CODE

Web Application to get the Live location:

index.html:

```
<!DOCTYPE html>
<html>

<head>
  <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"integrity="sha384-
    ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
    crossorigin="anonymous">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Smart Waste Management System</title>
    <link rel="icon" type="image/x-icon" href="/imgs/DUMPSTER.png">
    <link href="style.css" rel="stylesheet" type="text/css" />
    <script src="https://www.gstatic.com/firebasejs/9.14.0/firebase-app.js"></script>
    <script  src="https://www.gstatic.com/firebasejs/9.14.0/firebase-analytics.js"></script>
    <script  src="https://www.gstatic.com/firebasejs/9.14.0/firebase-database.js"></script>

    <script>
      var firebaseConfig =
      {
                apiKey: "AIzaSyCcZk7b1CLOGviwUpthRDLotrmFX0MFuTs",
        authDomain: "swms-3840.firebaseapp.com",
        projectId: "swms-3840",
        storageBucket: "swms-3840.appspot.com",
        messagingSenderId: "479902726304",
        appId: "1:479902726304:web:3d822880d1275ee57a71c5",
        measurementId: "G-MHP4N77MTP"
      };
      firebase.initializeApp(firebaseConfig)
    </script>
    <script defer src="db.js"></script>
</head>
```

```
<body style="background-color:#1F1B24;">
    <script src="maps.js"></script>
    <div id="map_container">
      <h1 id="live_location_heading" >LIVE LOCATION</h1>
      <div id="map"></div>
      <div id="alert_msg">ALERT MESSAGE!</div>
    </div>
    </div>
    <center>
       <a href="https://goo.gl/maps/G9XET5mzSw1ynHQ18" type="button" class="btn btn-dark">
          DUMPSTER
       </a>
    </center>
    <script
      src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBBLyWj-
    3FWtCbCXGW3ysEiI2fDfrv2v0Q&callback=myMap"></script></div>
</body>
</html>
```

db.js:

```
const cap_status = document.getElementById("cap_status");
const alert_msg = document.getElementById("alert_msg");

var ref = firebase.database().ref();

ref.on(
  "value",
  function (snapshot) {
    snapshot.forEach(function (childSnapshot) {
    var value = childSnapshot.val();

      const alert_msg_val = value.alert;
      const cap_status_val = value.distance_status;

      alert_msg.innerHTML = `${alert_msg_val}`;
    });
  },
  function (error) { console.log("Error: " + error.code);
  }
);
```

maps.js:

```
const database = firebase.database();

function myMap() {
  var ref1 = firebase.database().ref();

  ref1.on(
    "value",
    function (snapshot) {
      snapshot.forEach(function (childSnapshot) {
      var value = childSnapshot.val();
        const latitude = value.latitude;
        const longitude = value.longitude;

        var latlong = { lat: latitude, lng: longitude };
        var mapProp = {
          center: new google.maps.LatLng(latlong),
          zoom: 10,
        };
        var map = new google.maps.Map(document.getElementById("map"), mapProp);
        var marker = new google.maps.Marker({ position: latlong });
        marker.setMap(map);
      });
    },
    function (error) {
      console.log("Error: " + error.code);
    }
  );
  }
```

<u>Code to evaluate the level of the garbage in bin:</u>

bin1.py:

```
import requests
import json
import ibmiotf.application
import ibmiotf.device
import time
import random
import sys

# watson device details
organization = "73ffyv"
devicType = "BIN1"
deviceId = "BIN1ID"
authMethod= "token"
authToken= "123456789"

#generate random values for randomo variables (temperature&humidity)
def myCommandCallback(cmd):
    global a
    print("command recieved is:%s" %cmd.data['command'])
    control=cmd.data['command']
    print(control)

try:
    deviceOptions={"org": organization, "type": devicType,"id": deviceId,"auth-method":authMethod,"auth-
token":authToken}
    deviceCli = ibmiotf.device.Client(deviceOptions)
except Exception as e:
    print("Exception while connecting device %s" %str(e))
    sys.exit()

#connect and send a datapoint "temp" with value integer value into the cloud as a type of event for every 10
seconds
deviceCli.connect()

while True:
    distance= random.randint(10,70)
```

```
loadcell= random.randint(5,15)
data= {'dist':distance,'load':loadcell}
if loadcell < 13 and loadcell > 15:
load = "90 %"
elif loadcell < 8 and loadcell > 12:
     load = "60 %"
elif loadcell < 4 and loadcell > 7:
     load = "40 %"
else:
     load = "0 %"



if distance < 15:
     dist = 'Risk warning:' 'Garbage level is high, collection time :) 90 %'
elif distance < 40 and distance >16:
     dist = 'Risk warning:' 'garbage is above 60%'
elif distance < 60 and distance > 41:
     dist = 'Risk warning:' '40 %'
else:
     dist = 'Risk warning:' '17 %'



if load == "90 %" or distance == "90 %":
     warn  = 'alert :' ' Garbage level is high, collection time :)'
elif load == "60 %" or distance == "60 %":
     warn = 'alert :' 'garbage is above 60%'
else :
     warn = 'alert :' 'Levels are low, collection not needed '

def myOnPublishCallback(lat=11.035081,long=77.014616):
   print("Peelamedu, Coimbatore")
   print("published distance = %s " %distance,"loadcell:%s " %loadcell,"lon = %s " %long,"lat = %s"
%lat)
   print(load)
```

```
        print(dist)
        print(warn)

    time.sleep(10)
    success=deviceCli.publishEvent ("IoTSensor","json",warn,qos=0,on_publish= myOnPublishCallback)
    success=deviceCli.publishEvent ("IoTSensor","json",data,qos=0,on_publish= myOnPublishCallback)

    if not success:
        print("not connected to ibmiot")

    time.sleep(30)
    deviceCli.commandCallback=myCommandCallback

#disconnect the device
deviceCli.disconnect()
```

