**Nalaiya Thiran**

Batch No: B7-1A3E

COIMBATORE INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

**Smart Waste Management System For**

**Metropolitan Cities**

**Team ID: PNT2022TMID52668**

**Team Members:**

1905083 HARI PRASATH M

1905094 MEERADEVI S

1905111 SANJAI V

1905119 THENDRAL S

**Project Guide**:

Industry mentor: Dinesh

Faculty Mentor: R Nedunchezhian

**ABSTRACT**

One issue that most cities and municipalities are dealing with currently, is the degradation of environmental cleanliness with reference to waste management. This is a result of improper garbage collection management. Dumping garbage onto the streets and in public areas is a common synopsis found in all developing countries and this mainly ends up affecting the environment and creating several unhygienic conditions. To avoid improper garbage management and to create a hygienic environment, the concept of automation is used in waste management system. Any city being referred to as a "smart city" is because of its orderly and tidy surroundings. But currently, many issues including those related to smart grids, smart environments, and smart living are faced. Today, cities and metropolitan areas' top priority is proper garbage management.

Traditional waste management techniques are too simplistic to create an effective and reliable waste management. The ideology put forward includes hardware and software technologies i.e. connecting Wi-Fi system to the normal dustbin in order to provide free internet facilities to the user for a particular period of time. The technology awards the user for keeping the surrounding clean and thus work hand in hand for the proper waste management in a locality. The smart bin uses multiple technologies - firstly the technology for measuring the amount of trash dumped and secondly the movement of the waste and lastly sending necessary signals and connecting the user to the WiFi system. The proposed system will function on client server model, a cause that will assure clean environment, good health, and pollution free society.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | **Introduction**   * 1. Project Overview   2. Purpose | 5 |
| 2 | **Literature Survey**  2.1 Existing Problem  2.2 References  2.3 Problem Statement Definition | 7 |
| 3 | **Ideation and Proposed Solution**  3.1 Empathy Map Canvas  3.2 Ideation & Brainstorming  3.3 Proposed Solution  3.4 Problem Solution fit | 11 |
| 4 | **Requirement Analysis**  4.1 Functional requirements  4.2 Non-Functional requirements | 16 |
| 5 | **Project Design**  5.1 Data Flow Diagrams  5.2 Solution & Technical Architecture  5.3 User Stories | 18 |
| 6 | **Project Planning and Scheduling**  6.1 Sprint Planning & Estimation  6.2 Sprint Delivery Schedule  6.3 Reports from JIRA | 20 |
| 7 | **Coding and Solutioning**  7.1 Feature 1  7.2 Feature 2  7.3 Feature 3 | 23 |
| 8 | **Testing**  8.1 Test Cases  8.2 User Acceptance Testing | 36 |
| 9 | **Results**  9.1 Performance Metrics | 37 |
| 10 | **Advantages and Disadvantages** | 40 |
| 11 | **Conclusion** | 41 |
| 12 | **Future Works** | 42 |
| 13 | **Appendix**  13.1 Source Code  13.2 Project Links | 43 |

**CHAPTER 1: INTRODUCTION**

* 1. **Project Overview**

Smart waste management is an innovative approach to handling and collecting waste. Based on IoT (Internet of Things) technology, smart waste management provides data on waste generation patterns and behaviour.

Our Smart waste management solution uses sensors placed in garbage bins to measure fill levels and notifies city collection services when bins are ready to be emptied. There are load and ultrasonic sensors placed to continuously monitor the bins. This data is sent to the cloud (via a microcontroller that is connected to Wi-Fi) where it is stored after which it is processed further. When the levels exceed a certain limit, a notification is sent to the garbage collector via a web application.

Over time, historical data collected by sensors can be used to identify fill patterns, optimize driver routes and schedules, and reduce operational costs. The cost of these sensors is steadily decreasing, making IoT waste bins more feasible to implement and more attractive.

* 1. **Purpose**

Around 2.1 billion tonnes of municipal solid waste is generated annually around the globe. Population growth and rapid urbanization lead to a huge increase in waste generation, so the traditional methods of waste collection have become inefficient and costly. This system cannot measure the fullness levels of containers, and as a result, half-full containers can be emptied, and in contrast, pre-filled ones need to wait until the next collection period comes. Moreover, since drivers collect empty bins, predefined collection routes of the system cause waste of time, an increase in fuel consumption, and excessive use of resources.

In today’s ever-technological world, an innovative and data-driven approach is the only way forward, the waste sector needs a solution that empowers event-driven waste collection. The most efficient way this extraordinary amount of waste can be solved is through smart waste management without obsolete methods of waste collection. This empowers municipalities, cities, and waste collectors to optimize their waste operations, become more sustainable, and make more intelligent business decisions.

**CHAPTER 2: LITERATURE SURVEY**

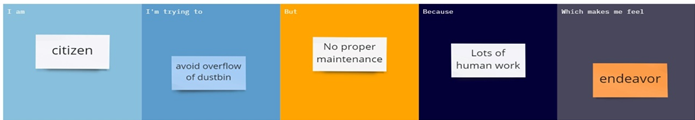
* 1. **Existing Problem**

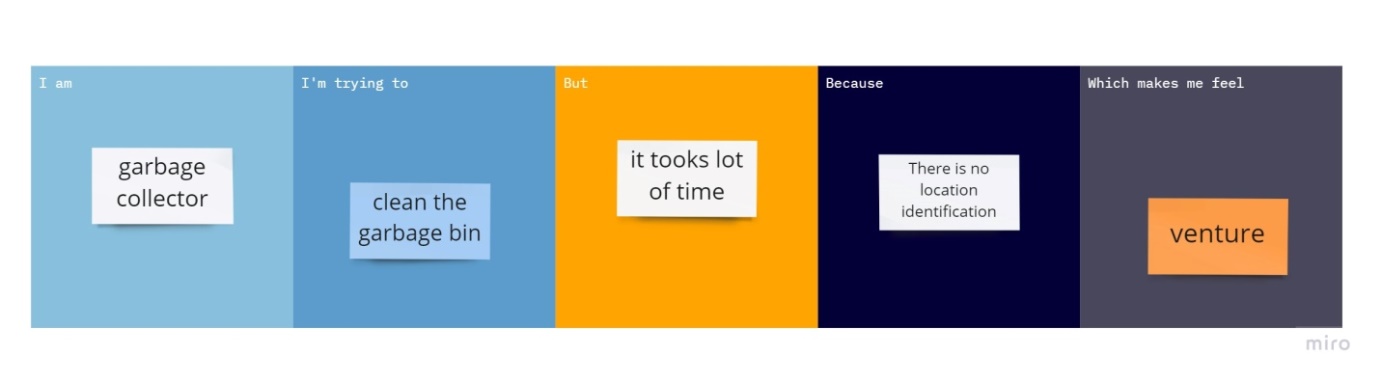
Around 80% of waste collections happen at the wrong time. Late waste collections lead to overflowing bins, unsanitary environments, citizen complaints, illegal dumping, and increased cleaning and collection costs. Early waste collections mean unnecessary carbon emissions, more traffic congestion, and higher running costs. The old way of doing waste management is highly inefficient. And in today’s ever-technological world, an innovative and data-driven approach is the only way forward. Traditionally, municipalities and waste management companies would operate on a fixed collection route and schedule. This means that waste collection trucks would drive the same collection route and empty every single waste container – even if the waste container did not need emptying. This means high labour and fuel costs – which residents ultimately foot the bill for.

* 1. **References**

|  |  |  |
| --- | --- | --- |
| **Paper Title** | **Author** | **Outcome** |
| IOT based Smart Garbage System | 1) T.Sinha  2) R.M. Sahuother | IoT Based Smart Garbage System which indicates directly that the dustbin is filled to a certain level by the garbage and cleaning or emptying them is a matter of immediate concern. This prevents lumping of garbage in the roadside dustbin which ends up giving foul smell and illness to people. The design of the smart dustbin includes a single by ultrasonic sensor which configured with Arduino Uno with this research, it is sending SMS to the Municipal Council that dustbin is to overflow. |
| Raspberry pi-based smart  waste management system using Internet of Things. | 1)Shaik Vaseem Akram  2)Rajesh Singh | Nowadays it is becoming a difficult task to distinguish wet and dry waste. The new waste management system covers several levels of enormous workforce. Every time, laborers must visit the garbage bins in the city area to check whether they are filled or not. The data communicates to the cloud server for real-time monitoring of the system. With the real-time fill level information collected via the monitoring platform, the system reduces garbage overflow by informing about such instances before they arrive |
| Smart Waste Management System. | 1) Sanjiban Charkraborty | This Waste management is one of the serious challenges of the cities, the system now used in cities, we continue to use an old and outmoded paradigm that no longer serves the entail of municipalities, Still find over spilled waste containers giving off irritating smells causing serious health issues and atmosphere impairment. |
| Smart Solid Waste Management. | 1) Mohd Helmy Abd Wahab | At the time of trash disposal, the material to be recycled could be identified using RFID technology. |
| Analysis of Load cell. | 1) Ranjeet Kumar  2) Sandeep Chhabra | Load Cells 4.1 General Load Cell related information A load cell is meant to measure the size of a mass but actually is a force sensor which transforms force into an electrical signal. The load cell needs the earth gravity to work. Every mass is attracted by the earth gravimetric field, that force is named “load”. |
| Smart Waste Management using Wireless Sensor Network | 1. Tarandeep Singh 2. Rita Mahajan 3. Deepak Bagai | In most of the places, garbage bins are not cleaned at periodic intervals, giving a hygienic issue. Thus, a system to manage bins, by using intelligent bins, gateway and remote base station is created. But this system is prone to attacks from hackers and complexity to build it is very high. |
| Smart Waste Management for Green Environment | 1. T. P. Fei | The system is based on Bootstrap platform. This system works on the waterfall methodology which has 4 crucial phases: planning and analysis, system design, system implementation and system testing. Using this system, operators can get the information regarding collection from trash bins. The limitations of this approach are that the resultant product has a short life and uniformity is lost after a certain period. |

