on

|  |  |  |
| --- | --- | --- |
| Name of member | Part of project worked on | Contribution percentage |
| Briant Toh | Node Red | 33% |
| Hyu Tan | Cloud Database | 33% |
| Joseph Chong | Web Interface | 33% |

# Section 6 Any other section you want to add

Delete this portion if you don’t have additional sections

# Section 7 References

References to online materials used

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