bin2.py:

```
import requests
import json
import ibmiotf.application
import ibmiotf.device
import time
import random
import sys

# watson device details
organization = "73ffyv"
devicType = "BIN2"
deviceId = "BIN2ID"
authMethod= "token"
authToken= "123456789"

#generate random values for randomo variables (temperature&humidity)
def myCommandCallback(cmd):
    global a
    print("command recieved is:%s" %cmd.data['command'])
    control=cmd.data['command']
    print(control)
try:
    deviceOptions={"org": organization, "type": devicType,"id": deviceId,"auth-method":authMethod,"auth-
token":authToken}
    deviceCli = ibmiotf.device.Client(deviceOptions)
except Exception as e:
    print("Exception while connecting device %s" %str(e))
    sys.exit()

#connect and send a datapoint "temp" with value integer value into the cloud as a type of event for every 10
seconds
deviceCli.connect()

while True:
    distance= random.randint(10,70)
    loadcell= random.randint(5,15)
    data= {'dist':distance,'load':loadcell}


    if loadcell < 13 and loadcell > 15:
        load = "90 %"
    elif loadcell < 8 and loadcell > 12:
          load = "60 %"
    elif loadcell < 4 and loadcell > 7:
          load = "40 %"
    else:
          load = "0 %"


    if distance < 15:
          dist = 'Risk warning:' 'Garbage level is high, collection time :) 90 %'
    elif distance < 40 and distance >16:
          dist = 'Risk warning:' 'garbage is above 60%'
    elif distance < 60 and distance > 41:
          dist = 'Risk warning:' '40 %'
    else:
          dist = 'Risk warning:' '17 %'


    if load == "90 %" or distance == "90 %":
          warn  = 'alert :' ' Garbage level is high, collection time :)'
    elif load == "60 %" or distance == "60 %":
          warn = 'alert :' 'garbage is above 60%'
    else :
         warn = 'alert :' 'Levels are low, collection not needed '
   def myOnPublishCallback(lat=11.068774,long=77.092978):print("PSG iTech, Coimbatore")
       print("published distance = %s " %distance,"loadcell:%s " %loadcell,"lon = %s " %long,"lat = %s"
     %lat)
        print(load)
        print(dist)
        print(warn)

    time.sleep(10)
    success=deviceCli.publishEvent ("IoTSensor","json",warn,qos=0,on_publish= myOnPublishCallback)
    success=deviceCli.publishEvent ("IoTSensor","json",data,qos=0,on_publish= myOnPublishCallback)

    if not success:
        print("not connected to ibmiot")

    time.sleep(30)
    deviceCli.commandCallback=myCommandCallback

#disconnect the device
deviceCli.disconnect()
```