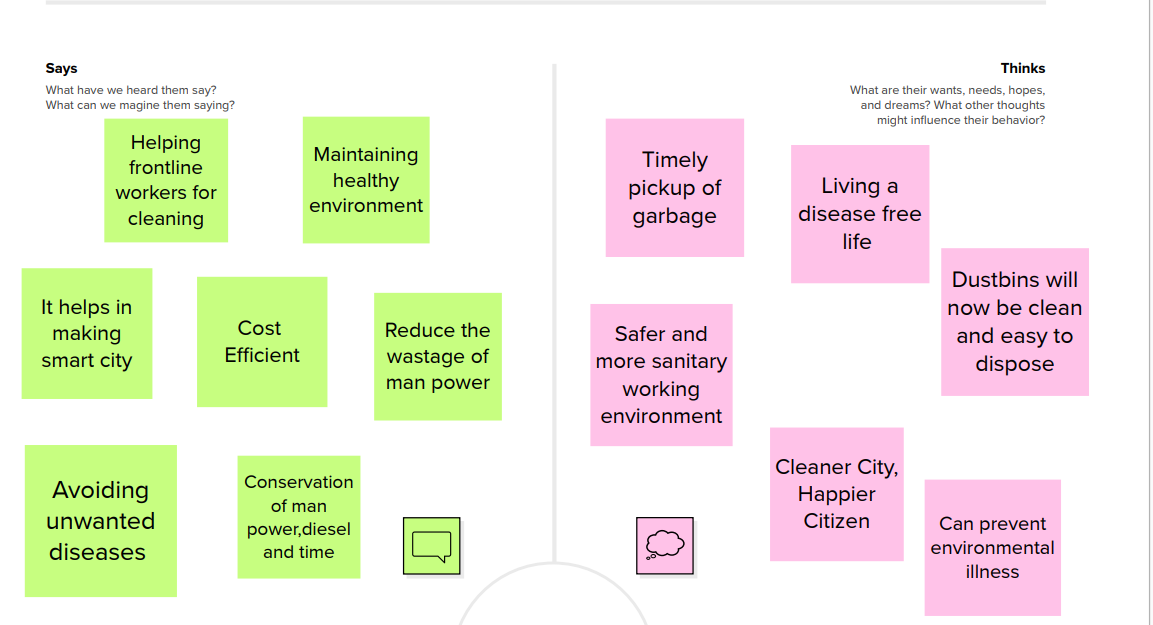
* 1. **Problem Statement Definition**

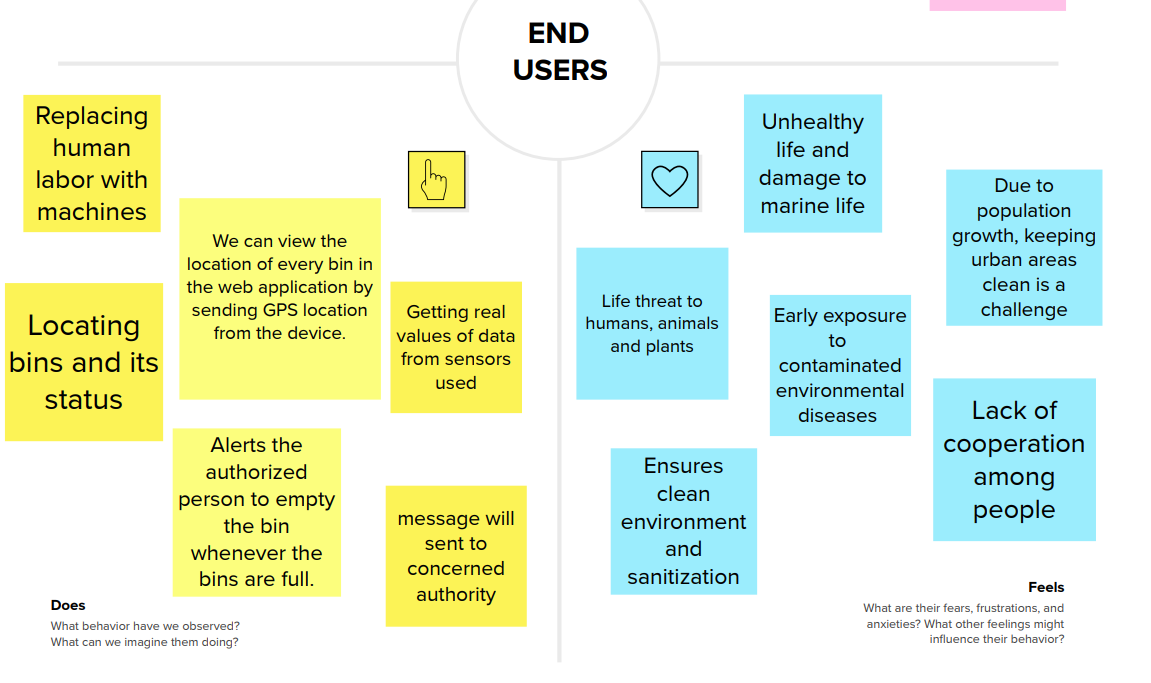




**CHAPTER 3: IDEATION & PROPOSED SOLUTION**

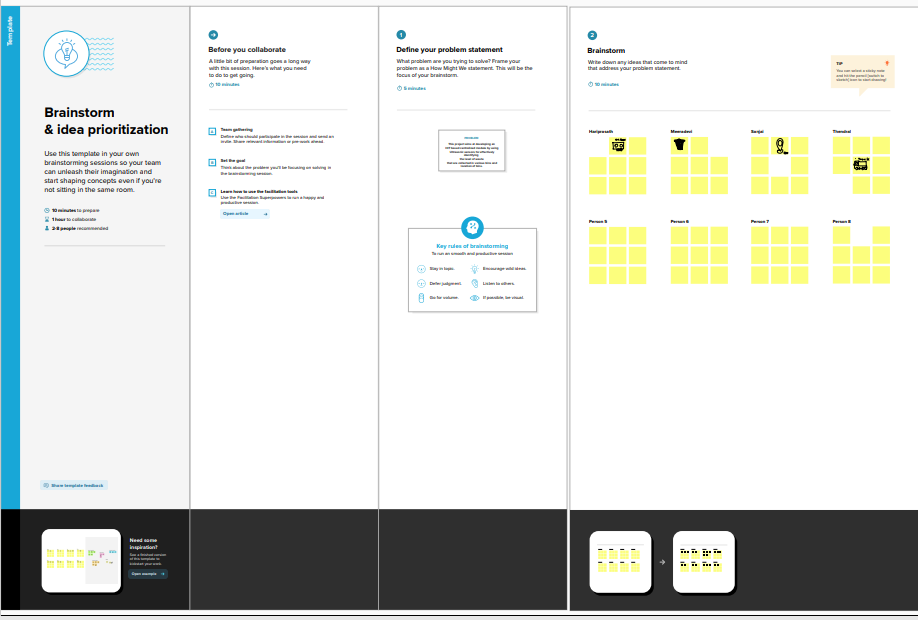
**3.1 Empathy Map Canvas**



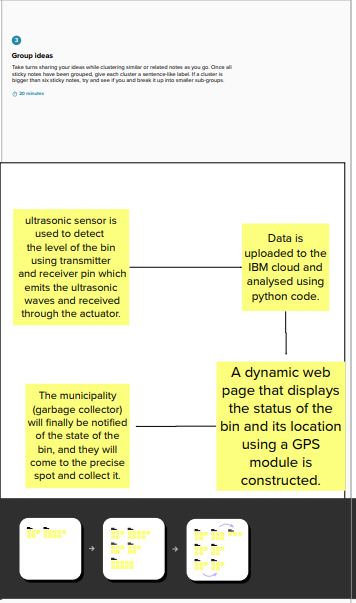


**3.2 Ideation & Brainstorming**

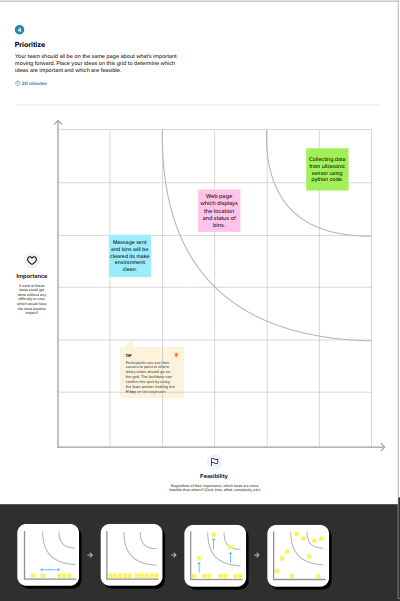
**3.2.1 Brainstorm by team members**



**3.2.2 Group ideas**



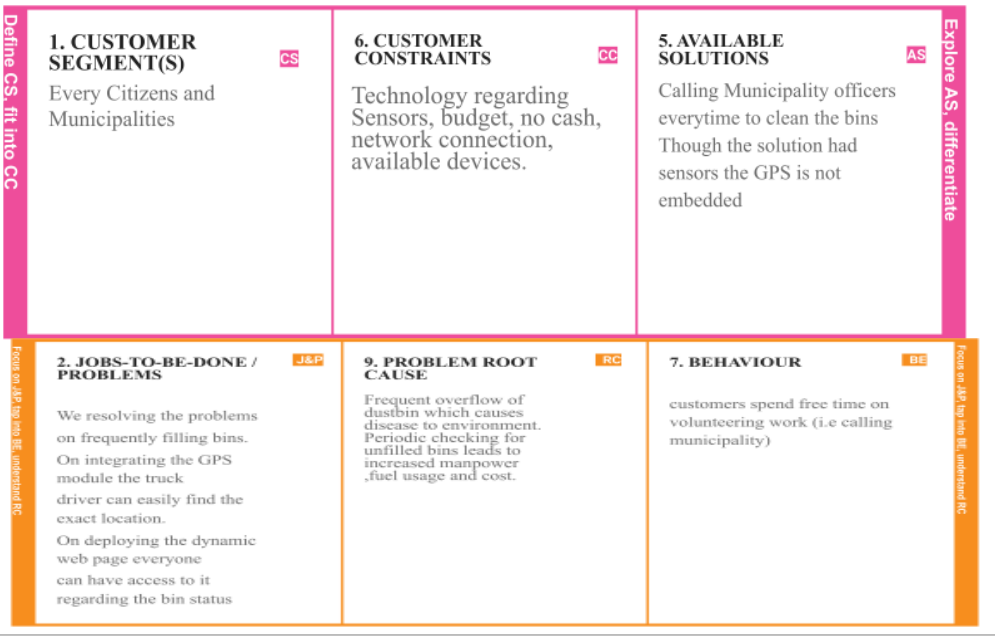
**3.2.3 Prioritize**

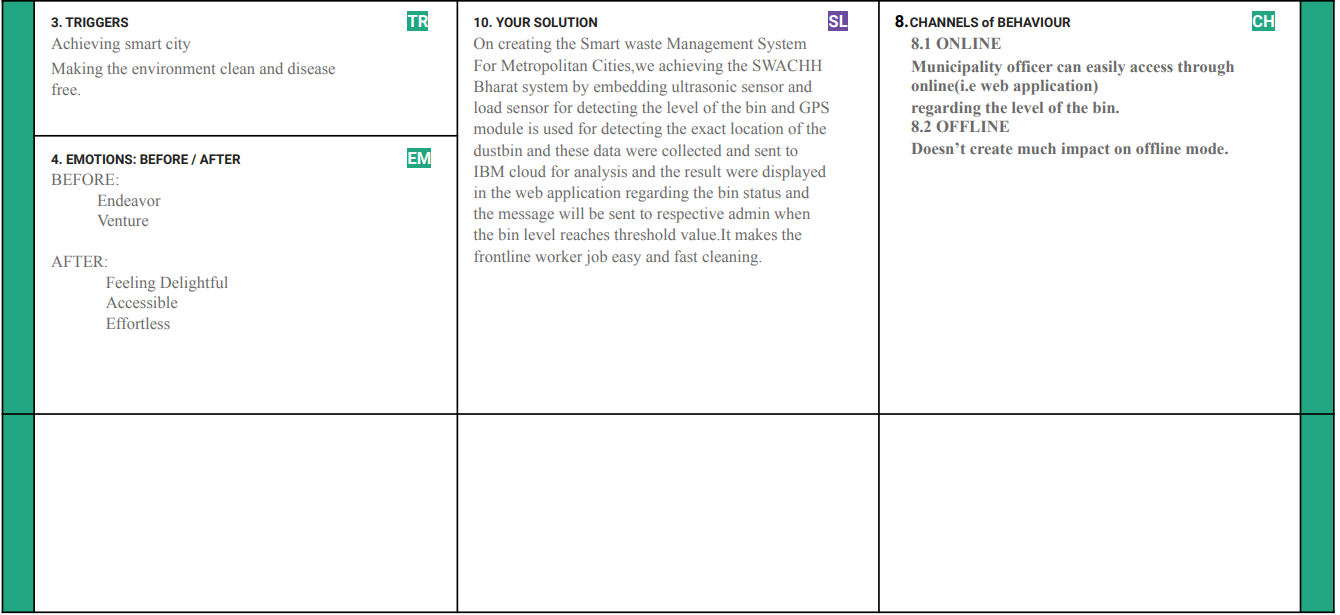


**3.3 Proposed Solution**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Parameter** | **Description** |
|  | Problem Statement (Problem to be solved) | This project aims at developing an IOT based centralized module by using ultrasonic and weight sensors to accurately determine the level and weight of waste, as well as its location using a GPS module, and to alert the administrator about the status of the bin. |
|  | Idea / Solution description | * Wastes are collected in trash bins. * Each and every trash bin is embedded with Ultrasonic and weight sensor. * Based on the waste overflow, once it crosses the particular threshold it sent the signal to the centralized module. * The centralized module will interact with the dashboard to indicate the level of each bin. |
|  | Novelty / Uniqueness | * In traditional waste management system, the corporation and the people are involved in manual collection of wastes without identifying the exact and shortest path. * However, our proposed method uses a GPS module to locate the bin, which minimizes the workload for front-line labor. |
|  | Social Impact / Customer Satisfaction | * This project helps in reducing manual collection of wastes from the bins which are located in various locations. * It diminishes the need for frontline workers to regularly check the trash can. * Our effort attempts to create an atmosphere free from infections. * By providing a clean environment, it satisfies the customer (citizens). |
|  | Business Model (Revenue Model) | * By offering a GPS to pinpoint the location, the need for human labor is reduced. * Waste collection costs are decreased. * Fuel expenditure on the garbage collection truck is kept to a minimum. |
|  | Scalability of the Solution | Dynamic Web Application is available for all the time. |

