**SCHOOL OF COMPUTING (SOC)**

|  |  |
| --- | --- |
| **Date of Submission:** | 21 Feb 2021 |

|  |  |
| --- | --- |
| **Prepared for:** | Ms Dora Chua |

|  |  |
| --- | --- |
| **Class:** | DISM/FT/3A/42 |

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

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 1828063 | Tiang Kar Hoe |
| 1804263 | Ng Wei En, Amos |

**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](#_Toc61178270)

[A. Where we have uploaded our tutorial 2](#_Toc61178271)

[B. What is the application about? 2](#_Toc61178272)

[C. How does the final RPI set-up looks like? 2](#_Toc61178273)

[D. How does the web or mobile application look like? 2](#_Toc61178274)

[E. System architecture of our system 3](#_Toc61178275)

[F. Evidence that we have met basic requirements 3](#_Toc61178276)

[G. Bonus features on top of basic requirements 4](#_Toc61178277)

[A. Quick-start guide (Readme first) 4](#_Toc61178278)

[Section 2 Hardware requirements 5](#_Toc61178279)

[Hardware checklist 5](#_Toc61178280)

[Hardware setup instructions 5](#_Toc61178281)

[Fritzing Diagram 5](#_Toc61178282)

[Section 3 Software Requirements 6](#_Toc61178283)

[Software checklist 6](#_Toc61178284)

[Software setup instructions 6](#_Toc61178285)

[Section 4 Source codes 6](#_Toc61178286)

[server.py 6](#_Toc61178287)

[index.html 6](#_Toc61178288)

[Section 5 Task List 7](#_Toc61178289)

[Section 6 Any other section you want to add 7](#_Toc61178290)

[Section 7 References 7](#_Toc61178291)

# 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/m31HH8IAgfs> |
| **Public tutorial link** |  |

* 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?

Our application keeps track of groceries at home and generates a grocery list if required. The target audience of this application are forgetful people who handles groceries. Our application makes use of a inventory table to track items and based on the threshold set, we are able to generate a grocery list for items below the threshold value so that the user would know what to buy without having to travel back n forth to check.

* 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.

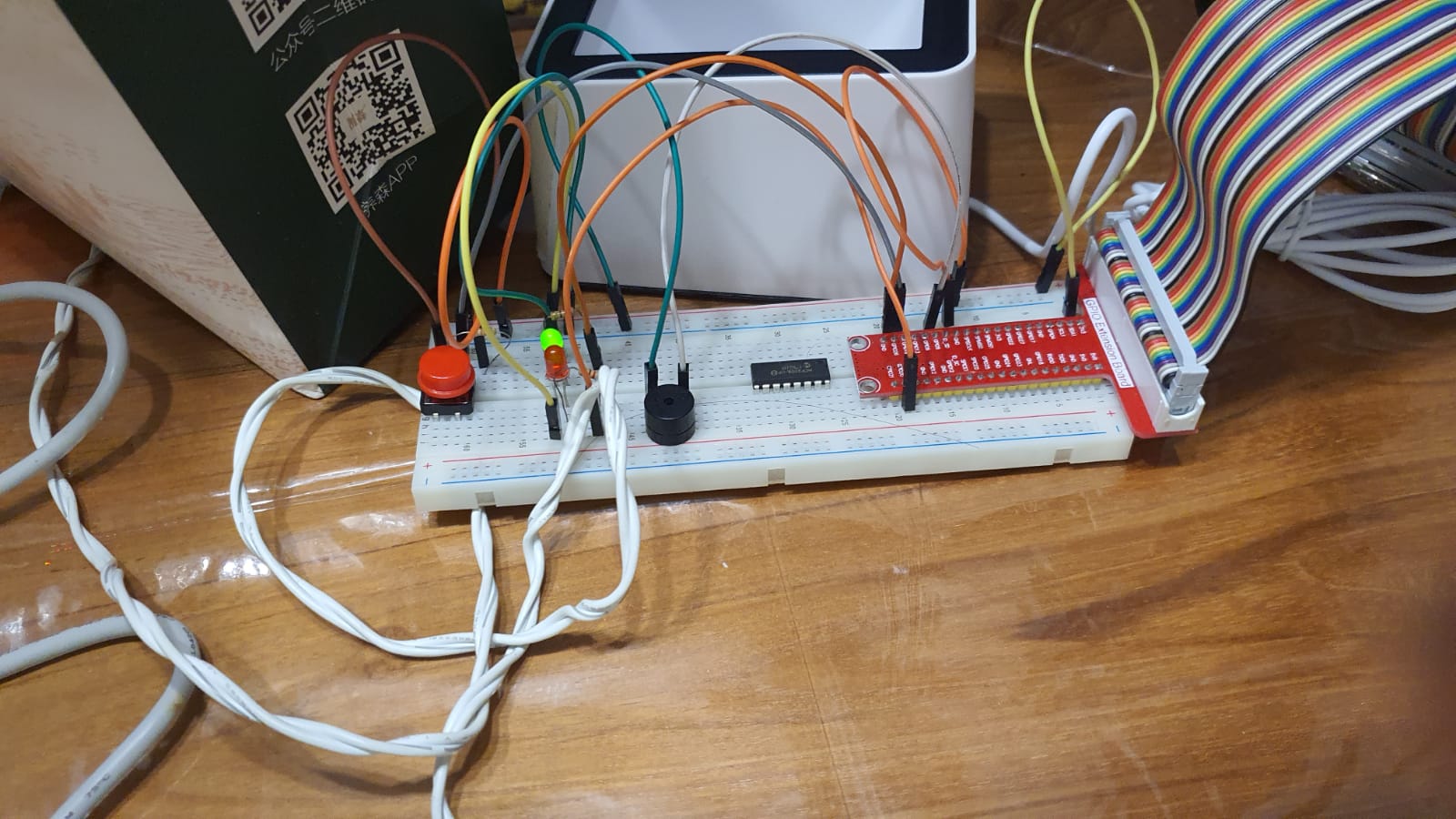


LED

Makeshift Fridge

Arduino + LDR

Figure 1



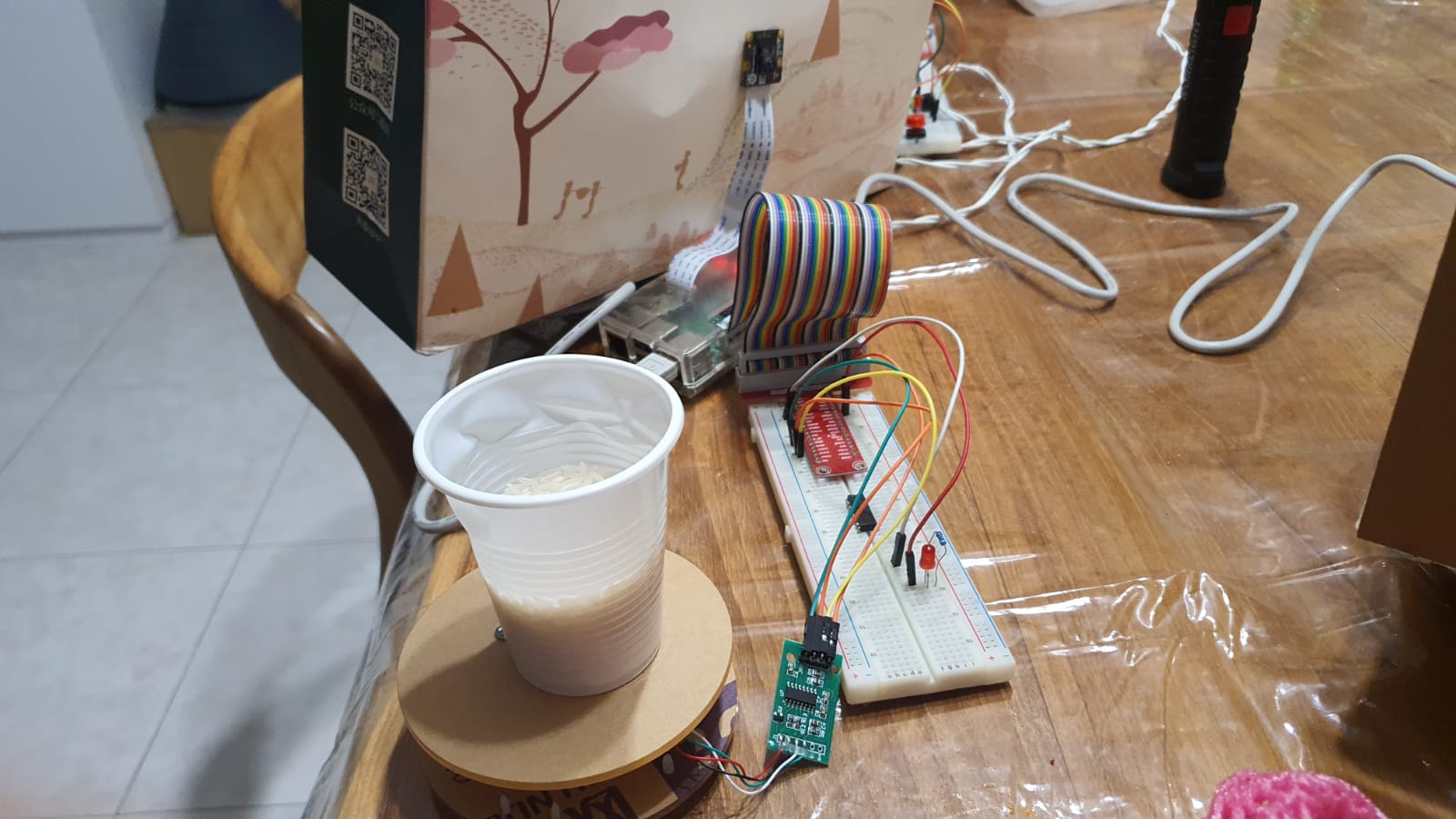
Barcode Scanner

Button

Buzzer

LED

Figure 2



Weight Sensor

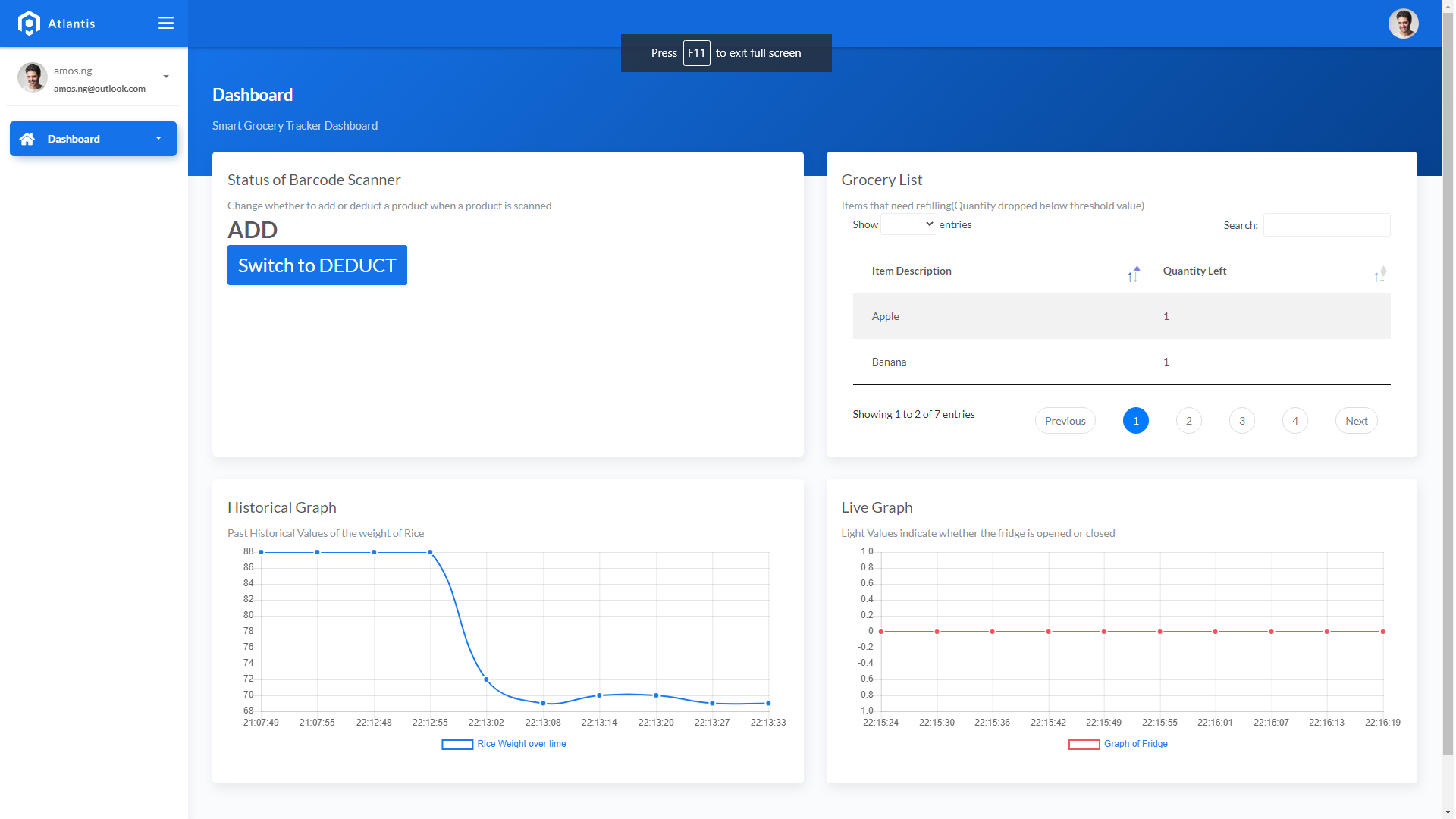
Camera

LED

Figure 3

* 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.