bin3.py:

```
import requests
import json
import ibmiotf.application
import ibmiotf.device
import time
import random
import sys

# watson device details
organization = "73ffyv"
devicType = "BIN3"
deviceId = "BIN3ID"
authMethod= "token"
authToken= "123456789"

#generate random values for randomo variables (temperature&humidity)
def myCommandCallback(cmd):
    global a
    print("command recieved is:%s" %cmd.data['command'])
    control=cmd.data['command']
```

```
        print(control)

try:
        deviceOptions={"org": organization, "type": devicType,"id": deviceId,"auth-
    method":authMethod,"auth-token":authToken}
        deviceCli = ibmiotf.device.Client(deviceOptions)
    except Exception as e:
        print("Exception while connecting device %s" %str(e))
        sys.exit()

#connect and send a datapoint "temp" with value integer value into the cloud as a type of event forevery
10 seconds
deviceCli.connect()

while True:
        distance= random.randint(10,70)
        loadcell= random.randint(5,15)
        data= {'dist':distance,'load':loadcell}



        if loadcell < 13 and loadcell > 15:
           load = "90 %"
        elif loadcell < 8 and loadcell > 12:
             load = "60 %"
        elif loadcell < 4 and loadcell > 7:
             load = "40 %"
        else:
             load = "0 %"


        if distance < 15:
             dist = 'Risk warning:' 'Garbage level is high, collection time :) 90 %'
        elif distance < 40 and distance >16:

             dist = 'Risk warning:' 'garbage is above 60%'
        elif distance < 60 and distance > 41:

             dist = 'Risk warning:' '40 %'
    else:
          dist = 'Risk warning:' '17 %'



    if load == "90 %" or distance == "90 %":
          warn  = 'alert :' ' Garbage level is high, collection time :)'
    elif load == "60 %" or distance == "60 %":
          warn = 'alert :' 'garbage is above 60%'
    else :
          warn = 'alert :' 'Levels are low, collection not needed '

    def  myOnPublishCallback(lat=11.007403,long=76.963439):
        print("Kattoor, Coimbatore")
        print("published distance = %s " %distance,"loadcell:%s " %loadcell,"lon = %s " %long,"lat = 
%s" %lat)
        print(load)
        print(dist)
        print(warn)

    time.sleep(10)
    success=deviceCli.publishEvent ("IoTSensor","json",warn,qos=0,on_publish=
myOnPublishCallback)
    success=deviceCli.publishEvent ("IoTSensor","json",data,qos=0,on_publish=
myOnPublishCallback)

    if not success:
        print("not connected to ibmiot")

    time.sleep(30) deviceCli.commandCallback=myCommandCallback
    #disconnect the devicedeviceCli.disconnect()
```

<u>Measuring the weight of the garbage bin:</u>

main.py:

```
from hx711 import HX711
```

```
hx = HX711(5,4,64)
print(1)
while True:
    hx.tare()
    read = hx.read()
    #average=hx.read_average()
    value=hx.read_average()
    print(value,"#")
```

hx711.py:

```
from machine import Pin, enable_irq, disable_irq, idle


class HX711:
    def___init_(self, dout, pd_sck, gain=128):


        self.pSCK = Pin(pd_sck , mode=Pin.OUT)
        self.pOUT = Pin(dout, mode=Pin.IN, pull=Pin.PULL_DOWN)
        self.pSCK.value(False)


        self.GAIN = 0
        self.OFFSET = 0
        self.SCALE = 1


        self.time_constant = 0.1
    self.filtered = 0

        self.set_gain(gain);

  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
        self.read()
        self.filtered = self.read()
        print('Gain & initial value set')

    def is_ready(self):
        return self.pOUT() == 0

    def read(self):
        # wait for the device being ready
        while self.pOUT() == 1:
            idle()

        # shift in data, and gain & channel info
        result = 0
        for j in range(24 + self.GAIN):
            state = disable_irq()
            self.pSCK(True)
            self.pSCK(False)
            enable_irq(state)
            result = (result << 1) | self.pOUT()

       # shift back the extra
```

```
    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

        self.read()
        self.filtered = self.read()
        print('Gain & initial value set')

    def is_ready(self):
        return self.pOUT() == 0

    def read(self):
        # wait for the device being ready
        while self.pOUT() == 1:
            idle()

        # shift in data, and gain & channel info
        result = 0
         for j in range(24 + self.GAIN):
            state = disable_irq()
            self.pSCK(True)
            self.pSCK(False)
            enable_irq(state)
            result = (result << 1) | self.pOUT()

        # shift back the extra bits
        result >>= self.GAIN

        # check sign
        if result > 0x7fffff:
            result -= 0x1000000

        return result
def read_average(self, times=3):
        s = 0
         for i in range(times):
         s += self.read()
        ss=(s/times)/210
        return '%.1f' % (ss)

    def read_lowpass(self):
        self.filtered += self.time_constant * (self.read() - self.filtered)
        return self.filtered

    def get_value(self, times=3):
        return self.read_average(times) - self.OFFSET

    def get_units(self, times=3):
        return self.get_value(times) / self.SCALE

    def tare(self, times=15):
        s = self.read_average(times)
        self.set_offset(s)

    def set_scale(self, scale):
        self.SCALE = scale

    def set_offset(self, offset):
        self.OFFSET = offset

    def set_time_constant(self, time_constant = None):
        if time_constant is None:
            return self.time_constant
        elif 0 < time_constant <1.0:
            self.time_constant = time_constant
```

```
def power_down(self):
    self.pSCK.value(False)
    self.pSCK.value(True)

def power_up(self):
    self.pSCK.value(False)
```