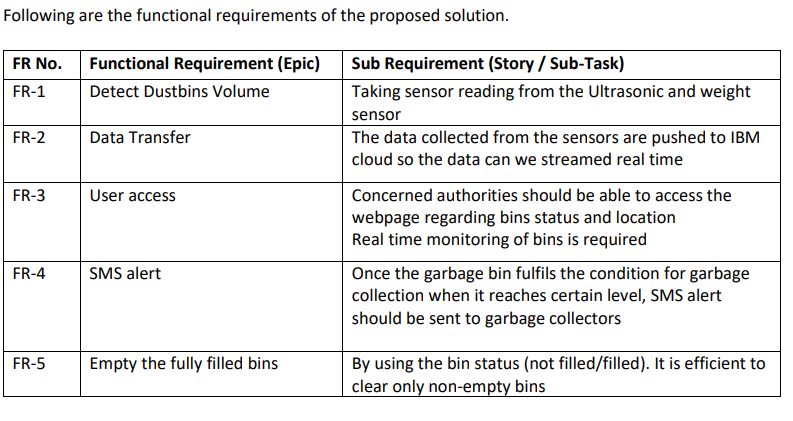
**3.4 Proposed Solution fit**



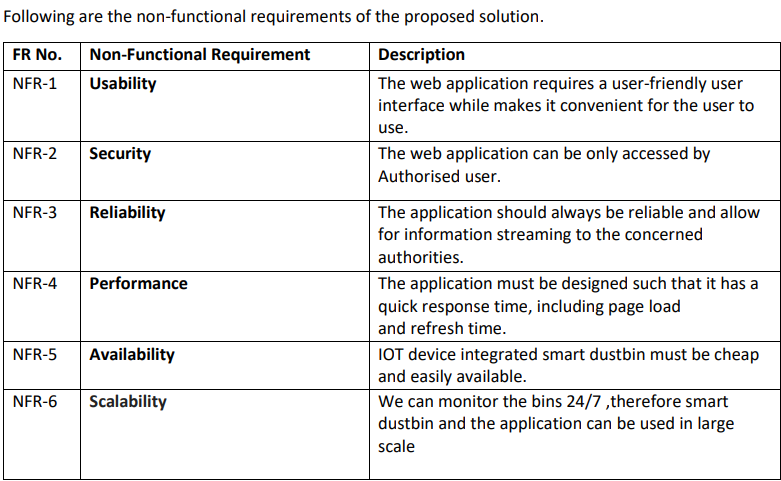


**CHAPTER 4: REQUIRMENT ANALYSIS**

**4.1 Functional Requirements**



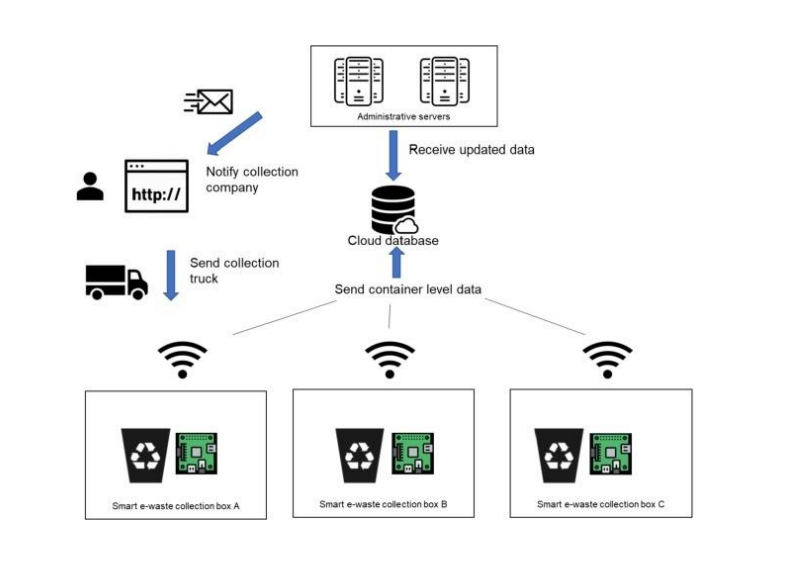
**4.2 Non-functional Requirements**



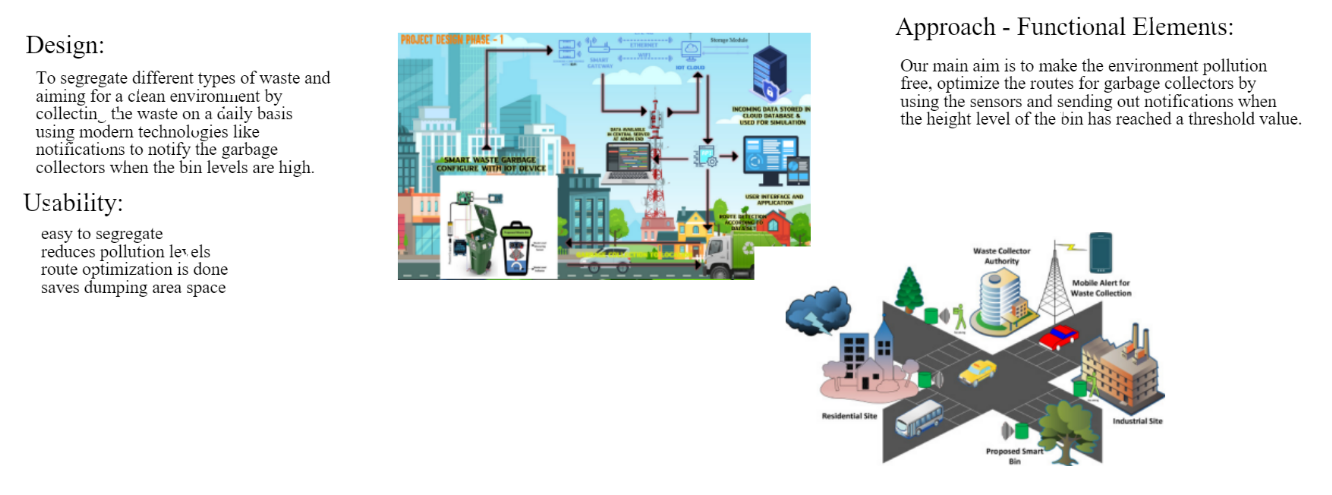
**CHAPTER 5: PROJECT DESIGN**

* 1. **Data Flow Diagram:**

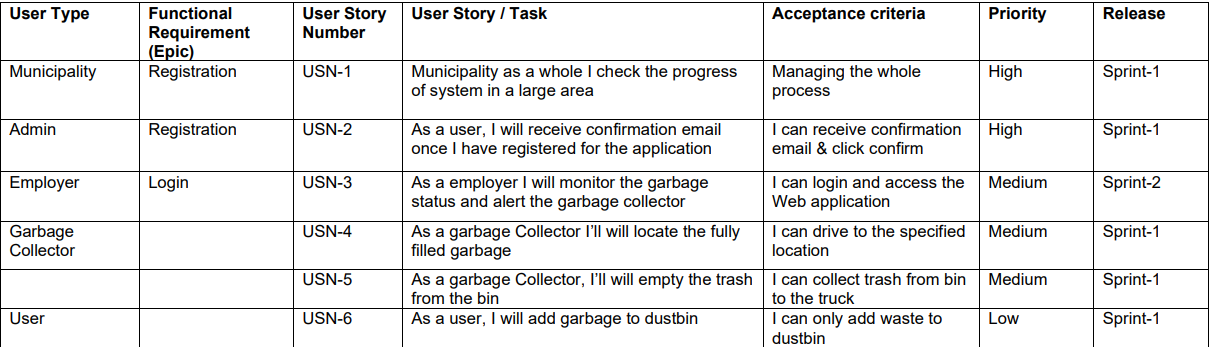
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