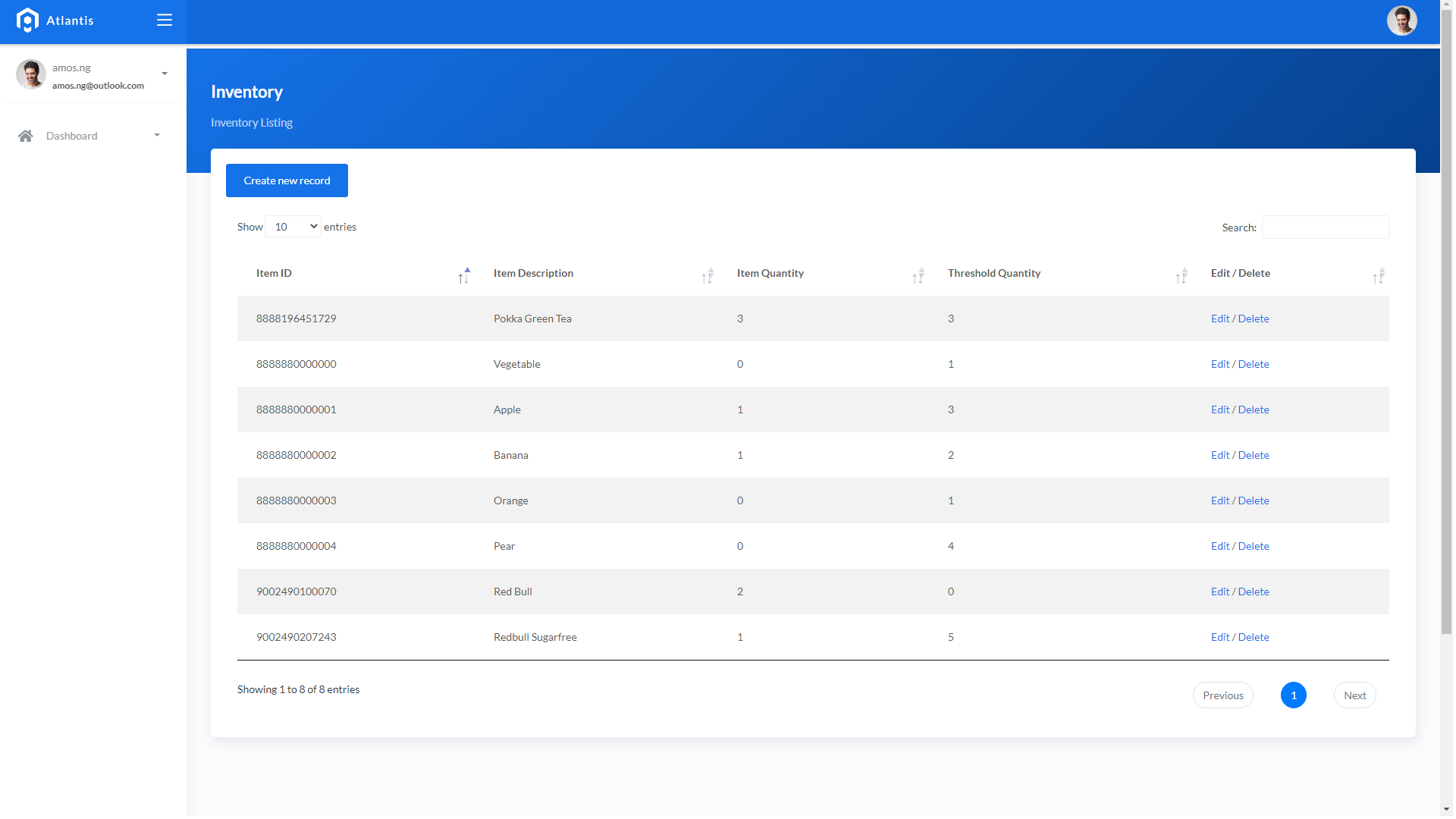
Generated Grocery List

Live Status of Barcode Scanner

Real Time Graph

Historical Graph

Figure 4



Edit/Delete

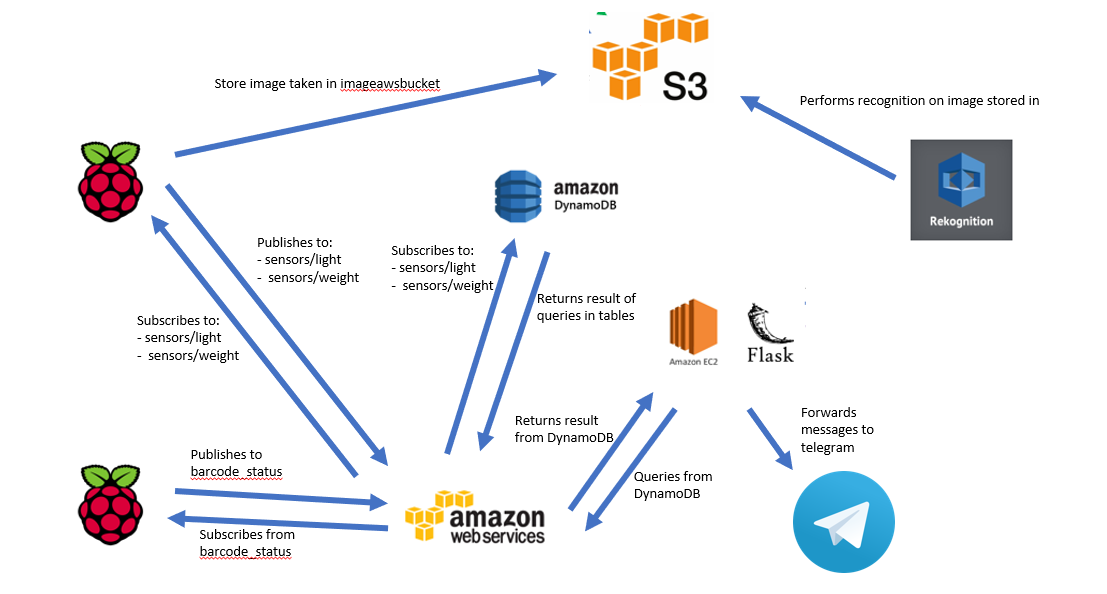
Inventory List

Add record

Figure 5

* 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, Weight, Barcode Scanner and camera. Reference Figure 1, 2, 3 |
| Used MQTT | Our MQTT endpoint --> “sensors/light” & “sensors/weight”  Example of data sent through MQTT : light and weight data |
| Stored data in cloud | Stored light, weight, inventory data in DynamoDB Tables |
| Used cloud service | Use AWS Rekognition, S3, hosted web server on EC2 |
| Provide real-time sensor value / status | Show the real-time value of Light sensor |
| Provide historical sensor value/ status | Show historical vlaue of weight sensor |
| Control actuator | Placed button on webpage and breadboard to control LED |

* 1. Bonus features on top of basic requirements

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

1. Log in system that uses credentials stored in sqlite.
2. Telegram bot that fetches grocery list, control actuator and fetches images from s3
3. Edit DynamoDB Table from web application
   1. Quick-start guide (Readme first)

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

1. Upload “capture\_light.ino” to the arduino to start the LDR.
2. “python weight.py” to start capturing weight
3. “python LDR.py” to start capture light values
4. “python picam\_rekognition.py” to take images and upload to S3 for rekognition
5. “python manage.py makemigrations” and “python manage.py migrate” to create tables
6. “python manage.py runserver” to start the web application. To speficy port number, add 0.0.0.0:<port\_no> at the end
7. “python telegram\_app.py” for Telegram bot

# Section 2 Hardware requirements

Hardware checklist

2 Red LED

1 Green LED

1 button

1 buzzer

1 barcode scanner

1 hx711 weight sensor

1 Light sensor

1 Arduino

2 Raspberry Pi with Camera

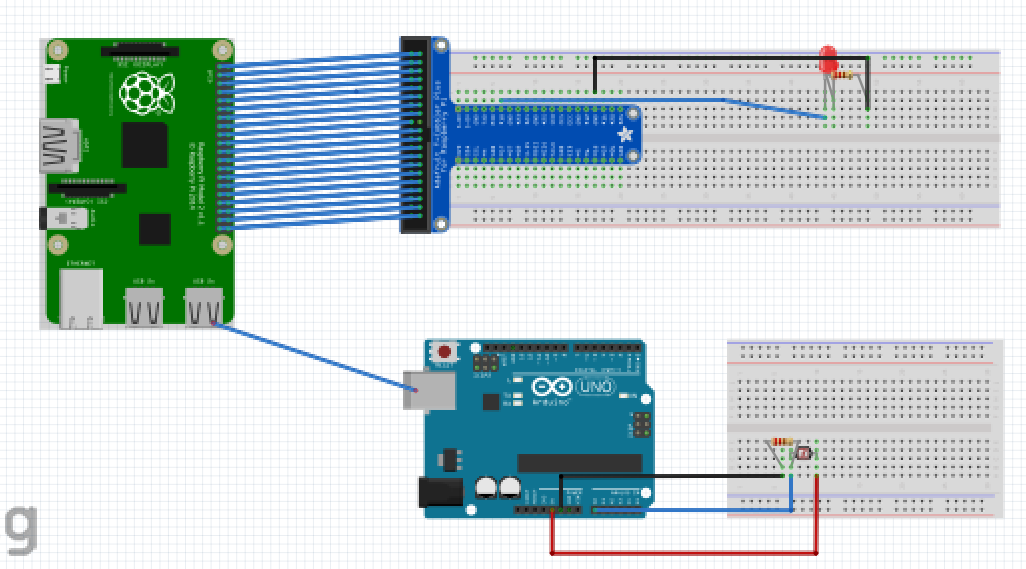
Hardware setup instructions

The hx711 weight sensor cannot be found in the program. To patch, VCC pin to 5V, GND pin to GND, DT pin to GPIO5 and SCK pin to GPIO6.