**5.2 Solution and Technical Architecture**

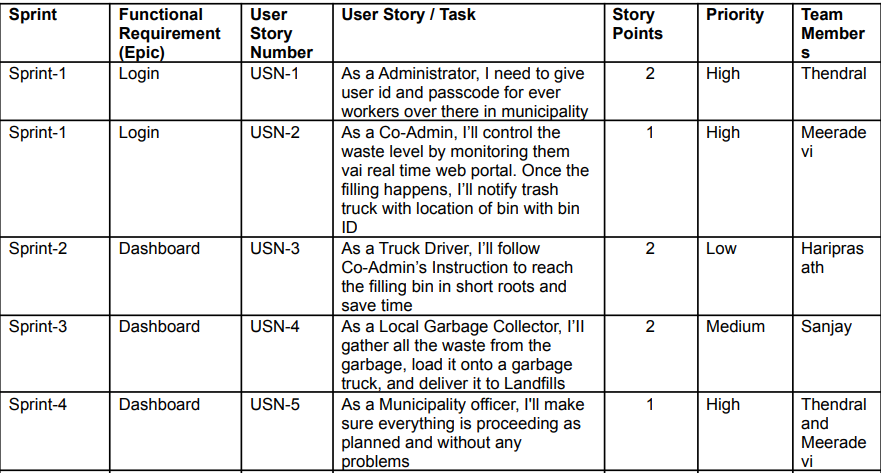
****

* 1. **User Stories**

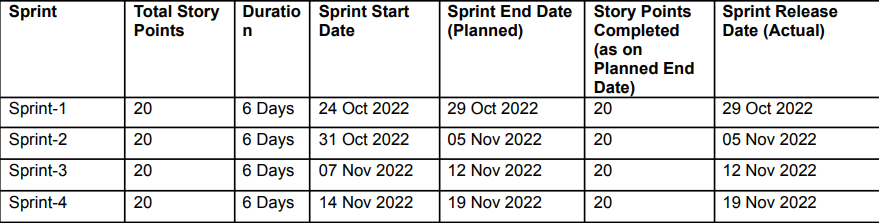


**CHAPTER 6: PROJECT PLANNING & SCHEDULING**

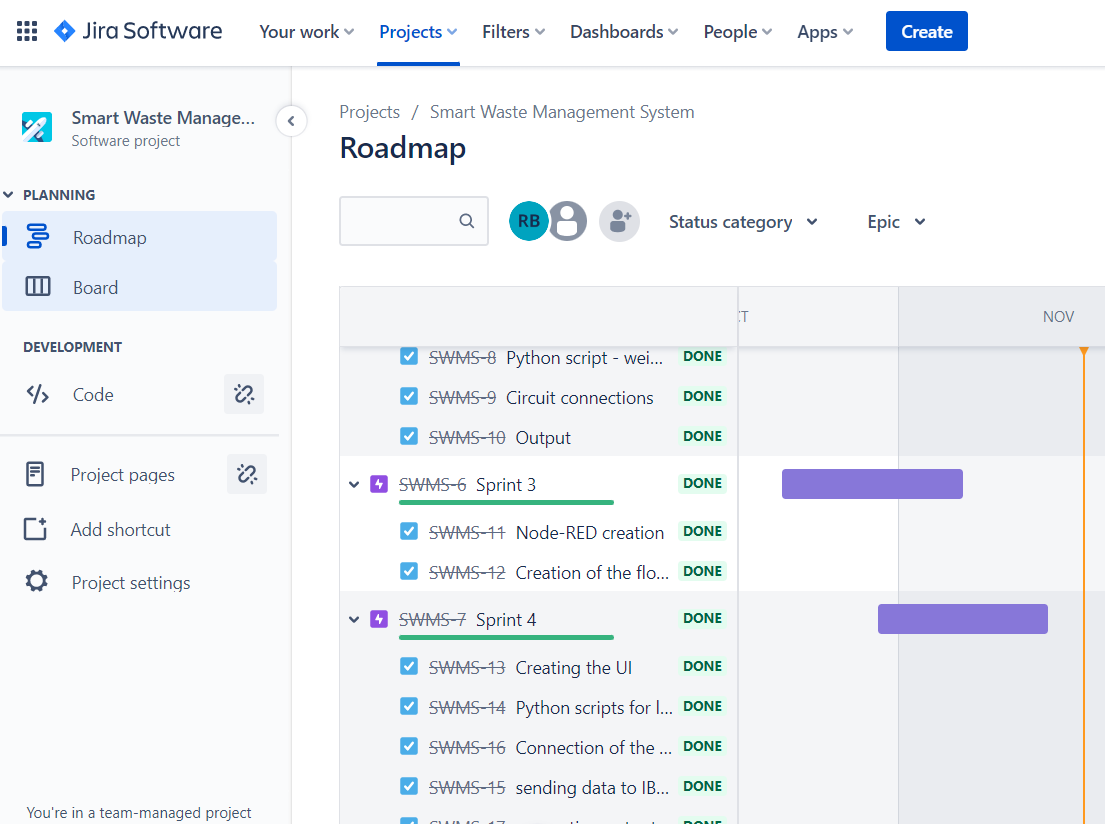
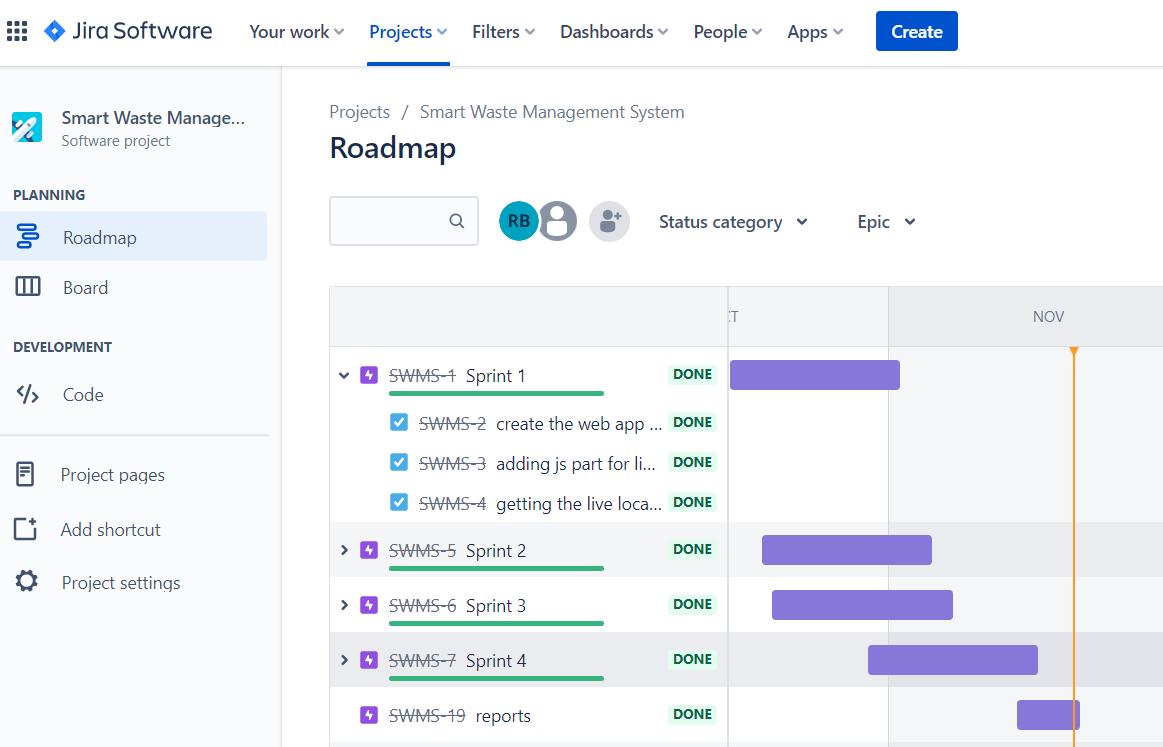
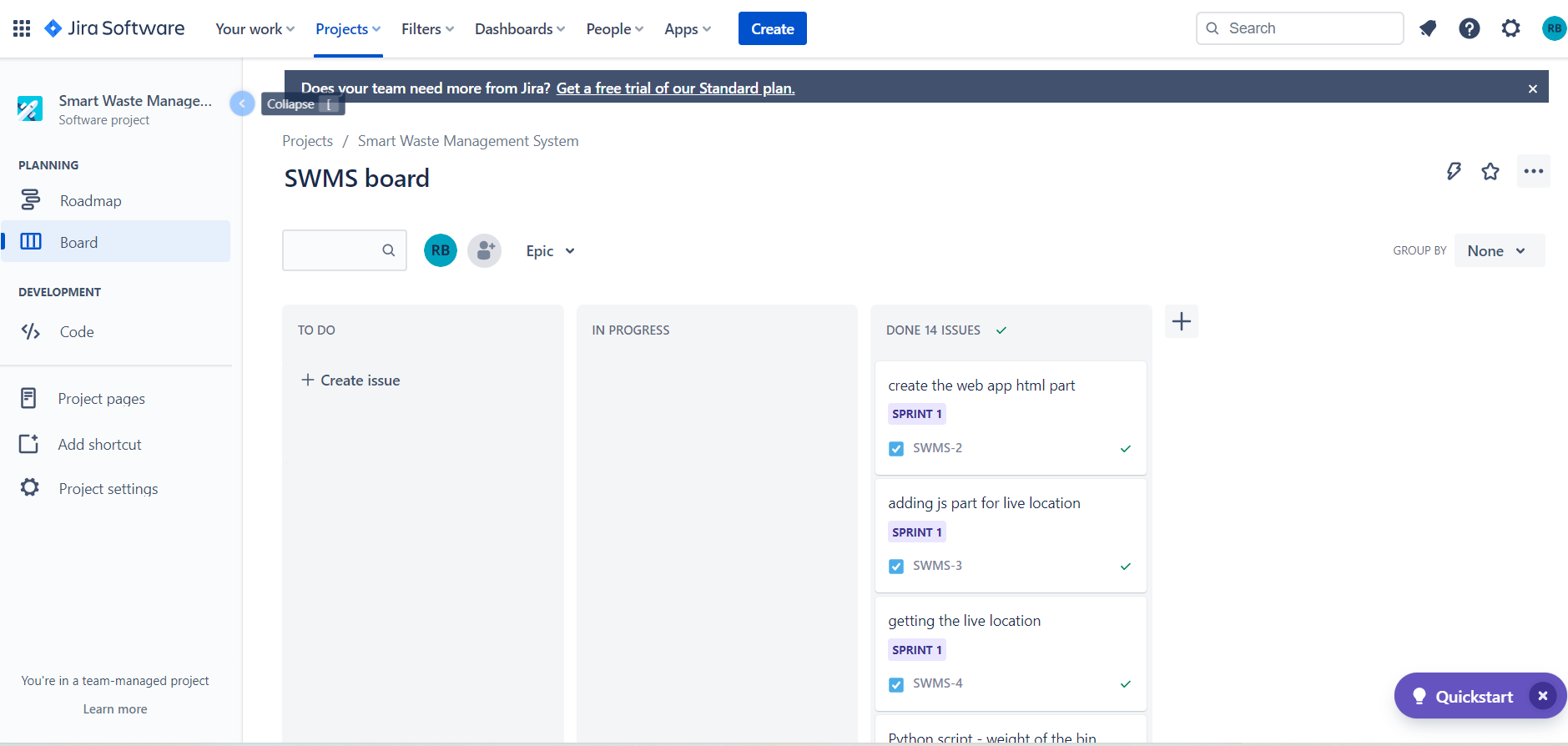
**6.1 Sprint Planning & Estimation:**



**6.2 Sprint Delivery Schedule:**



**6.3 Reports from JIRA:**



**CHAPTER 7: CODING AND SOLUTIONING**

**7.1 Feature 1:**

The main and first feature of the smart waste management is to get the live location of anyone who access the website for putting out a request for garbage collection in their locality. The live location is obtained as a result of the below 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>Garbage Management System</title>

<link rel="icon" type="image/x-icon" href="/Images/DUMPSTER.png">

<link href="style.css" rel="stylesheet" type="text/css" />

<script src="https://www.gstatic.com/firebasejs/8.10.1/firebase-app.js"></script>

<script src="https://www.gstatic.com/firebasejs/8.10.1/firebase-database.js"></script>

<script>

const firebaseConfig = {

apiKey: "AIzaSyBlGTTyxZGOhJhG5OkChfMh61gy9LxmJK8",

authDomain: "smartdustbin-3c2bc.firebaseapp.com",

projectId: "smartdustbin-3c2bc",

storageBucket: "smartdustbin-3c2bc.appspot.com",

messagingSenderId: "995689523667",

appId: "1:995689523667:web:31c032808b118c81732726",

measurementId: "G-YFGT614LK6"

};

firebase.initializeApp(firebaseConfig)

</script>

<script defer src="database.js"></script>

</head>

<body style="background-color:#fff;">

<script src="map.js"></script>

<div id="map\_container">

<h1 id="live\_location\_heading">GARBAGE BIN LOCATION</h1>

<div id="map"></div>

<div id="alert\_msg">BIN INFORMATION!</div>

</div>

</div>

<center><a

href="https://www.google.com/maps/place/Coimbatore+Institute+of+Technology/@11.0283312,77.0250919,17z/data=!3m1!4b1!4m5!3m4!1s0x3ba858979f85136b:0xdd9ca28d3c37cf8a!8m2!3d11.0283259!4d77.0272806"

type="button" class="btn btn-dark">GARBAGE BIN</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);

    }

  );

}

**7.2 Feature 2:**

In this part, the filled level of the bin is measured with the help of IBM IOT Watson platform devices, IBM Cloud interface and Node-RED is used for creating the dashboard nodes that helps us create a UI to display the distance, that is, the fill level of the bin. It also intimates the location of the bin with the fill level and alerts the collection authority if the fill level goes beyond a threshold value.

**Code to evaluate the level of the garbage in bin:**

**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 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=11.007403,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 device

deviceCli.disconnect()

**bin4.py:**

import requests

import json

import ibmiotf.application

import ibmiotf.device

import time

import random

import sys

# watson device details

organization = "73ffyv"

devicType = "BIN4"

deviceId = "BIN4ID"

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=11.453306,long=77.426024):

print("Seethammal Colony, Gobichittipalayam")

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()

**7.3 Feature 3:**

An additional feature added to the smart waste management system is to measure the weight of the bin using hx711 load cell. The weight of the bin is the output of the below code.

**Measuring the weight of the garbage bin:**

**main.py:**

from hx711 import HX711

hx = HX711(5,4,64)

print(1)

while True:

    hx.tare()

    read = hx.read()

    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 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)

**CHAPTER 8: TESTING**

**8.1 Test cases:**

**Unit testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case no.** | **Sensor/Stage** | **Input** | **Expected output** | **Obtained output** | **Status** |
| 1. | Ultrasonic | Garbage level in bin  i)Null  ii)Full  iii)Range in % | Correct level or distance | As expected | Pass |
| 2. | ESP – 32 | Microcontroller to process the input data | To collect the data from sensor | As expected | Pass |
| 3. | Load cell | To measure mechanical force | Calculate the force due to the bin weight | As expected | Pass |
| 4. | Gauge | To display the tares | Display the level for collection | As expected | Pass |
| 5. | HX710 | Weight of the bin  (in kg) | Measure the weight | As expected | Pass |