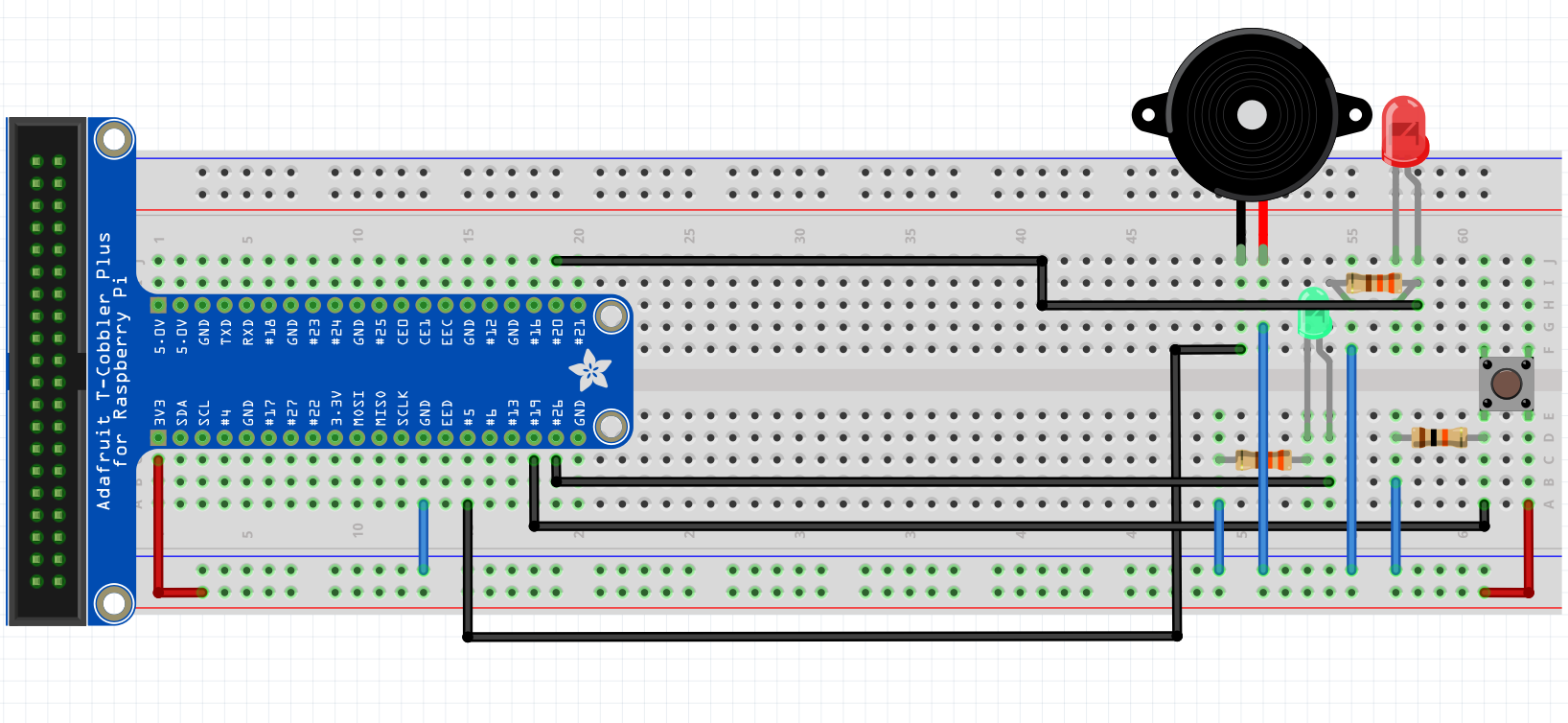
Fritzing Diagram

Paste a Fritzing diagram of your setup here

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



The hx711 weight sensor cannot be found in the program. To patch, VCC pin to 5V, GND pin to GND, DT pin to GPIO5 and SCK pin to GPIO6.



# 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. Asgiref
2. autopep8
3. django==2.2.10
4. pycodestyle
5. pytz
6. sqlparse
7. Unipath
8. dj-database-url
9. python-decouple
10. gunicorn
11. whitenoise
12. boto3
13. simplejson
14. requests

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

Execute “sudo raspi-config”

Go to interface setttings and inteface options -> enable Serial User need to check what interface the barcode is on. it might be on /dev/hidraw0 or /dev/hidraw1 or in some cases /dev/hidraw2 . User need to then go to line 118 of api.py and edit accordingly.

# Section 4 Source codes

All source codes, including Python, HTML files etc

### Inventory.html

{% extends "layouts/base.html" %}

{% block title %} Dashboard {% endblock %}

<!-- Specific Page CSS goes HERE  -->

{% block stylesheets %}

{% endblock stylesheets %}

{% block content %}

    <div class="content">

        <div class="panel-header bg-primary-gradient">

            <div class="page-inner py-5">

                <div class="d-flex align-items-left align-items-md-center flex-column flex-md-row">

                    <div>

                        <h2 class="text-white pb-2 fw-bold">Inventory</h2>

                        <h5 class="text-white op-7 mb-2">Inventory Listing</h5>

                    </div>

                </div>

            </div>

        </div>

        <div class="page-inner mt--5">

            <div class="row mt--2">

                <div class="col">

                    <div class="card full-height">

                        <div class="card-body">

                            <!-- <div class="card-title"Inventory Lisiting</div>

                            <div class="card-category">Daily information about statistics in system</div> -->

                            <button class="editor\_create btn btn-primary" style="display:block">Create new record</button>

                            <div class="d-flex flex-wrap justify-content-around pb-2 pt-4">

                                <table class="table table-striped mt-3" id="datatable">

                                <thead>

                                    <tr>

                                        <th scope="col">Item ID</th>

                                        <th scope="col">Item Description</th>

                                        <th scope="col">Item Quantity</th>

                                        <th scope="col">Threshold Quantity</th>

                                        <th scope="col">Edit / Delete</th>

                                    </tr>

                                </thead>

                            </table>

                            </div>

                        </div>

                    </div>

                </div>

            </div>

            </div>

        </div>

    </div>

<style>

    .dataTables\_wrapper{

        width:100%;

    }

</style>

{% endblock content %}

<!-- Specific Page JS goes HERE  -->

{% block javascripts %}

    <script type="text/javascript">

    var editor;

    $(document).ready(function() {

    editor = new $.fn.dataTable.Editor( {

        "ajax": "../inventory\_update/",

        "table": "#datatable",

        "fields": [ {

                "label": "Item ID:",

                "name": "upc"

            }, {

                "label": "Item Description:",

                "name": "item\_desc"

            }, {

                "label": "Quantity:",

                "name": "qty"

            }, {

                "label": "Threshold Quantity:",

                "name": "threshold"

            }

        ]

    } );

// New record

    $('button.editor\_create').on('click', function (e) {

        e.preventDefault();

       editor.create({

            title: 'Create new record',

            buttons: 'Add'

        }

        );

    } );

    // Edit record

    $('#datatable').on('click', 'a.editor\_edit', function (e) {

        e.preventDefault();

        editor.edit( $(this).closest('tr'), {

            title: 'Edit record',

            buttons: 'Update'

        } );

    } );

    // Delete a record

    $('#datatable').on('click', 'a.editor\_remove', function (e) {

        e.preventDefault();

        editor.remove( $(this).closest('tr'), {

            title: 'Delete record',

            message: 'Are you sure you wish to remove this record?',

            buttons: 'Delete'

        } );

    } );

        $('#datatable').DataTable( {

        "autoWidth": true,

        ajax: "../inventory\_data/",

        columns: [

            { data: "upc" },

            { data: "item\_desc" },

            { data: "qty" },

            { data: "threshold" },

            {

                data: null,

                className: "center",

                defaultContent: '<a href="" class="editor\_edit">Edit</a> / <a href="" class="editor\_remove">Delete</a>'

            }

        ]

    });

});

    </script>

    <script>

        Circles.create({

            id:'circles-1',

            radius:45,

            value:60,

            maxValue:100,

            width:7,

            text: 5,

            colors:['#f1f1f1', '#FF9E27'],

            duration:400,

            wrpClass:'circles-wrp',

            textClass:'circles-text',

            styleWrapper:true,

            styleText:true

        })

        Circles.create({

            id:'circles-2',

            radius:45,

            value:70,

            maxValue:100,

            width:7,

            text: 36,

            colors:['#f1f1f1', '#2BB930'],

            duration:400,

            wrpClass:'circles-wrp',

            textClass:'circles-text',

            styleWrapper:true,

            styleText:true

        })

        Circles.create({

            id:'circles-3',

            radius:45,

            value:40,

            maxValue:100,

            width:7,

            text: 12,

            colors:['#f1f1f1', '#F25961'],

            duration:400,

            wrpClass:'circles-wrp',

            textClass:'circles-text',

            styleWrapper:true,

            styleText:true

        })

        var totalIncomeChart = document.getElementById('totalIncomeChart').getContext('2d');

        var mytotalIncomeChart = new Chart(totalIncomeChart, {

            type: 'bar',

            data: {

                labels: ["S", "M", "T", "W", "T", "F", "S", "S", "M", "T"],

                datasets : [{

                    label: "Total Income",

                    backgroundColor: '#ff9e27',

                    borderColor: 'rgb(23, 125, 255)',

                    data: [6, 4, 9, 5, 4, 6, 4, 3, 8, 10],

                }],

            },

            options: {

                responsive: true,

                maintainAspectRatio: false,

                legend: {

                    display: false,

                },

                scales: {

                    yAxes: [{

                        ticks: {

                            display: false //this will remove only the label

                        },

                        gridLines : {

                            drawBorder: false,

                            display : false

                        }

                    }],

                    xAxes : [ {

                        gridLines : {

                            drawBorder: false,

                            display : false

                        }

                    }]

                },

            }

        });

        $('#lineChart').sparkline([105,103,123,100,95,105,115], {

            type: 'line',

            height: '70',

            width: '100%',

            lineWidth: '2',

            lineColor: '#ffa534',

            fillColor: 'rgba(255, 165, 52, .14)'

        });

    </script>

{% endblock javascripts %}

### index.html

{% extends "layouts/base.html" %}

{% block title %} Dashboard {% endblock %}

<!-- Specific Page CSS goes HERE  -->

{% block stylesheets %}{% endblock stylesheets %}

{% block content %}

    <div class="content">

        <div class="panel-header bg-primary-gradient">

            <div class="page-inner py-5">

                <div class="d-flex align-items-left align-items-md-center flex-column flex-md-row">

                    <div>

                        <h2 class="text-white pb-2 fw-bold">Dashboard</h2>

                        <h5 class="text-white op-7 mb-2">Smart Grocery Tracker Dashboard</h5>

                    </div>

                </div>

            </div>

        </div>

        <div class="page-inner mt--5">

            <div class="row mt--2">

                <div class="col-md-6">

                    <div class="card full-height">

                        <div class="card-body">

                            <div class="card-title">Status of Barcode Scanner</div>

                            <div class="card-category">Change whether to add or deduct a product when a product is scanned</div>

                            <div id="status1" style="font-size:30px;" class="h5 mb-0 font-weight-bold text-gray-800">-</div>

                            <a href="#" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm" style="color:white;font-size:25px;" id="button">Switch to -</a>

                        </div>

                    </div>

                </div>

                <div class="col-md-6">

<div class="card full-height">

                        <div class="card-body">

                            <div class="card-title">Grocery List</div>

                            <div class="card-category">Items that need refilling(Quantity dropped below threshold value)</div>

                            <table class="table table-striped mt-3" id="grocerytable">

                                <thead>

                                    <tr>

                                        <th scope="col">Item Description</th>

                                        <th scope="col">Quantity Left</th>

                                    </tr>

                                </thead>

                            </table>

                        </div>

                    </div>

                </div>

            </div>

            <div class="row">

                <div class="col-md-6">

                    <div class="card full-height">

                        <div class="card-body">

                            <div class="card-title">Historical Graph</div>

<div class="card-category">Past Historical Values of the weight of Rice</div>

                                <div class="chart-container">

                                    <canvas id="historicalgraph"></canvas>

                                </div>

                        </div>

                    </div>

                </div>

                <div class="col-md-6">

                    <div class="card full-height">

                        <div class="card-body">

                            <div class="card-title">Live Graph</div>

<div class="card-category">Light Values indicate whether the fridge is opened or closed</div>

                                <div class="chart-container">

                                    <canvas id="livegraph"></canvas>

                                </div>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </div>

{% endblock content %}

<!-- Specific Page JS goes HERE  -->

{% block javascripts %}

    <script>

        var historicalGraph = document.getElementById("historicalgraph").getContext('2d');

        var liveGraph = document.getElementById("livegraph").getContext('2d');

        $(document).ready(function(){

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

            $.get("/toggle\_barcode\_status/");

            checkState();

        });

            $("#grocerytable").DataTable({

            "pageLength":2,

            "autoWidth": true,

            ajax: "../get\_grocery\_list/",

            columns: [

            {data:"item\_desc"},

            {data:"qty"}

            ]

            });

            plotHistoricalGraph();

            plotLiveGraph();

            setInterval(function(){

            checkState();

            },3000);

            setInterval(function(){

                plotLiveGraph()

            },6000);

        });

function checkState(){

            $.get("/get\_barcode\_status/", function(data){

            console.log("'"+data+"'")

                if (data == "0"){

                    $("#status1").html("DEDUCT")

                    $("#button").html("Switch to ADD")

                } else {

                    $("#status1").html("ADD")

                    $("#button").html("Switch to DEDUCT")

                }

            });

        }

    function plotLiveGraph(){

        $.get('/live\_graph/', function(data){

                console.log(data)

                var data\_labels = data.labels

                console.log(data\_labels)

                var data\_data = data.data

                console.log(data\_data)

                var myLiveGraph = new Chart(liveGraph, {

            type: 'line',

            data: {

                labels: data\_labels,

                datasets:[{

                    label: "Graph of Fridge",

                    borderColor: "#f3545c",

                    pointBorderColor: "#FFF",

                    pointBackgroundColor: "#f3545c",

                    pointBorderWidth: 2,

                    pointHoverRadius: 4,

                    pointHoverBorderWidth: 1,

                    pointRadius: 4,

                    backgroundColor: 'transparent',

                    fill: true,

                    borderWidth: 2,

                    data: data\_data

                }]