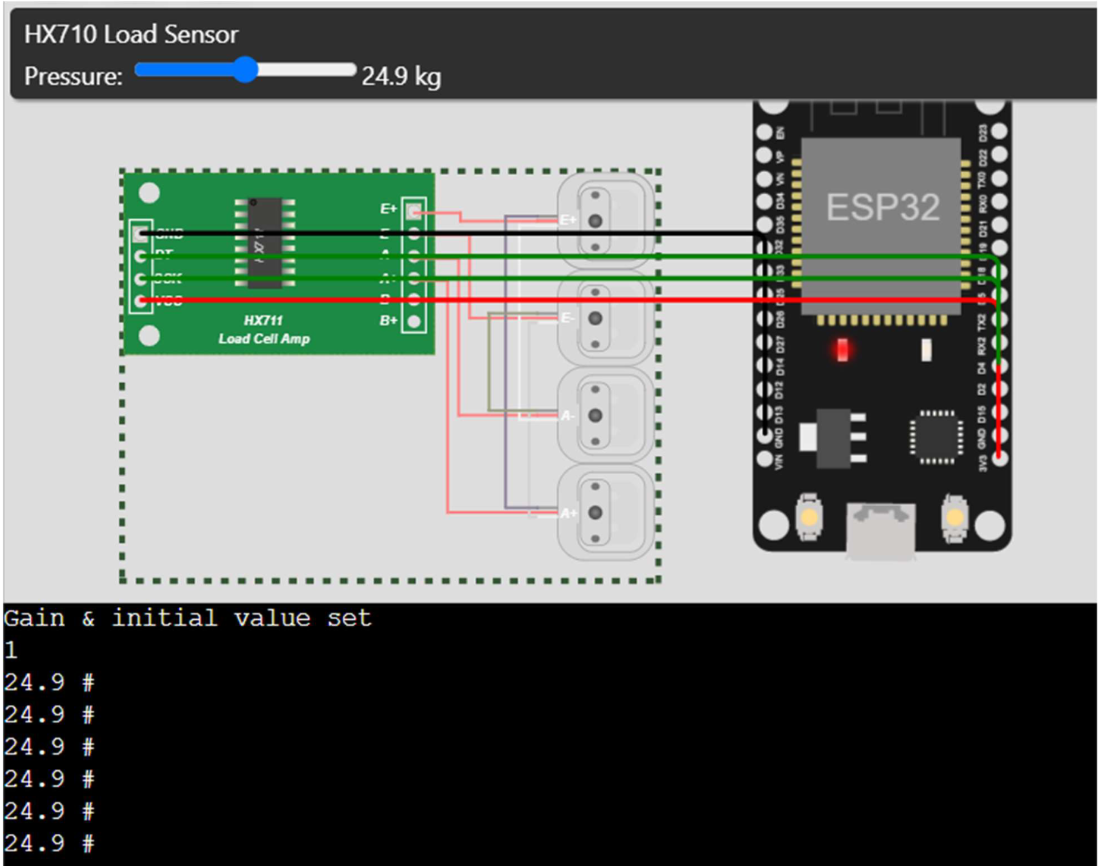
**8.2 User Acceptance testing**

Acceptance testing - is the final phase of product testing prior to public launch. A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.

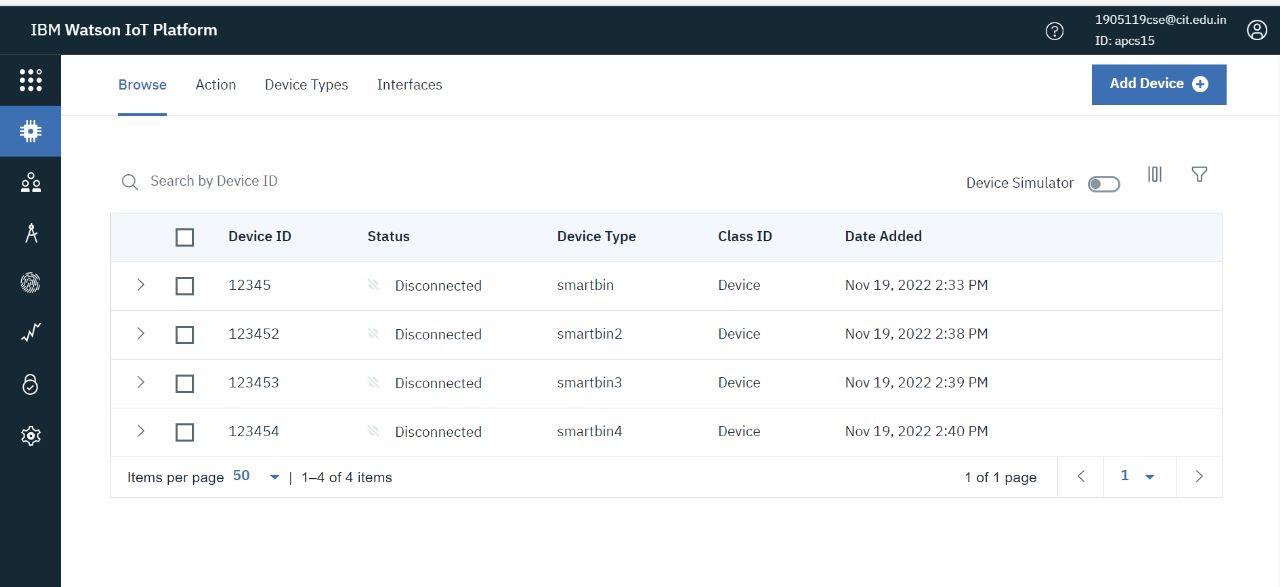
**CHAPTER 9: RESULTS**

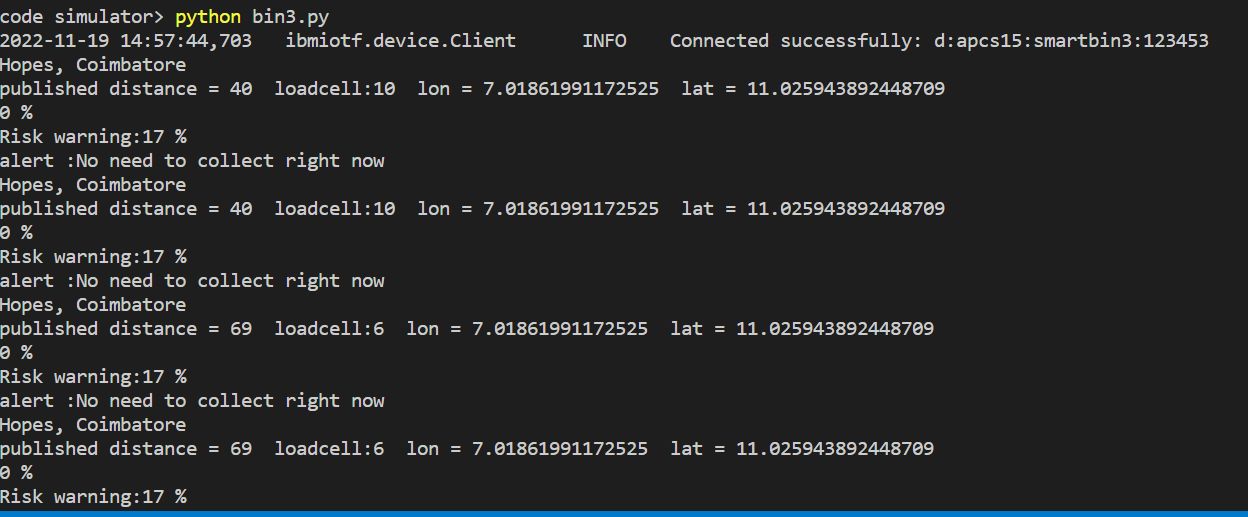
**Outputs:**

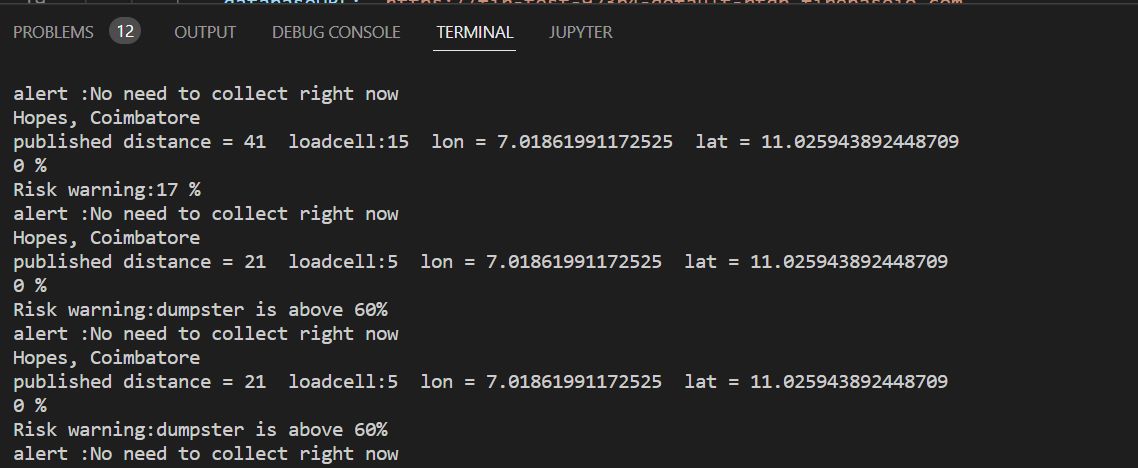
Calculating the weight of the bin using hx711 load sensor:

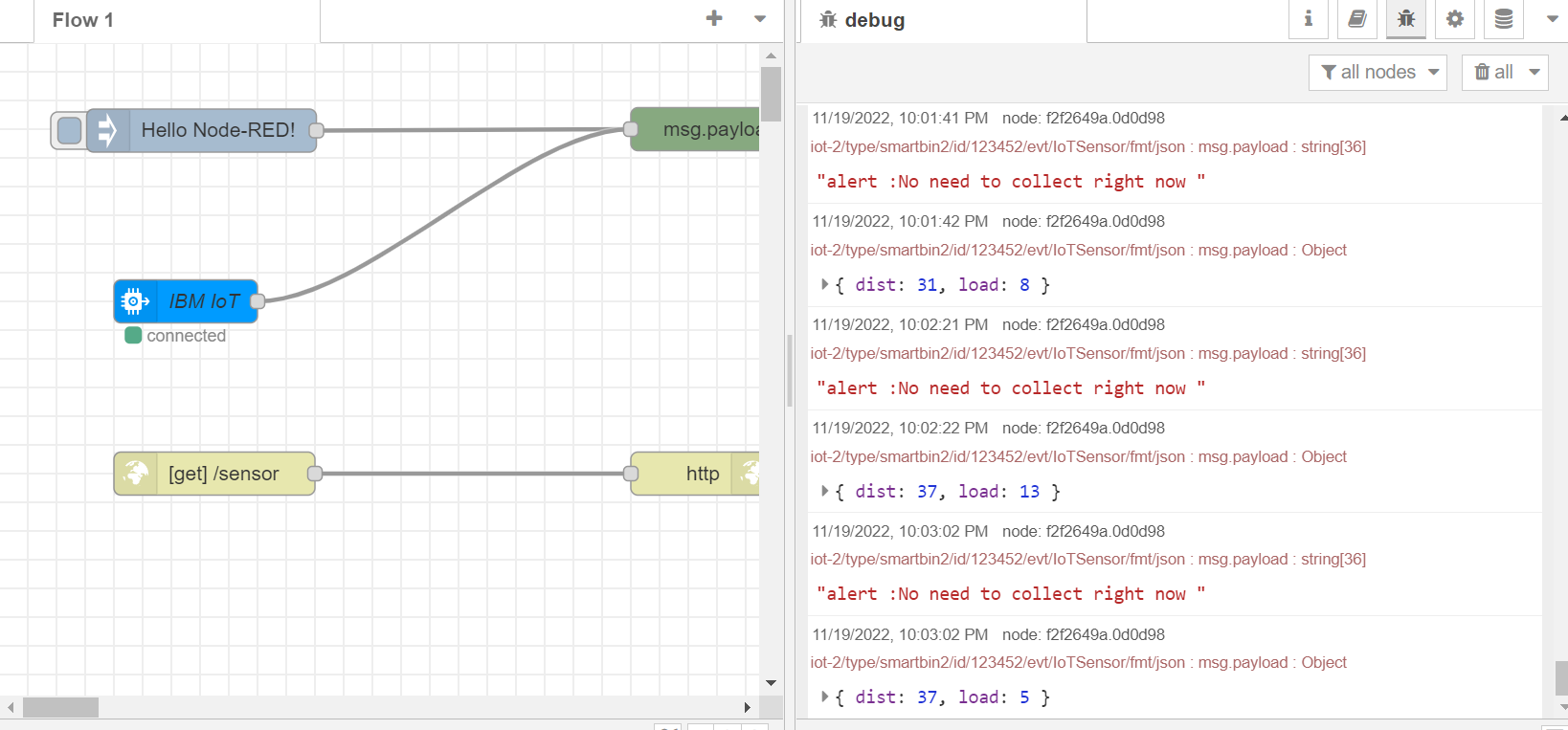


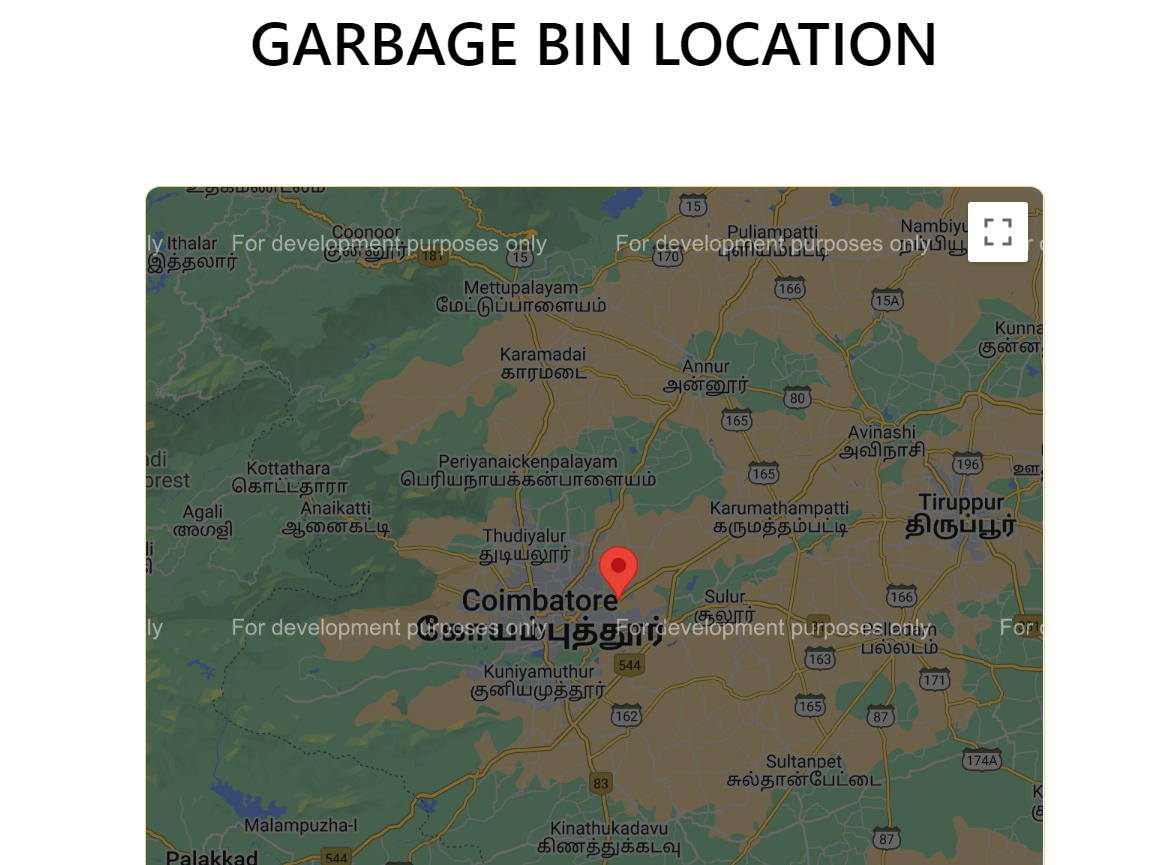
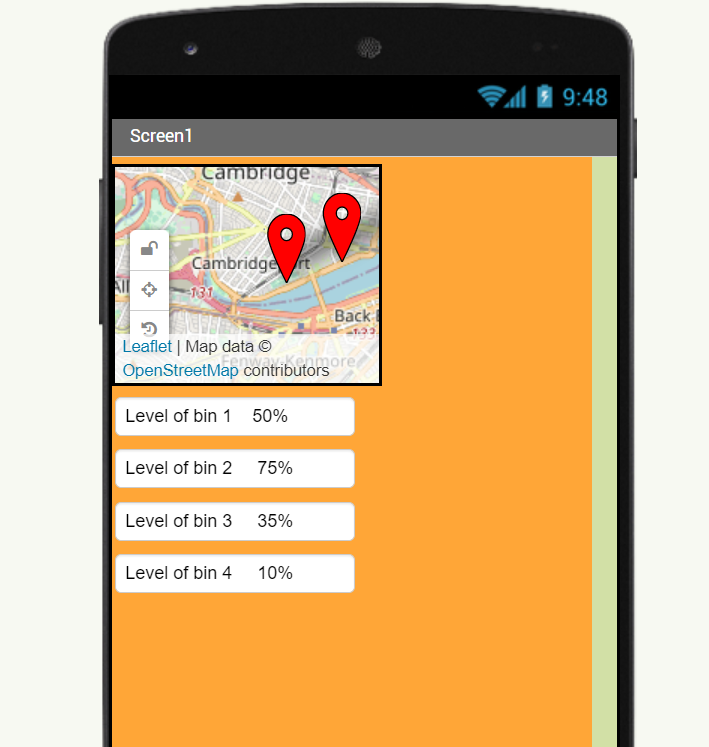
Measuring the weight of the bin and displaying the output using the nodes dashboard UI and python script, using IBM Watson as the intermediate to connect the devices:

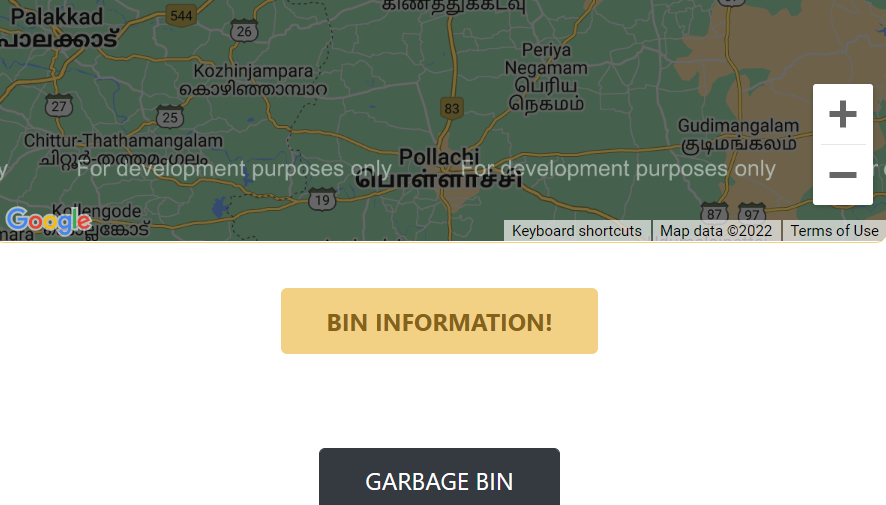


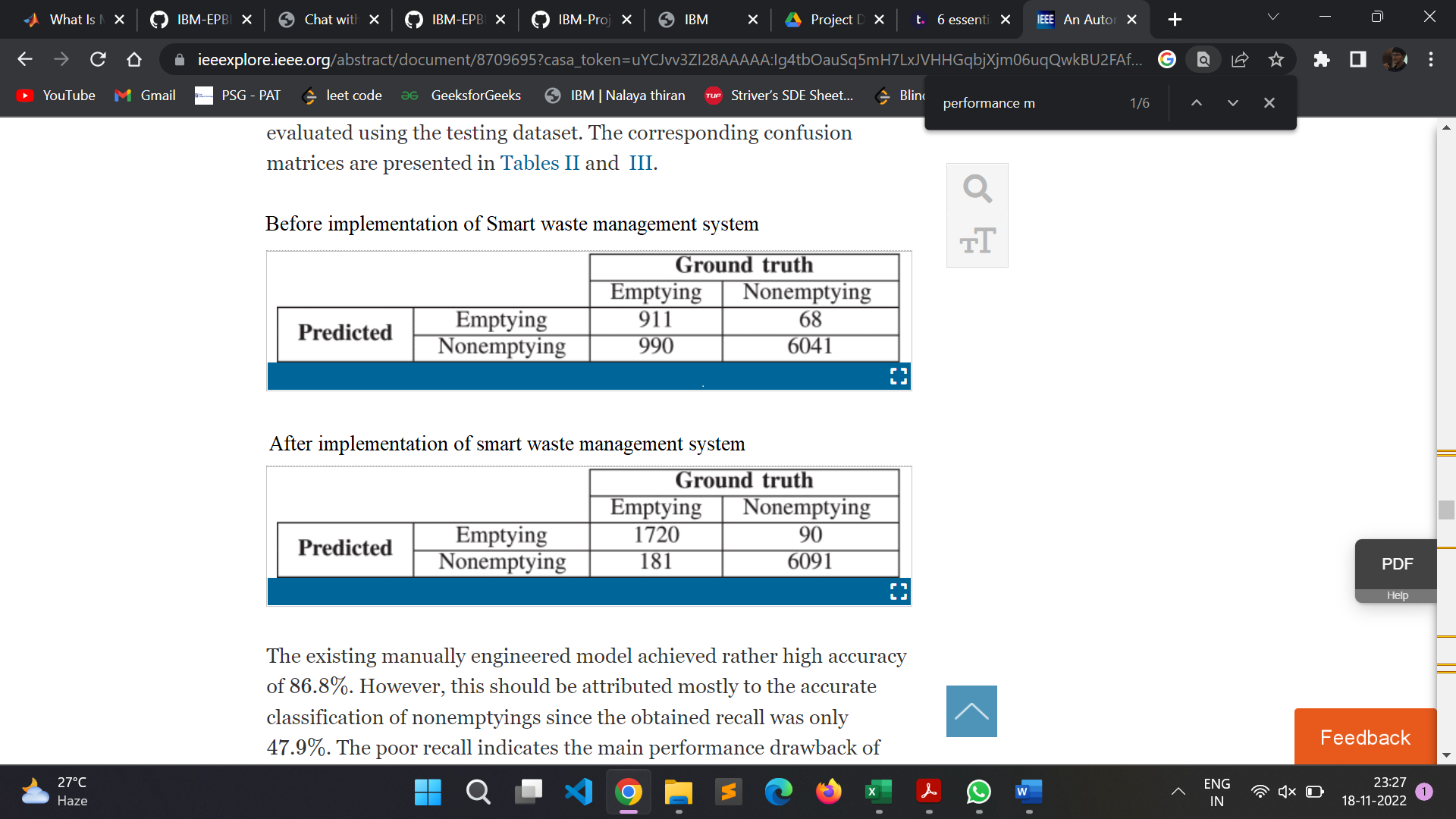








****

****

**CHAPTER 10: ADVANTAGES AND DISADVANTAGES**

**10.1 Advantages:**

* Intelligent compaction of waste by monitoring fill level in real-time using sensors.
* It keeps our surroundings clean and keeps free from bad odour.
* Reduces manpower requirement to handle the garbage collection
* Emphasizes of healthy environment and keep the cities cleaner and more beautiful.
* It reduces infrastructure, operating and maintenance costs by upto 30%.
* Increases recycling rate of waste.

**10.2 Disadvantages:**

* Initial large-scale implementation takes cost.
* System requires more number waste bins for separate waste collection.
* Wireless technologies used should have proper connections as they have shorter range and lower data speed
* Training programs should be provided to people involving in the ecosystem of smart waste management.
* Sensors may encounter damage so it should be kept under protective ambience to prevent the damage.
* Replacement of sensors require knowledgeable people and thus acknowledgement of malfunction of sensor.

**CHAPTER 11: CONCLUSION**

Improper disposal and improper maintenance of domestic waste create issues in public health and environment pollution thus this paper attempts to provide practical solution towards managing the waste collaborating it with the use of IOT. by using the smart waste management system, we can manage waste properly we are also able to sort the Bio-degradable and non-Biodegradable waste properly which reduces the pollution in the environment. Various waste management initiatives taken for human well-being and to improve the TWM practices were broadly discussed in this chapter. The parameters that influence the technology and economic aspects of waste management were also discussed clearly. Different types of barriers in TWM, such as economic hitches, political issues, legislative disputes, informative and managerial as well as solutions and success factors for implementing an effective management of toxic organic waste within a globular context, were also discussed giving some real examples. The effect of urbanization on the environmental degradation and economic growth was also discussed. The proposed system will help to overcome all the serious issues related to waste and keep the environment clean.

**CHAPTER 12: FUTURE WORK**

Based on the real-time and historical data collected and stored in the cloud waste collection schedules and routes can be optimized. Predictive analytics could be used to make decisions ahead of time and offers insight into waste bin locations. Graph theory optimization algorithms can be used to manage waste collection strategies dynamically and efficiently. Every day, the workers can receive the newly calculated routes in their navigation devices. The system can be designed to learn from experience and to make decisions not only on the daily waste level status but also on future state forecast, traffic congestion, balanced cost-efficiency functions, and other affecting factors that a priori humans cannot foresee.

Garbage collectors could access the application on their mobile phone/tablets using the internet. Real-time GPS assistance can be used to direct them to the pre-decided route. As they go collecting the garbage from the containers, the management is also aware of the progress as the vehicle, as well as the garbage containers, are traced in real-time. The management staff gets their own personalized administration panel over a computer/tablet which gives them a bird eye view over the entire operations.

An alternative solution using image processing and camera as a passive sensor could be used. But, the cost of those image processing cameras is higher as compared to the ultrasonic sensors, which leads to high solution implementation cost.

**CHAPTER 13: APPENDIX**

**13.1 Source 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);

    }

  );

}

**style.css:**

html, body

{

height: 100%;

margin: 0px;

padding:0px;

}

#container

{

display: flex;

flex-direction: row;

height: 100%;

width: 100%;

position: relative;

}

#logo\_container

{

height: 100%;

width: 12%;

background-color: #C5C6D0;

display: flex;

flex-direction: column;

vertical-align: text-bottom;

}

.logo

{

width:70%;

margin: 5% 15%;

/\* border-radius: 50%; \*/

}

#logo\_3

{

vertical-align: text-bottom;

}

#data\_container

{

height: 100%;

width: 20%;

margin-left: 1%;

margin-right: 1%;

display: flex;

flex-direction: column;

}

#data\_status

{

height:60%;

width:8%;

margin:7%;

background-color: #691F6E;

display: flex;

flex-direction: column;

border-radius:20px;

}

#load\_status

{

background-image: url("/Images/KG.png");

background-repeat: no-repeat;

background-size: 170px;

background-position: left center;

}

#cap\_status

{

background-image: url("/Images/dust.png");

background-repeat: no-repeat;

background-size: 150px;

background-position: left center;

}

.status

{

width: 80%;

height: 40%;

margin:5% 10%;

background-color:#185adc;

border-radius:20px;

display: flex;

justify-content: center;

align-items: center;

color: white;

font-size: 60px;

}

.datas

{

width:86%;

margin:2.5% 7%;

height:10%;

background: url(water.png);

background-repeat: repeat-x;

animation: datas 10s linear infinite;

box-shadow: 0 0 0 6px #98d7eb, 0 20px 35px rgba(0,0,0,1);

}

#map\_container

{

height: 100%;

width: 100%;

display: flex;

flex-direction: column;

background-color:white;

}

#live\_location\_heading

{

margin:5%;

text-align: center;

color: black;

}

#map

{

min-height: 600px;

min-width: 600px;

max-width: 70%;

margin: 10px auto ;

border: 1px solid #f2d184;

border-radius: 10px;

}

#alert\_msg

{

max-width: 500px;

margin: 20px auto !important;

margin:4%;

background-color:#f2d184;

border-radius:4px;

display: flex;

justify-content: center;

align-items: center;

color: #866118;

font-size: 16px;

padding: 10px 30px;

font-weight: bold;

cursor: pointer;

}

.lat

{

margin: 0px;

font-size:0px;

}

.btn{

padding: 8px 30px;

}

@keyframes datas{

0%

{

background-position: -500px 100px;

}

40%

{

background-position: 1000px -10px;

}

80% {

background-position: 2000px 40px;

}

100% {

background-position: 2700px 95px;

}

}

**Code to evaluate the level of the garbage in bin and intimate the collection authority with the location of the bin:**

**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 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=11.007403,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 device

deviceCli.disconnect()

**bin4.py:**

import requests

import json

import ibmiotf.application

import ibmiotf.device

import time

import random

import sys

# watson device details

organization = "73ffyv"

devicType = "BIN4"

deviceId = "BIN4ID"

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=11.453306,long=77.426024):

print("Seethammal Colony, Gobichittipalayam")

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()

**Measuring the weight of the garbage bin:**

**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 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)

**13.2 Project Links:**

Github Link:

<https://github.com/IBM-EPBL/IBM-Project-48562-1660809505>

Demo Link:

<https://drive.google.com/file/d/1S2KQmPVb8L9CttsCIwVyrqNEa0OboyAA/view?usp=sharing>