            },

            options : {

                responsive: true,

                maintainAspectRatio: false,

                legend: {

                    position: 'bottom',

                    labels : {

                        padding: 10,

                        fontColor: '#1d7af3',

                    }

                },

                tooltips: {

                    bodySpacing: 4,

                    mode:"nearest",

                    intersect: 0,

                    position:"nearest",

                    xPadding:10,

                    yPadding:10,

                    caretPadding:10

                },

                layout:{

                    padding:{left:15,right:15,top:15,bottom:15}

                }

            }

        });

            })

    }

    function plotHistoricalGraph(){

        $.get('/historical\_graph/', function(data){

                console.log(data)

                var data\_labels = data.labels

                console.log(data\_labels)

                var data\_data = data.data

                console.log(data\_data)

                var myHistoricalGraph = new Chart(historicalGraph, {

            type: 'line',

            data: {

                labels: data\_labels,

                datasets:[{

                    label: "Rice Weight over time",

                    borderColor: "#1d7af3",

                    pointBorderColor: "#FFF",

                    pointBackgroundColor: "#1d7af3",

                    pointBorderWidth: 2,

                    pointHoverRadius: 4,

                    pointHoverBorderWidth: 1,

                    pointRadius: 4,

                    backgroundColor: 'transparent',

                    fill: true,

                    borderWidth: 2,

                    data: data\_data

                }]

            },

            options : {

                responsive: true,

                maintainAspectRatio: false,

                legend: {

                    position: 'bottom',

                    labels : {

                        padding: 10,

                        fontColor: '#1d7af3',

                    }

                },

                tooltips: {

                    bodySpacing: 4,

                    mode:"nearest",

                    intersect: 0,

                    position:"nearest",

                    xPadding:10,

                    yPadding:10,

                    caretPadding:10

                },

                layout:{

                    padding:{left:15,right:15,top:15,bottom:15}

                }

            }

        });

            })

    }

    </script>

{% endblock javascripts %}

### urls.py

# -\*- encoding: utf-8 -\*-

"""

Copyright (c) 2019 - present AppSeed.us

"""

from django.urls import path

from .views import \*

from django.contrib.auth.views import LogoutView

urlpatterns = [

    path('login/', login\_view, name="login"),

    path('register/', register\_user, name="register"),

    path("logout/", LogoutView.as\_view(), name="logout"),

    path("inventory/", inventory, name='inventory'),

    path("inventory\_data/", inventory\_data, name='inventory\_data'),

path("inventory\_update/", inventory\_update, name='inventory\_update'),

    path("historical\_graph/", historical\_graph, name='historical\_graph'),

path("live\_graph/", live\_graph, name='live\_graph'),

path("get\_barcode\_status/", get\_barcode\_status, name='get\_barcode\_status'),

path("get\_grocery\_list/", get\_grocery\_list, name='get\_grocery\_list'),

path("toggle\_barcode\_status/", toggle\_barcode\_status, name='toggle\_barcode\_status'),

path("check\_upc/", check\_upc, name='check\_upc'),

    path("product/", product, name='product')

]

### views.py

# -\*- encoding: utf-8 -\*-

"""

Copyright (c) 2019 - present AppSeed.us

"""

import boto3

from boto3.dynamodb.conditions import Key

from botocore.exceptions import ClientError

from django.contrib.auth import authenticate, login

from django.http import JsonResponse, HttpResponse

# Create your views here.

from django.shortcuts import render, redirect

import simplejson

from django.views.decorators.csrf import csrf\_exempt

from datetime import datetime

from .forms import LoginForm, SignUpForm

# Credits for the BotHandler Class

# https://github.com/magnitopic/YouTubeCode

import requests

class BotHandler:

    token = '1620569172:AAEoFuZf02nunzAHPrUhyt50lGexusDrFIE'  # Telegram Bot Token

    def \_\_init\_\_(self):

        self.api\_url = "https://api.telegram.org/bot{}/".format(self.token)

    # url = "https://api.telegram.org/bot<token>/"

    def get\_updates(self, offset=0, timeout=30):

        method = 'getUpdates'

        params = {'timeout': timeout, 'offset': offset}

        resp = requests.get(self.api\_url + method, params)

        result\_json = resp.json()['result']

        return result\_json

    def send\_message(self, chat\_id, text):

        params = {'chat\_id': chat\_id, 'text': text, 'parse\_mode': 'HTML'}

        method = 'sendMessage'

        resp = requests.post(self.api\_url + method, params)

        return resp

    def send\_messages(self, chat\_id, text):

        for x in chat\_id:

            params = {'chat\_id': x, 'text': text, 'parse\_mode': 'HTML'}

            method = 'sendMessage'

            resp = requests.post(self.api\_url + method, params)

        return True

    def get\_first\_update(self):

        get\_result = self.get\_updates()

        if len(get\_result) > 0:

            last\_update = get\_result[0]

        else:

            last\_update = None

        return last\_update

class Dynamodb():

    def \_\_init\_\_(self):

        self.dynamodb = boto3.resource('dynamodb')

    def get\_all(self, tablename):

        table = self.dynamodb.Table(tablename)

        response = table.scan()

        if 'Items' in response:

            return table.scan()['Items']

        else:

            return False

    def get\_all\_filter(self,tablename,device\_name):

        table = self.dynamodb.Table(tablename)

        startdate = "2021-02"

        response = table.query(

            KeyConditionExpression=Key('deviceid').eq(device\_name) & Key('datetimeid').begins\_with(startdate),

            ScanIndexForward=False

        )

        items = response['Items']

        data = items[:10]

        return data[::-1]

    def get(self,tablename,key):

        try:

            table = self.dynamodb.Table(tablename)

            response = table.get\_item(Key=key)

            if len(response)>0:

                if "Item" in response:

                    return response['Item']

                else:

                    return False

            else:

                return False

        except ClientError as e:

            print("ClientError occurred!")

            print(e)

            print("tablename:{} provided key:{}".format(tablename,key))

            return False

    def add(self,tablename,values):

        try:

            table = self.dynamodb.Table(tablename)

            table.put\_item(Item=values)

            return True

        except ClientError:

            print("Error! ValidationException occurred, could not add to DynamoDB!")

            return False

    def update(self,tablename,key,upd\_expression,values=None):

        try:

            table = self.dynamodb.Table(tablename)

            if values is None:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression)

            else:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression,ExpressionAttributeValues=values)

            return resp

        except ClientError as e:

            print("Error! ValidationException occurred, could not update DynamoDB!")

            print(e)

            return False

    def delete(self,tablename,key):

        try:

            table = self.dynamodb.Table(tablename)

            resp = table.delete\_item(Key=key)

            return resp

        except ClientError as e:

            print("Error! ValidationException occurred, could not update DynamoDB!")

            print(e)

            return False

dynamodb = Dynamodb()

IOTbot = BotHandler()

@csrf\_exempt

def check\_upc(request):

    #check on the upc

    print(request.POST['upc'])

    upc = int(request.POST['upc'])

    #fetch inventory

    inventory = dynamodb.get("inventory", {"upc": upc})

    #get qty

    qty = inventory['qty']

    #fetch upc\_product

    upc\_product = dynamodb.get("upc\_product", {"upc": upc})

    item\_desc = upc\_product['item\_desc']

    threshold = upc\_product['threshold']

    if qty <= threshold:

        #fetch subscriber list

        subscribers = dynamodb.get\_all("subscribe")

        if subscribers:

            for x in subscribers:

                if x['s\_status'] == 1:

                    #subscriber

                    chat\_id = x['chat\_id']

                    message = "The following product is low on quantity, please refill! Item Description: {} Quantity Left: {}".format(item\_desc,qty)

                    IOTbot.send\_message(chat\_id,message)

    return JsonResponse(True,safe=False)

def get\_grocery\_list(request):

    #fetch from inventory

    #compare with threshold value in upc\_product

    inventory = dynamodb.get\_all("inventory")

    list = []

    for x in inventory:

        #fetch the threshold value

        upc = x['upc']

        qty = int(x['qty'])

        upc\_product = dynamodb.get("upc\_product", {"upc": upc})

        threshold\_value = int(upc\_product['threshold'])

        item\_desc = upc\_product['item\_desc']

        if qty <= threshold\_value:

            item = {"item\_desc":item\_desc,"qty":qty}

            list.append(item)

    json = {"data":list}

    return JsonResponse(json,safe=False)

def get\_barcode\_status(request):

    #False - Deduct Mode

    #True - Add Mode

    Key = {'id': 1}

    result = dynamodb.get('barcode\_status', Key)

    if result:

        mode = result['b\_mode']

        if mode == 0:

            return JsonResponse("0", safe=False)

        else:

            return JsonResponse("1", safe=False)

    else:

        return JsonResponse("-1", safe=False)

def toggle\_barcode\_status(request):

    Key = {'id': 1}

    result = dynamodb.get('barcode\_status', Key)

    if result:

        mode = result['b\_mode']

        if mode == 0:

            mode = 1

        else:

            mode = 0

        print(mode)

        ExpressionAttributeValues = {':val1': mode}

        Key = {'id': 1}

        UpdateExpression = 'SET b\_mode = :val1'

        resp = dynamodb.update('barcode\_status', Key, UpdateExpression, ExpressionAttributeValues)

        if resp:

            return JsonResponse(True, safe=False)

        else:

            return JsonResponse(False, safe=False)

def live\_graph(request):

    results = dynamodb.get\_all\_filter("device\_values", "deviceid\_light")

    labels = []

    data = []

    for x in results:

        y = int(x["value"])

        if y > 600:

            y = 1

        else:

            y = 0

        z = x['datetimeid']

        z = z.split(".")[0]

        z = datetime.strptime(z, "%Y-%m-%dT%H:%M:%S")

        time = "{}:{}:{}".format(str(z.hour).zfill(2), str(z.minute).zfill(2), str(z.second).zfill(2))

        labels.append(time)

        data.append(y)

        # convert datetime

    json = {'labels': labels, 'data': data}

    return JsonResponse(json, safe=False)

def historical\_graph(request):

    results = dynamodb.get\_all\_filter("device\_values","deviceid\_weight")

    labels = []

    data = []

    for x in results:

        y = int(x["value"])

        z = x['datetimeid']

        z = z.split(".")[0]

        z = datetime.strptime(z,"%Y-%m-%dT%H:%M:%S")

        time = "{}:{}:{}".format(str(z.hour).zfill(2),str(z.minute).zfill(2),str(z.second).zfill(2))

        labels.append(time)

        data.append(y)

        #convert datetime

    json = {'labels':labels,'data':data}

    return JsonResponse(json, safe=False)

@csrf\_exempt

def inventory\_update(request):

    data = request.POST

    item\_data = {}

    action = ""

    for x in data:

        if "upc" in x:

            item\_data['DT\_RowId'] = data[x]

            item\_data['upc'] = int(data[x])

        elif "item\_desc" in x:

            item\_data['item\_desc'] = data[x]

        elif "qty" in x:

            item\_data['qty'] = int(data[x])

        elif "threshold" in x:

            item\_data['threshold'] = int(data[x])

        elif "action" in x:

            action = data[x]

    #action - edit / remove / create

    key = {"upc": item\_data['upc']}

    if action == "edit":

        #2 tables upc\_product and inventory

        #delete and add

        response\_1 = dynamodb.delete("inventory", key)

        response\_2 = dynamodb.delete("upc\_product", key)

        values = {"upc": item\_data["upc"], "item\_desc": item\_data['item\_desc'], "threshold": item\_data['threshold']}

        response\_3 = dynamodb.add("upc\_product", values)

        values = {"upc": item\_data['upc'], "qty": item\_data['qty']}

        response\_4 = dynamodb.add("inventory", values)

        if response\_1 and response\_2 and response\_3 and response\_4:

            x = []

            x.append(item\_data)

            data = {"data": x}

            return JsonResponse(data, safe=False)

        else:

            return JsonResponse(False)

    elif action == "remove":

        #2 tables upc\_product and inventory

        response\_1 = dynamodb.delete("inventory",key)

        response\_2 = dynamodb.delete("upc\_product",key)

        if response\_1 and response\_2:

            data = {"data":[]}

            return JsonResponse(data,safe=False)

        else:

            return JsonResponse(False)

    elif action == "create":

        #upc\_product

        values = {"upc": item\_data["upc"], "item\_desc": item\_data['item\_desc'], "threshold": item\_data['threshold']}

        response\_1 = dynamodb.add("upc\_product",values)

        values = {"upc": item\_data['upc'], "qty": item\_data['qty']}

        response\_2 = dynamodb.add("inventory",values)

        if response\_1 and response\_2:

            x = []

            x.append(item\_data)

            data = {"data": x}

            return JsonResponse(data, safe=False)

        else:

            return JsonResponse(False)

    #for x in data:

        #upc = x[0]

        #item\_desc = x[1]

        #item\_qty = x[2]

        #item\_threshold = x[3]

        #item\_action = x[4]

    return HttpResponse(data)

def inventory\_data(request):

    # fetch inventory list

    response = dynamodb.get\_all("inventory")

    # response\_2 = dynamodb.get\_all("upc\_product")

    i = 1

    for x in response:

        # find corresponding upc\_product entry

        upc = x['upc']

        result = dynamodb.get("upc\_product", {'upc': upc})

        if result:

            x['upc'] = int(x['upc'])

            x['qty'] = int(x['qty'])

            x['item\_desc'] = result['item\_desc']

            x['threshold'] = int(result['threshold'])

            x['DT\_RowId'] = upc

    if response:

        #json = simplejson.dumps(response, use\_decimal=True)

        data = {"data":response}

        return JsonResponse(data, safe=False)

    else:

        return JsonResponse(False)

def inventory(request):

    return render(request, 'inventory.html')

def product(request):

    pass

def login\_view(request):

    form = LoginForm(request.POST or None)

    msg = None

    if request.method == "POST":

        if form.is\_valid():

            username = form.cleaned\_data.get("username")

            password = form.cleaned\_data.get("password")

            user = authenticate(username=username, password=password)

            if user is not None:

                login(request, user)

                return redirect("/")

            else:

                msg = 'Invalid credentials'

        else:

            msg = 'Error validating the form'

    return render(request, "accounts/login.html", {"form": form, "msg" : msg})

def register\_user(request):

    msg     = None

    success = False

    if request.method == "POST":

        form = SignUpForm(request.POST)

        if form.is\_valid():

            form.save()

            username = form.cleaned\_data.get("username")

            raw\_password = form.cleaned\_data.get("password1")

            user = authenticate(username=username, password=raw\_password)

            msg     = 'User created - please <a href="/login">login</a>.'

            success = True

            #return redirect("/login/")

        else:

            msg = 'Form is not valid'

    else:

        form = SignUpForm()

    return render(request, "accounts/register.html", {"form": form, "msg" : msg, "success" : success })

### api.py

#!/usr/bin/python

import sys

import requests

import json

import boto3

from botocore.exceptions import ClientError

api\_key = "3C84072AE757BA0B54F8FDA0BF44A64F" #https://upcdatabase.org/

class Dynamodb():

    def \_\_init\_\_(self):

        self.dynamodb = boto3.resource('dynamodb')

    def get(self,tablename,key):

        try:

            table = self.dynamodb.Table(tablename)

            response = table.get\_item(Key=key)

            if len(response)>0:

                if "Item" in response:

                    return response['Item']

                else:

                    return False

            else:

                return False

        except ClientError as e:

            print("ClientError occurred!")

            print(e)

            print("tablename:{} provided key:{}".format(tablename,key))

            return False

    def add(self,tablename,values):

        try:

            table = self.dynamodb.Table(tablename)

            table.put\_item(Item=values)

            return True

        except ClientError:

            print("Error! ValidationException occurred, could not add to DynamoDB!")

            return False

    def update(self,tablename,key,upd\_expression,values=None):

        try:

            table = self.dynamodb.Table(tablename)

            if values is None:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression)

            else:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression,ExpressionAttributeValues=values)

            return resp

        except ClientError as e:

            print("Error! ValidationException occurred, could not update DynamoDB!")

            print(e)

            return False

dynamodb = Dynamodb()

def add\_deduct\_item(upc,mode):

    #check if inventory item exists:

    key = {'upc':upc}

    response = dynamodb.get("inventory",key)

    if response:

        #in database

        #check mode

        if mode == 0:

            #deduct mode

            upd\_expression = "SET qty = qty - :val"

        else:

            #add mode

            upd\_expression = "SET qty = qty + :val"

        key = {'upc':upc}

        values = {':val':1}

        response = dynamodb.update("inventory",key,upd\_expression,values)

        return response

    else:

        #not in database

        #create entry

        #Note to self:This part should always be an add, no deduct.

        #Function should only trigger if add\_if\_not\_exist\_item is true.

        values = {"upc":upc,"qty":1}

        response = dynamodb.add("inventory",values)

        return response

def add\_if\_not\_exist\_item(upc):

    key = {'upc':upc}

    response = dynamodb.get("upc\_product",key)

    if response:

        #in database

        return True

    else:

        #find in the UPC Database

        lookup = UPC\_lookup(upc)

        if lookup:

            #Found

            item\_desc = "Unknown"

            if lookup['title'].strip() != "":

                print(len(lookup['title'].strip()))

                print(lookup['title'])

                item\_desc = lookup['title']

            elif lookup['description'].strip() != "":

                item\_desc = lookup['description']

            values = {"upc":upc,"item\_desc":item\_desc,"threshold":0}

            dynamodb.add("upc\_product",values)

            values = {"upc":upc,"qty":0}

            dynamodb.add("inventory",values)

            return True

        else:

            return False

def barcode\_reader():

    """Barcode code obtained from 'brechmos'

    https://www.raspberrypi.org/forums/viewtopic.php?f=45&t=55100"""

    hid = {4: 'a', 5: 'b', 6: 'c', 7: 'd', 8: 'e', 9: 'f', 10: 'g', 11: 'h', 12: 'i', 13: 'j', 14: 'k', 15: 'l', 16: 'm',

           17: 'n', 18: 'o', 19: 'p', 20: 'q', 21: 'r', 22: 's', 23: 't', 24: 'u', 25: 'v', 26: 'w', 27: 'x', 28: 'y',

           29: 'z', 30: '1', 31: '2', 32: '3', 33: '4', 34: '5', 35: '6', 36: '7', 37: '8', 38: '9', 39: '0', 44: ' ',

           45: '-', 46: '=', 47: '[', 48: ']', 49: '\\', 51: ';', 52: '\'', 53: '~', 54: ',', 55: '.', 56: '/'}

    hid2 = {4: 'A', 5: 'B', 6: 'C', 7: 'D', 8: 'E', 9: 'F', 10: 'G', 11: 'H', 12: 'I', 13: 'J', 14: 'K', 15: 'L', 16: 'M',

            17: 'N', 18: 'O', 19: 'P', 20: 'Q', 21: 'R', 22: 'S', 23: 'T', 24: 'U', 25: 'V', 26: 'W', 27: 'X', 28: 'Y',

            29: 'Z', 30: '!', 31: '@', 32: '#', 33: '$', 34: '%', 35: '^', 36: '&', 37: '\*', 38: '(', 39: ')', 44: ' ',

            45: '\_', 46: '+', 47: '{', 48: '}', 49: '|', 51: ':', 52: '"', 53: '~', 54: '<', 55: '>', 56: '?'}

    fp = open('/dev/hidraw1', 'rb')

    ss = ""

    shift = False

    done = False

    while not done:

        ## Get the character from the HID

        buffer = fp.read(8)

        for c in buffer:

            if int(c) > 0:

                ##  40 is carriage return which signifies

                ##  we are done looking for characters

                if int(c) == 40:

                    done = True

                    break;

                ##  If we are shifted then we have to

                ##  use the hid2 characters.

                if shift:

                    ## If it is a '2' then it is the shift key

                    if int(c) == 2:

                        shift = True

                    ## if not a 2 then lookup the mapping

                    else:

                        ss += hid2[int(c)]

                        shift = False

                ##  If we are not shifted then use

                ##  the hid characters

                else:

                    ## If it is a '2' then it is the shift key

                    if int(c) == 2:

                        shift = True

                    ## if not a 2 then lookup the mapping

                    else:

                        ss += hid[int(c)]

    return ss

def UPC\_lookup(upc):

    '''V3 API'''

    url = "https://api.upcdatabase.org/product/%s?apikey=%s" % (upc, api\_key)

    headers = {

        'cache-control': "no-cache",

    }

    response = requests.request("GET", url, headers=headers)

    if response.status\_code == 200:

        resp = response.json()

        print(resp)

        if resp['success']:

            return resp

        else:

            return False

    else:

        return False

    #print("-----" \* 5)

    #print(upc)

    #print(json.dumps(response.json(), indent=2))

    #print("-----" \* 5 + "\n")

### barcode\_scanner.py

#!/usr/bin/python

import sys

import requests

import json

import boto3

from botocore.exceptions import ClientError

api\_key = "3C84072AE757BA0B54F8FDA0BF44A64F" #https://upcdatabase.org/

class Dynamodb():

    def \_\_init\_\_(self):

        self.dynamodb = boto3.resource('dynamodb')

    def get(self,tablename,key):

        try:

            table = self.dynamodb.Table(tablename)

            response = table.get\_item(Key=key)

            if len(response)>0:

                if "Item" in response:

                    return response['Item']

                else:

                    return False

            else:

                return False

        except ClientError as e:

            print("ClientError occurred!")

            print(e)

            print("tablename:{} provided key:{}".format(tablename,key))

            return False

    def add(self,tablename,values):

        try:

            table = self.dynamodb.Table(tablename)

            table.put\_item(Item=values)

            return True

        except ClientError:

            print("Error! ValidationException occurred, could not add to DynamoDB!")

            return False

    def update(self,tablename,key,upd\_expression,values=None):

        try:

            table = self.dynamodb.Table(tablename)

            if values is None:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression)

            else:

                resp = table.update\_item(Key=key,UpdateExpression=upd\_expression,ExpressionAttributeValues=values)

            return resp

        except ClientError as e:

            print("Error! ValidationException occurred, could not update DynamoDB!")

            print(e)

            return False

dynamodb = Dynamodb()

def add\_deduct\_item(upc,mode):

    #check if inventory item exists:

    key = {'upc':upc}

    response = dynamodb.get("inventory",key)

    if response:

        #in database

        #check mode

        if mode == 0:

            #deduct mode

            upd\_expression = "SET qty = qty - :val"

        else:

            #add mode

            upd\_expression = "SET qty = qty + :val"

        key = {'upc':upc}

        values = {':val':1}

        response = dynamodb.update("inventory",key,upd\_expression,values)

        return response

    else:

        #not in database

        #create entry

        #Note to self:This part should always be an add, no deduct.

        #Function should only trigger if add\_if\_not\_exist\_item is true.

        values = {"upc":upc,"qty":1}

        response = dynamodb.add("inventory",values)

        return response

def add\_if\_not\_exist\_item(upc):

    key = {'upc':upc}

    response = dynamodb.get("upc\_product",key)

    if response:

        #in database

        return True

    else:

        #find in the UPC Database

        lookup = UPC\_lookup(upc)

        if lookup:

            #Found

            item\_desc = "Unknown"

            if lookup['title'].strip() != "":

                print(len(lookup['title'].strip()))

                print(lookup['title'])

                item\_desc = lookup['title']

            elif lookup['description'].strip() != "":

                item\_desc = lookup['description']

            values = {"upc":upc,"item\_desc":item\_desc,"threshold":0}

            dynamodb.add("upc\_product",values)

            values = {"upc":upc,"qty":0}

            dynamodb.add("inventory",values)

            return True

        else:

            return False

def barcode\_reader():

    """Barcode code obtained from 'brechmos'

    https://www.raspberrypi.org/forums/viewtopic.php?f=45&t=55100"""

    hid = {4: 'a', 5: 'b', 6: 'c', 7: 'd', 8: 'e', 9: 'f', 10: 'g', 11: 'h', 12: 'i', 13: 'j', 14: 'k', 15: 'l', 16: 'm',

           17: 'n', 18: 'o', 19: 'p', 20: 'q', 21: 'r', 22: 's', 23: 't', 24: 'u', 25: 'v', 26: 'w', 27: 'x', 28: 'y',

           29: 'z', 30: '1', 31: '2', 32: '3', 33: '4', 34: '5', 35: '6', 36: '7', 37: '8', 38: '9', 39: '0', 44: ' ',

           45: '-', 46: '=', 47: '[', 48: ']', 49: '\\', 51: ';', 52: '\'', 53: '~', 54: ',', 55: '.', 56: '/'}

    hid2 = {4: 'A', 5: 'B', 6: 'C', 7: 'D', 8: 'E', 9: 'F', 10: 'G', 11: 'H', 12: 'I', 13: 'J', 14: 'K', 15: 'L', 16: 'M',

            17: 'N', 18: 'O', 19: 'P', 20: 'Q', 21: 'R', 22: 'S', 23: 'T', 24: 'U', 25: 'V', 26: 'W', 27: 'X', 28: 'Y',

            29: 'Z', 30: '!', 31: '@', 32: '#', 33: '$', 34: '%', 35: '^', 36: '&', 37: '\*', 38: '(', 39: ')', 44: ' ',

            45: '\_', 46: '+', 47: '{', 48: '}', 49: '|', 51: ':', 52: '"', 53: '~', 54: '<', 55: '>', 56: '?'}

    fp = open('/dev/hidraw1', 'rb')

    ss = ""

    shift = False

    done = False

    while not done:

        ## Get the character from the HID

        buffer = fp.read(8)

        for c in buffer:

            if int(c) > 0:

                ##  40 is carriage return which signifies

                ##  we are done looking for characters

                if int(c) == 40:

                    done = True

                    break;

                ##  If we are shifted then we have to

                ##  use the hid2 characters.

                if shift:

                    ## If it is a '2' then it is the shift key

                    if int(c) == 2:

                        shift = True

                    ## if not a 2 then lookup the mapping

                    else:

                        ss += hid2[int(c)]

                        shift = False

                ##  If we are not shifted then use

                ##  the hid characters

                else:

                    ## If it is a '2' then it is the shift key

                    if int(c) == 2:

                        shift = True

                    ## if not a 2 then lookup the mapping

                    else:

                        ss += hid[int(c)]

    return ss

def UPC\_lookup(upc):

    '''V3 API'''

    url = "https://api.upcdatabase.org/product/%s?apikey=%s" % (upc, api\_key)

    headers = {

        'cache-control': "no-cache",

    }

    response = requests.request("GET", url, headers=headers)

    if response.status\_code == 200:

        resp = response.json()

        print(resp)

        if resp['success']:

            return resp

        else:

            return False

    else:

        return False

    #print("-----" \* 5)

    #print(upc)

    #print(json.dumps(response.json(), indent=2))

    #print("-----" \* 5 + "\n")

### button.py

from gpiozero import LED, Button

from signal import pause

import sys

from time import sleep

from api import Dynamodb

dynamodb = Dynamodb()

led\_green = LED(26)

led\_red = LED(20)

button = Button(19,pull\_up=False)

def toggle\_state(led):

    #to pass in led\_green

    if led.is\_lit:

        #gre

        ExpressionAttributeValues={':val1':1}

    else:

        ExpressionAttributeValues={':val1':0}

    Key = {'id': 1}

    UpdateExpression = 'SET b\_mode = :val1'

    dynamodb.update('barcode\_status',Key,UpdateExpression,ExpressionAttributeValues)

while True:

    try:

        #check current mode in database

        #0 - deduct mode

        #1 - add mode

        Key = {'id':1}

        result = dynamodb.get('barcode\_status',Key)

        if result:

            mode = result['b\_mode']

            if mode == 0:

                led\_red.on()

                led\_green.off()

            else:

                led\_green.on()

                led\_red.off()

        button.wait\_for\_press(timeout=5)

        if button.is\_pressed:

            print("Button pressed!")

            if led\_green.is\_lit:

                led\_green.off()

                led\_red.on()

            else:

                led\_green.on()

                led\_red.off()

            toggle\_state(led\_green)

            sleep(1)

    except KeyboardInterrupt:

        led\_green.off()

        led\_red.off()

        print("Closing")

        break

### emulated\_hx711.py

import time

import random

import math

import threading

class HX711:

    def \_\_init\_\_(self, dout, pd\_sck, gain=128):

        self.PD\_SCK = pd\_sck

        self.DOUT = dout

        # Last time we've been read.

        self.lastReadTime = time.time()

        self.sampleRateHz = 80.0

        self.resetTimeStamp = time.time()

        self.sampleCount = 0

        self.simulateTare = False

        # Mutex for reading from the HX711, in case multiple threads in client

        # software try to access get values from the class at the same time.

        self.readLock = threading.Lock()

        self.GAIN = 0

        self.REFERENCE\_UNIT = 1  # The value returned by the hx711 that corresponds to your reference unit AFTER dividing by the SCALE.

        self.OFFSET = 1

        self.lastVal = long(0)

        self.DEBUG\_PRINTING = False

        self.byte\_format = 'MSB'

        self.bit\_format = 'MSB'

        self.set\_gain(gain)

        # Think about whether this is necessary.

        time.sleep(1)

    def convertToTwosComplement24bit(self, inputValue):

       # HX711 has saturating logic.

       if inputValue >= 0x7fffff:

          return 0x7fffff

       # If it's a positive value, just return it, masked with our max value.

       if inputValue >= 0:

          return inputValue & 0x7fffff

       if inputValue < 0:

          # HX711 has saturating logic.

          if inputValue < -0x800000:

             inputValue = -0x800000

          diff = inputValue + 0x800000

          return 0x800000 + diff

    def convertFromTwosComplement24bit(self, inputValue):

        return -(inputValue & 0x800000) + (inputValue & 0x7fffff)

    def is\_ready(self):

        # Calculate how long we should be waiting between samples, given the

        # sample rate.

        sampleDelaySeconds = 1.0 / self.sampleRateHz

        return time.time() >= self.lastReadTime + sampleDelaySeconds

    def set\_gain(self, gain):

        if gain is 128:

            self.GAIN = 1

        elif gain is 64:

            self.GAIN = 3

        elif gain is 32:

            self.GAIN = 2

        # Read out a set of raw bytes and throw it away.

        self.readRawBytes()

    def get\_gain(self):

        if self.GAIN == 1:

            return 128

        if self.GAIN == 3:

            return 64

        if self.GAIN == 2:

            return 32

        # Shouldn't get here.

        return 0

    def readRawBytes(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the virtual HX711 serial interface.

        self.readLock.acquire()

        # Wait until HX711 is ready for us to read a sample.

        while not self.is\_ready():

           pass

        self.lastReadTime = time.time()

        # Generate a 24bit 2s complement sample for the virtual HX711.

        rawSample = self.convertToTwosComplement24bit(self.generateFakeSample())

        # Read three bytes of data from the HX711.

        firstByte  = (rawSample >> 16) & 0xFF

        secondByte = (rawSample >> 8)  & 0xFF

        thirdByte  = rawSample & 0xFF

        # Release the Read Lock, now that we've finished driving the virtual HX711

        # serial interface.

        self.readLock.release()

        # Depending on how we're configured, return an orderd list of raw byte

        # values.

        if self.byte\_format == 'LSB':

           return [thirdByte, secondByte, firstByte]

        else:

           return [firstByte, secondByte, thirdByte]

    def read\_long(self):

        # Get a sample from the HX711 in the form of raw bytes.

        dataBytes = self.readRawBytes()

        if self.DEBUG\_PRINTING:

            print(dataBytes,)

        # Join the raw bytes into a single 24bit 2s complement value.

        twosComplementValue = ((dataBytes[0] << 16) |

                               (dataBytes[1] << 8)  |

                               dataBytes[2])

        if self.DEBUG\_PRINTING:

            print("Twos: 0x%06x" % twosComplementValue)

        # Convert from 24bit twos-complement to a signed value.

        signedIntValue = self.convertFromTwosComplement24bit(twosComplementValue)

        # Record the latest sample value we've read.

        self.lastVal = signedIntValue

        # Return the sample value we've read from the HX711.

        return int(signedIntValue)

    def read\_average(self, times=3):

        # Make sure we've been asked to take a rational amount of samples.

        if times <= 0:

            print("HX711().read\_average(): times must >= 1!!  Assuming value of 1.")

            times = 1

        # If we're only average across one value, just read it and return it.

        if times == 1:

            return self.read\_long()

        # If we're averaging across a low amount of values, just take an

        # arithmetic mean.

        if times < 5:

            values = int(0)

            for i in range(times):

                values += self.read\_long()

            return values / times

        # If we're taking a lot of samples, we'll collect them in a list, remove

        # the outliers, then take the mean of the remaining set.

        valueList = []

        for x in range(times):

            valueList += [self.read\_long()]

        valueList.sort()

        # We'll be trimming 20% of outlier samples from top and bottom of collected set.

        trimAmount = int(len(valueList) \* 0.2)

        # Trim the edge case values.

        valueList = valueList[trimAmount:-trimAmount]

        # Return the mean of remaining samples.

        return sum(valueList) / len(valueList)

    def get\_value(self, times=3):

        return self.read\_average(times) - self.OFFSET

    def get\_weight(self, times=3):

        value = self.get\_value(times)

        value = value / self.REFERENCE\_UNIT

        return value

    def tare(self, times=15):

        # If we aren't simulating Taring because it takes too long, just skip it.

        if not self.simulateTare:

            return 0

        # Backup REFERENCE\_UNIT value

        reference\_unit = self.REFERENCE\_UNIT

        self.set\_reference\_unit(1)

        value = self.read\_average(times)

        if self.DEBUG\_PRINTING:

            print("Tare value:", value)

        self.set\_offset(value)

        # Restore the reference unit, now that we've got our offset.

        self.set\_reference\_unit(reference\_unit)

        return value;

    def set\_reading\_format(self, byte\_format="LSB", bit\_format="MSB"):

        if byte\_format == "LSB":

            self.byte\_format = byte\_format

        elif byte\_format == "MSB":

            self.byte\_format = byte\_format

        else:

            print("Unrecognised byte\_format: \"%s\"" % byte\_format)

        if bit\_format == "LSB":

            self.bit\_format = bit\_format

        elif bit\_format == "MSB":

            self.bit\_format = bit\_format

        else:

            print("Unrecognised bit\_format: \"%s\"" % bit\_format)

    def set\_offset(self, offset):

        self.OFFSET = offset

    def get\_offset(self):

        return self.OFFSET

    def set\_reference\_unit(self, reference\_unit):

        # Make sure we aren't asked to use an invalid reference unit.

        if reference\_unit == 0:

            print("HX711().set\_reference\_unit(): Can't use 0 as a reference unit!!")

            return

        self.REFERENCE\_UNIT = reference\_unit

    def power\_down(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the HX711 serial interface.

        self.readLock.acquire()

        # Wait 100us for the virtual HX711 to power down.

        time.sleep(0.0001)

        # Release the Read Lock, now that we've finished driving the HX711

        # serial interface.

        self.readLock.release()

    def power\_up(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the HX711 serial interface.

        self.readLock.acquire()

        # Wait 100 us for the virtual HX711 to power back up.

        time.sleep(0.0001)

        # Release the Read Lock, now that we've finished driving the HX711

        # serial interface.

        self.readLock.release()

        # HX711 will now be defaulted to Channel A with gain of 128.  If this

        # isn't what client software has requested from us, take a sample and

        # throw it away, so that next sample from the HX711 will be from the

        # correct channel/gain.

        if self.get\_gain() != 128:

            self.readRawBytes()

    def reset(self):

        # self.power\_down()

        # self.power\_up()

        # Mark time when we were reset.  We'll use this for sample generation.

        self.resetTimeStamp = time.time()

    def generateFakeSample(self):

       sampleTimeStamp = time.time() - self.resetTimeStamp

       noiseScale = 1.0

       noiseValue = random.randrange(-(noiseScale \* 1000),(noiseScale \* 1000)) / 1000.0

       sample     = math.sin(math.radians(sampleTimeStamp \* 20)) \* 72.0

       self.sampleCount += 1

       if sample < 0.0:

          sample = -sample

       sample += noiseValue

       BIG\_ERROR\_SAMPLE\_FREQUENCY = 142

       ###BIG\_ERROR\_SAMPLE\_FREQUENCY = 15

       BIG\_ERROR\_SAMPLES = [0.0, 40.0, 70.0, 150.0, 280.0, 580.0]

       if random.randrange(0, BIG\_ERROR\_SAMPLE\_FREQUENCY) == 0:

          sample = random.sample(BIG\_ERROR\_SAMPLES, 1)[0]

          print("Sample %d: Injecting %f as a random bad sample." % (self.sampleCount, sample))

       sample \*= 1000

       sample \*= self.REFERENCE\_UNIT

       return int(sample)

### hx711.py

#

import RPi.GPIO as GPIO

import time

import threading

class HX711:

    def \_\_init\_\_(self, dout, pd\_sck, gain=128):

        self.PD\_SCK = pd\_sck

        self.DOUT = dout

        # Mutex for reading from the HX711, in case multiple threads in client

        # software try to access get values from the class at the same time.

        self.readLock = threading.Lock()

        GPIO.setmode(GPIO.BCM)

        GPIO.setup(self.PD\_SCK, GPIO.OUT)

        GPIO.setup(self.DOUT, GPIO.IN)

        self.GAIN = 0

        # The value returned by the hx711 that corresponds to your reference

        # unit AFTER dividing by the SCALE.

        self.REFERENCE\_UNIT = 1

        self.REFERENCE\_UNIT\_B = 1

        self.OFFSET = 1

        self.OFFSET\_B = 1

        self.lastVal = int(0)

        self.DEBUG\_PRINTING = False

        self.byte\_format = 'MSB'

        self.bit\_format = 'MSB'

        self.set\_gain(gain)

        # Think about whether this is necessary.

        time.sleep(1)

    def convertFromTwosComplement24bit(self, inputValue):

        return -(inputValue & 0x800000) + (inputValue & 0x7fffff)

    def is\_ready(self):

        return GPIO.input(self.DOUT) == 0

    def set\_gain(self, gain):

        if gain is 128:

            self.GAIN = 1

        elif gain is 64:

            self.GAIN = 3

        elif gain is 32:

            self.GAIN = 2

        GPIO.output(self.PD\_SCK, False)

        # Read out a set of raw bytes and throw it away.

        self.readRawBytes()

    def get\_gain(self):

        if self.GAIN == 1:

            return 128

        if self.GAIN == 3:

            return 64

        if self.GAIN == 2:

            return 32

        # Shouldn't get here.

        return 0

    def readNextBit(self):

       # Clock HX711 Digital Serial Clock (PD\_SCK).  DOUT will be

       # ready 1us after PD\_SCK rising edge, so we sample after

       # lowering PD\_SCL, when we know DOUT will be stable.

       GPIO.output(self.PD\_SCK, True)

       GPIO.output(self.PD\_SCK, False)

       value = GPIO.input(self.DOUT)

       # Convert Boolean to int and return it.

       return int(value)

    def readNextByte(self):

       byteValue = 0

       # Read bits and build the byte from top, or bottom, depending

       # on whether we are in MSB or LSB bit mode.

       for x in range(8):

          if self.bit\_format == 'MSB':

             byteValue <<= 1

             byteValue |= self.readNextBit()

          else:

             byteValue >>= 1

             byteValue |= self.readNextBit() \* 0x80

       # Return the packed byte.

       return byteValue

    def readRawBytes(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the HX711 serial interface.

        self.readLock.acquire()

        # Wait until HX711 is ready for us to read a sample.

        while not self.is\_ready():

           pass

        # Read three bytes of data from the HX711.

        firstByte  = self.readNextByte()

        secondByte = self.readNextByte()

        thirdByte  = self.readNextByte()

        # HX711 Channel and gain factor are set by number of bits read

        # after 24 data bits.

        for i in range(self.GAIN):

           # Clock a bit out of the HX711 and throw it away.

           self.readNextBit()

        # Release the Read Lock, now that we've finished driving the HX711

        # serial interface.

        self.readLock.release()

        # Depending on how we're configured, return an orderd list of raw byte

        # values.

        if self.byte\_format == 'LSB':

           return [thirdByte, secondByte, firstByte]

        else:

           return [firstByte, secondByte, thirdByte]

    def read\_long(self):

        # Get a sample from the HX711 in the form of raw bytes.

        dataBytes = self.readRawBytes()

        if self.DEBUG\_PRINTING:

            print(dataBytes,)

        # Join the raw bytes into a single 24bit 2s complement value.

        twosComplementValue = ((dataBytes[0] << 16) |

                               (dataBytes[1] << 8)  |

                               dataBytes[2])

        if self.DEBUG\_PRINTING:

            print("Twos: 0x%06x" % twosComplementValue)

        # Convert from 24bit twos-complement to a signed value.

        signedIntValue = self.convertFromTwosComplement24bit(twosComplementValue)

        # Record the latest sample value we've read.

        self.lastVal = signedIntValue

        # Return the sample value we've read from the HX711.

        return int(signedIntValue)

    def read\_average(self, times=3):

        # Make sure we've been asked to take a rational amount of samples.

        if times <= 0:

            raise ValueError("HX711()::read\_average(): times must >= 1!!")

        # If we're only average across one value, just read it and return it.

        if times == 1:

            return self.read\_long()

        # If we're averaging across a low amount of values, just take the

        # median.

        if times < 5:

            return self.read\_median(times)

        # If we're taking a lot of samples, we'll collect them in a list, remove

        # the outliers, then take the mean of the remaining set.

        valueList = []

        for x in range(times):

            valueList += [self.read\_long()]

        valueList.sort()

        # We'll be trimming 20% of outlier samples from top and bottom of collected set.

        trimAmount = int(len(valueList) \* 0.2)

        # Trim the edge case values.

        valueList = valueList[trimAmount:-trimAmount]

        # Return the mean of remaining samples.

        return sum(valueList) / len(valueList)

    # A median-based read method, might help when getting random value spikes

    # for unknown or CPU-related reasons

    def read\_median(self, times=3):

       if times <= 0:

          raise ValueError("HX711::read\_median(): times must be greater than zero!")

       # If times == 1, just return a single reading.

       if times == 1:

          return self.read\_long()

       valueList = []

       for x in range(times):

          valueList += [self.read\_long()]

       valueList.sort()

       # If times is odd we can just take the centre value.

       if (times & 0x1) == 0x1:

          return valueList[len(valueList) // 2]

       else:

          # If times is even we have to take the arithmetic mean of

          # the two middle values.

          midpoint = len(valueList) / 2

          return sum(valueList[midpoint:midpoint+2]) / 2.0

    # Compatibility function, uses channel A version

    def get\_value(self, times=3):

        return self.get\_value\_A(times)

    def get\_value\_A(self, times=3):

        return self.read\_median(times) - self.get\_offset\_A()

    def get\_value\_B(self, times=3):

        # for channel B, we need to set\_gain(32)

        g = self.get\_gain()

        self.set\_gain(32)

        value = self.read\_median(times) - self.get\_offset\_B()

        self.set\_gain(g)

        return value

    # Compatibility function, uses channel A version

    def get\_weight(self, times=3):

        return self.get\_weight\_A(times)

    def get\_weight\_A(self, times=3):

        value = self.get\_value\_A(times)

        value = value / self.REFERENCE\_UNIT

        return value

    def get\_weight\_B(self, times=3):

        value = self.get\_value\_B(times)

        value = value / self.REFERENCE\_UNIT\_B

        return value

    # Sets tare for channel A for compatibility purposes

    def tare(self, times=15):

        self.tare\_A(times)

    def tare\_A(self, times=15):

        # Backup REFERENCE\_UNIT value

        backupReferenceUnit = self.get\_reference\_unit\_A()

        self.set\_reference\_unit\_A(1)

        value = self.read\_average(times)

        if self.DEBUG\_PRINTING:

            print("Tare A value:", value)

        self.set\_offset\_A(value)

        # Restore the reference unit, now that we've got our offset.

        self.set\_reference\_unit\_A(backupReferenceUnit)

        return value

    def tare\_B(self, times=15):

        # Backup REFERENCE\_UNIT value

        backupReferenceUnit = self.get\_reference\_unit\_B()

        self.set\_reference\_unit\_B(1)

        # for channel B, we need to set\_gain(32)

        backupGain = self.get\_gain()

        self.set\_gain(32)

        value = self.read\_average(times)

        if self.DEBUG\_PRINTING:

            print("Tare B value:", value)

        self.set\_offset\_B(value)

        # Restore gain/channel/reference unit settings.

        self.set\_gain(backupGain)

        self.set\_reference\_unit\_B(backupReferenceUnit)

        return value

    def set\_reading\_format(self, byte\_format="LSB", bit\_format="MSB"):

        if byte\_format == "LSB":

            self.byte\_format = byte\_format

        elif byte\_format == "MSB":

            self.byte\_format = byte\_format

        else:

            raise ValueError("Unrecognised byte\_format: \"%s\"" % byte\_format)

        if bit\_format == "LSB":

            self.bit\_format = bit\_format

        elif bit\_format == "MSB":

            self.bit\_format = bit\_format

        else:

            raise ValueError("Unrecognised bitformat: \"%s\"" % bit\_format)

    # sets offset for channel A for compatibility reasons

    def set\_offset(self, offset):

        self.set\_offset\_A(offset)

    def set\_offset\_A(self, offset):

        self.OFFSET = offset

    def set\_offset\_B(self, offset):

        self.OFFSET\_B = offset

    def get\_offset(self):

        return self.get\_offset\_A()

    def get\_offset\_A(self):

        return self.OFFSET

    def get\_offset\_B(self):

        return self.OFFSET\_B

    def set\_reference\_unit(self, reference\_unit):

        self.set\_reference\_unit\_A(reference\_unit)

    def set\_reference\_unit\_A(self, reference\_unit):

        # Make sure we aren't asked to use an invalid reference unit.

        if reference\_unit == 0:

            raise ValueError("HX711::set\_reference\_unit\_A() can't accept 0 as a reference unit!")

            return

        self.REFERENCE\_UNIT = reference\_unit

    def set\_reference\_unit\_B(self, reference\_unit):

        # Make sure we aren't asked to use an invalid reference unit.

        if reference\_unit == 0:

            raise ValueError("HX711::set\_reference\_unit\_A() can't accept 0 as a reference unit!")

            return

        self.REFERENCE\_UNIT\_B = reference\_unit

    def get\_reference\_unit(self):

        return get\_reference\_unit\_A()

    def get\_reference\_unit\_A(self):

        return self.REFERENCE\_UNIT

    def get\_reference\_unit\_B(self):

        return self.REFERENCE\_UNIT\_B

    def power\_down(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the HX711 serial interface.

        self.readLock.acquire()

        # Cause a rising edge on HX711 Digital Serial Clock (PD\_SCK).  We then

        # leave it held up and wait 100 us.  After 60us the HX711 should be

        # powered down.

        GPIO.output(self.PD\_SCK, False)

        GPIO.output(self.PD\_SCK, True)

        time.sleep(0.0001)

        # Release the Read Lock, now that we've finished driving the HX711

        # serial interface.

        self.readLock.release()

    def power\_up(self):

        # Wait for and get the Read Lock, incase another thread is already

        # driving the HX711 serial interface.

        self.readLock.acquire()

        # Lower the HX711 Digital Serial Clock (PD\_SCK) line.

        GPIO.output(self.PD\_SCK, False)

        # Wait 100 us for the HX711 to power back up.

        time.sleep(0.0001)

        # Release the Read Lock, now that we've finished driving the HX711

        # serial interface.

        self.readLock.release()

        # HX711 will now be defaulted to Channel A with gain of 128.  If this

        # isn't what client software has requested from us, take a sample and

        # throw it away, so that next sample from the HX711 will be from the

        # correct channel/gain.

        if self.get\_gain() != 128:

            self.readRawBytes()

    def reset(self):

        self.power\_down()

        self.power\_up()

### LDR.py

import serial

import sys

from time import sleep

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

import json

from datetime import date

import datetime

def customCallback(client, userdata, message):

    print("Received a new message: ")

    print(message.payload)

    print("from topic: ")

    print(message.topic)

    print("--------------\n\n")

host = "a76z46r8md8xy-ats.iot.us-east-1.amazonaws.com"

rootCAPath = "rootca.pem"

certificatePath = "certificate.pem.crt"

privateKeyPath = "private.pem.key"

my\_rpi = AWSIoTMQTTClient("BasicLightPubSub")

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)

while 1:

    ser=serial.Serial("/dev/ttyUSB0",9600)

    b=ser.readline()

    bd=b.decode()

    value = float(bd.rstrip())

    #print("Light value:" + str(value))

    #if value>700.0:

        #print("Too much light")

    message = {}

    message["deviceid"] = "deviceid\_light"

    now = datetime.datetime.now()

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

    message["value"] = value

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

    sleep(5) # use this to set interval in seconds

### picam\_rekognition.py

import boto3

import botocore

from picamera import PiCamera

from time import sleep

import os

from api import Dynamodb

from gpiozero import LED

import re

# Set the filename and bucket name

BUCKET = 'imageawsbucket'  # replace with your own unique bucket name

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

file\_path = '/home/pi/Desktop'

file\_name = 'image.jpg'

dynamodb = Dynamodb()

led = LED(18)

camera = PiCamera()

def onLED():

    led.on()

def offLED():

    led.off()

def takePhoto():

    if os.path.exists(file\_path + "/" + file\_name):

        os.remove(file\_path + "/" + file\_name)

    sleep(2)

    onLED()

    sleep(1)

    camera.capture(file\_path + "/" + file\_name)

    offLED()

    return

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\_name, key, max\_labels=10, min\_confidence=95, region="us-east-1"):

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

    response = rekognition.detect\_labels(

        Image={

            "S3Object": {

                "Bucket": bucket\_name,

                "Name": key,

            }

        },

        MaxLabels=max\_labels,

        MinConfidence=min\_confidence,

    )

    listdontupdate=[]

    items = []

    items = dynamodb.get\_all("upc\_product")

    for label in response['Labels']:

        print ("Label: " + label['Name'])

        #print ("Confidence: " + str(label['Confidence']))

        print ("Number of instances:" +  str(len(label['Instances'])))

        expression = "SET qty = :val"

        values = {':val':len(label['Instances'])}

        #print(items)

        for i in items:

            key={"upc": i['upc']}

            if label['Name'] == i['item\_desc']:

                dynamodb.update("inventory",key,expression,values)

                print("Database Updated")

                listdontupdate.append(i['upc'])

    #update everything else in the fridge to 0

    for i in items:

        if re.search("[8]{6,6}\\d{7,7}",str(i['upc'])) is not None:

            if i['upc'] not in listdontupdate:

                expression = "SET qty = :val"

                values = {':val': '0'}

                key={"upc": i['upc']}

                dynamodb.update("inventory",key,expression,values)

                print("extras updated")

        #for instance in label['Instances']:

        #   print (" Bounding box")

        #   print (" Top: " + str(instance['BoundingBox']['Top']))

        #   print (" Left: " + str(instance['BoundingBox']['Left']))

        #   print (" Width: " + str(instance['BoundingBox']['Width']))

        #   print (" Height: " + str(instance['BoundingBox']['Height']))

        #   print (" Confidence: " + str(instance['Confidence']))

        #   print()

while 1:

    takePhoto()

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

    detect\_labels(BUCKET, file\_name)

    sleep(3) # if not testing set to 60 or longer

    #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"]

### weight.py

#! /usr/bin/python2

# coding: utf-8

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

import time

import sys

import mysql.connector

from time import sleep

from datetime import date

import datetime

import RPi.GPIO as GPIO

from gpiozero import LED

from twilio.rest import Client

import json

from api import Dynamodb

dynamodb = Dynamodb()

def customCallback(client, userdata, message):

    print("Received a new message: ")

    print(message.payload)

    print("from topic: ")

    print(message.topic)

    print("--------------\n\n")

# Set up for the HX711 module

EMULATE\_HX711=False

referenceUnit = 479

if not EMULATE\_HX711:

    import RPi.GPIO as GPIO

    from hx711 import HX711

else:

    from emulated\_hx711 import HX711

# Assigning GPIO pins to the hardware

# GPIO: 5 dout purple

# 13 pd\_sck yellow

hx = HX711(5, 6)

hx.set\_reading\_format("MSB", "MSB")

# The ‘referenceUnit’ in this project is set to 479 to read the weight accurately.

# First set it to 1 then get the readings of the weight on the sensor of any object

# Based on the reading shown, divide by the actual weight of the object which will be the value of the ‘referenceUnit’

hx.set\_reference\_unit(referenceUnit)

hx.reset()

print('Wait for the message to say that the tare is done before placing the water bottle on the sensor')

hx.tare()

print("Tare done! You may place the object now")

# Giving the user time to put the bottle on the scale

sleep(5)

host = "a76z46r8md8xy-ats.iot.us-east-1.amazonaws.com"

rootCAPath = "rootca.pem"

certificatePath = "certificate.pem.crt"

privateKeyPath = "private.pem.key"

my\_rpi = AWSIoTMQTTClient("BasicWeightPubSub")

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/weight", 1, customCallback)

sleep(2)

try:

    while 1:

        # Getting the weight

        val = hx.get\_weight(5)

        hx.power\_down()

        hx.power\_up()

        message = {}

        message["deviceid"] = "deviceid\_weight"

        now = datetime.datetime.now()

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

        message["value"] = val

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

        sleep(5) # use this to set interval in seconds

        items = []

        items = dynamodb.get\_all("upc\_product")

        expression = "SET qty = :val"

        values = {':val':val}

        #print(items)

        for i in items:

            #print(i['upc'])

            if i['item\_desc']=="Rice":

                key={"upc": i['upc']}

                dynamodb.update("inventory",key,expression,values)

                print("Database Updated")

except:

    print("program terminated")

### bothandler.py

#Credits for the BotHandler Class

#https://github.com/magnitopic/YouTubeCode

import requests

class BotHandler:

    token = '1620569172:AAEoFuZf02nunzAHPrUhyt50lGexusDrFIE' #Telegram Bot Token

    def \_\_init\_\_(self):

            self.api\_url = "https://api.telegram.org/bot{}/".format(self.token)

    #url = "https://api.telegram.org/bot<token>/"

    def get\_updates(self, offset=0, timeout=30):

        method = 'getUpdates'

        params = {'timeout': timeout, 'offset': offset}

        resp = requests.get(self.api\_url + method, params)

        result\_json = resp.json()['result']

        return result\_json

    def send\_message(self, chat\_id, text):

        params = {'chat\_id': chat\_id, 'text': text, 'parse\_mode': 'HTML'}

        method = 'sendMessage'

        resp = requests.post(self.api\_url + method, params)

        return resp

    def send\_messages(self,chat\_id,text):

        for x in chat\_id:

            params = {'chat\_id': chat\_id, 'text': text, 'parse\_mode': 'HTML'}

            method = 'sendMessage'

            resp = requests.post(self.api\_url + method, params)

        return True

    def send\_image(self,chat\_id,text,image):

        params = {'chat\_id': chat\_id, 'caption': text}

        files = {'photo':image}

        headers = {'content-type':"multipart/form-data"}

        method = 'sendPhoto'

        resp = requests.post(self.api\_url + method, files=files,data=params)

        print(resp.content)

        return resp

    def get\_first\_update(self):

        get\_result = self.get\_updates()

        if len(get\_result) > 0:

            last\_update = get\_result[0]

        else:

            last\_update = None

        return last\_update

### mysql\_api.py

import mysql.connector

import datetime

import decimal

import sys

import json

from flask import jsonify

import numpy

class GenericEncoder(json.JSONEncoder):

    def default(self,obj):

        if isinstance(obj, numpy.generic):

            return numpy.asscaler(obj)

        elif isinstance(obj, datetime.datetime):

            return obj.strftime('%Y-%m-%d %H:%M:%S')

        elif isinstance(obj, decimal.Decimal):

            return float(obj)

        else:

            return json.JSONEncoder.default(self,obj)

website = "http://10.10.10.161:8001"

videopath = "/home/pi/Desktop/IOT PROJ/Videos"

class Database():

    username = 'root'

    password = 'P@ssw0rd'

    host = '10.10.10.161'

    database = 'iot'

    cnx = None

    cursor = None

    def \_\_init\_\_(self):

        try:

            self.cnx = mysql.connector.connect(user=self.username,password=self.password,host=self.host,database=self.database)

            self.cursor = self.cnx.cursor()

        except:

            print("Unable to connect to database!")

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

    def select\_one\_row(self,sql):

        try:

            self.cursor.execute(sql)

            entry = self.cursor.fetchone()

            self.cnx.commit()

            return entry

        except:

            print("Program Failure!")

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return None

    def select\_rows(self,sql):

        try:

            self.cursor.execute(sql)

            entry = self.cursor.fetchall()

            self.cnx.commit()

            return entry

        except:

            print("Program Failure!")

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return None

    def insert\_update(self,sql):

        try:

            self.cursor.execute(sql)

            self.cnx.commit()

            return True

        except:

            print("Program Failure!")

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return False

    def verify(self,username,password):

        try:

            sql = "SELECT \* FROM users WHERE username = '{}' AND password = '{}'".format(username,password)

            return self.select\_one\_row(sql)

        except:

            print("Program Failure!")

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return False

    def get\_subscribers\_list(self):

        try:

            sql = "SELECT chat\_id FROM subscribe WHERE subscribe='1'"

            subscribers = []

            entry = self.select\_rows(sql)

            for x in entry:

                subscribers.append(x[0])

            return subscribers;

        except:

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return None

    def fetch\_fromdb\_as\_json(self,sql):

        try:

            self.cursor.execute(sql)

            row\_headers=[x[0] for x in self.cursor.description]

            results = self.cursor.fetchall()

            data = []

            for result in results:

                data.append(dict(zip(row\_headers,result)))

            data\_reversed = data[::-1]

            data = {'data':data\_reversed}

            return self.data\_to\_json(data)

        except:

            print(sys.exc\_info()[0])

            print(sys.exc\_info()[1])

            return None

    def data\_to\_json(self,data):

        json\_data = json.dumps(data,cls=GenericEncoder)

        return json\_data

    def close(self):

        self.cursor.close()

        self.cnx.close()

### telegram\_app.py

import requests

import datetime

from api import Dynamodb, website

from bothandler import BotHandler

import boto3

dynamodb = Dynamodb()

iot\_bot = BotHandler()

def subscribe(chatid):

    key = {"chat\_id":chatid}

    result = dynamodb.get("subscribe",key)

    if result:

        #get subscriber status

        subscribe = result['s\_status']

        if subscribe == 1:

            return "Already Subscribed!"

        else:

            #change status

            values = {":stat":1}

            upd\_expression = "SET s\_status = :stat"

            resp = dynamodb.update("subscribe",key,upd\_expression,values)

            if resp:

                return "Status changed to subscribed!"

            else:

                return "An error occurred!"

    else:

        #not in table, add a new entry

        values = {"chat\_id":chatid,"s\_status":1}

        resp = dynamodb.add("subscribe",values)

        if resp:

            return "Added to the subscriber list!"

        else:

            return "An error occurred!"

def unsubscribe(chatid):

    key = {"chat\_id":chatid}

    result = dynamodb.get("subscribe",key)

    if result:

        #user in database

        subscribe = result['s\_status']

        if subscribe == 0:

            return "Already Not Subscribed!"

        else:

            values = {":stat":0}

            upd\_expression = "SET s\_status = :stat"

            resp = dynamodb.update("subscribe",key,upd\_expression,values)

            if resp:

                return "Status changed to unsubscribed!"

            else:

                return "An error occurred!"

    else:

        return "You have not subscribed before!"

def toggle\_barcode\_status():

    Key = {'id': 1}

    result = dynamodb.get('barcode\_status', Key)

    if result:

        mode = result['b\_mode']

        if mode == 0:

            mode = 1

        else:

            mode = 0

        print(mode)

        ExpressionAttributeValues = {':val1': mode}

        Key = {'id': 1}

        UpdateExpression = 'SET b\_mode = :val1'

        resp = dynamodb.update('barcode\_status', Key, UpdateExpression, ExpressionAttributeValues)

        if resp:

            if mode == 0:

                return "Barcode Scanner now in deduct mode!"

            else:

                return "Barcode Scanner now in add mode!"

        else:

            return "An error occurred!"

def grocery\_list():

    #fetch from inventory

    #compare with threshold value in upc\_product

    inventory = dynamodb.get\_all("inventory")

    items = ""

    for x in inventory:

        #fetch the threshold value

        upc = x['upc']

        qty = int(x['qty'])

        upc\_product = dynamodb.get("upc\_product", {"upc": upc})

        threshold\_value = int(upc\_product['threshold'])

        item\_desc = upc\_product['item\_desc']

        if qty <= threshold\_value:

            x = "Item Description: {}, Quantity Left: {}".format(item\_desc,qty)

            items = "{}{}\n".format(items,x)

    return items

def fridgeimage():

    s3 = boto3.client("s3")

    x = s3.get\_object(Bucket="imageawsbucket",Key="image.jpg")

    image = x['Body'].read()

    return image

def main():

    new\_offset = 0

    print("Telegram bot launching...")

    while True:

        all\_updates=iot\_bot.get\_updates(new\_offset)

        if len(all\_updates) > 0:

            for current\_update in all\_updates:

                first\_update\_id = current\_update['update\_id']

                if 'text' not in current\_update['message']:

                    first\_chat\_text='New member'

                else:

                    first\_chat\_text = current\_update['message']['text']

                first\_chat\_id = current\_update['message']['chat']['id']

                if 'first\_name' in current\_update['message']:

                    first\_chat\_name = current\_update['message']['chat']['first\_name']

                elif 'new\_chat\_member' in current\_update['message']:

                    first\_chat\_name = current\_update['message']['new\_chat\_member']['username']

                elif 'from' in current\_update['message']:

                    first\_chat\_name = current\_update['message']['from']['first\_name']

                else:

                    first\_chat\_name = "unknown"

                if first\_chat\_text == "/togglestate":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, toggle\_barcode\_status())

                elif first\_chat\_text == "/website":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, website)

                elif first\_chat\_text == "/subscribe":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, subscribe(first\_chat\_id))

                elif first\_chat\_text == "/unsubscribe":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, unsubscribe(first\_chat\_id))

                elif first\_chat\_text == "/grocerylist":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, grocery\_list())

                elif first\_chat\_text == "/fridgeimage":

                    new\_offset = first\_update\_id + 1

                    iot\_bot.send\_message(first\_chat\_id, "Image coming your way, hang tight!")

                    iot\_bot.send\_image(first\_chat\_id, "Image of Fridge",fridgeimage())

                    print("After")

if \_\_name\_\_ == '\_\_main\_\_':

    try:

        main()

    except KeyboardInterrupt:

        exit()

# 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 |
| Tiang Kar Hoe | Sensors, MQTT pubsub | 50% |
| Ng Wei En, Amos | Web Application, APIs | 50% |
|  |  |  |

# Section 7 References

<https://github.com/app-generator/django-dashboard-atlantis>

<https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-dg.pdf>

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