KIT-KALAIGNARKARUNANIDHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

SMART WASTE MANAGEMENT SYSTEM FOR METROPOLITAN CITIES

TEAM ID : PNT2022TMID07334

PROJECT MEMBERS :

HEMA K

ARO CELCIYA J

ASHOK KUMAR S

GNANA SINDHUJA B

CONTENT

INTRODUCTION

1.1 Project Overview

The Internet of Things (IoT) is a concept in which surrounding objects are connected through wired and wireless networks without user intervention. In the field of IoT, the objects communicate and exchange information to provide advanced intelligent services for users.This project deals with the problem of waste management in smart cities, where the garbage collection system is not optimized. This project enables the organizations to meet their needs of smart garbage management systems. This system allows the user to know the fill level of each garbage bin in a locality or city at all times, to give a cost-effective and time-saving route to the truck drivers.

1.2 Purpose

The proposed system would be able to automate the solid waste monitoring process and management of the overall collection process using IOT (Internet of Things). In the proposed system, whenever the waste bin gets filled this is acknowledged by placing the circuit at the waste bin, which transmits it to the receiver at the desired place in the area or spot.

• In the proposed system, the received signal indicates the waste bin status at the monitoring and controlling system.

LITERATURE SURVEY

Mohammad aledhari,Ala al-Fuqaha,Mohsen Guizani(2015):Iot is enabled by the latest developments in RFID,smart sensors,communication technologies, and internet protocols,the basic premise is to have smart sensors collaborate directly without human involvement to deliver a new class of applications.the current revolution in internet,mobile, and machine to machine technologies can be seen as the first phase of the iot. In the coming years, the IoT is expected to bridge diverse technologies to enable new applications by connecting physical objects together in support of intelligent decision making. This paper starts by providing a horizontal overview of the IoT. Then, we give an overview of some technical details that pertain to the IoT enabling technologies, protocols, and applications. Compared to other survey papers in the field, our objective is to provide a more thorough summary of the most relevant protocols and application issues to enable res earchers and application developers to get up to speed quickly on how the different protocols fit together to deliver desired functionalities without having to go through RFCs and the standards specifications.

Effective waste collection with shortest path semi-static and dynamic routing

Theodoros vasileios Anagnostopoulos and arkady zaslavsky(2014):

Smart cities are the next step in human habitation. In this context the proliferation of sensors and actuators within the Internet of Things (IoT) concept creates a real opportunity for increasing information awareness and subsequent efficient resource utilization. IoT-enabled smart cities will generate new services. One such service is the waste collection from the streets of smart cities. In the past, waste collection was treated with static routing models. These models were not flexible in case of segment collapse. In this paper we introduce a semi-static and dynamic shortest path routing model enhanced with sensing capabilities through the Internet connected objects in order to achieve effective waste collection.

top-k query based dynamic scheduling for iot -enabled smart city waste collection

Sergei khoruzhniov,Arkady Zaslavsky(2015): Brilliant Cities are being planned and worked for agreeable human residence. Among administrations that Smart Urban areas will offer is the naturally well disposed waste/junk accumulation and preparing. In this paper, we inspire and propose an Internet of Things (IoT) - empowered framework engineering to accomplish dynamic waste accumulation and conveyance to handling plants or exceptional junk tips. Previously, squander accumulation was dealt with in a fairly static way utilizing traditional operations look into approach. As proposed in this paper, these days, with the multiplication of sensors and actuators, as well as solid and universal portable correspondences, the Web of Things (IoT) empowers dynamic arrangements went for advancing the waste vehicle armada measure, accumulation courses and organized waste get.

Robust waste collection exploiting cost efficiency of iot potentiality in smart cities

Alexey Medvedev(2015) :

Smart Cities constitute the future of civil habitation. Internet of Things (IoT) enable innovative services exploiting sensor data from sensors embedded in the city. Waste collection is treated as a potential IoT service which exploits robustness and cost efficiency of a heterogeneous fleet. In this paper we propose a dynamic routing algorithm which is robust and copes when a truck is overloaded or damaged and need replacement. We also incorporate a system model which assumes two kinds of trucks for waste collection, the Low Capacity Trucks (LCTs) and the High Capacity Trucks (HCTs). By incorporating HCTs we achieve reduction of the waste collection operational costs because route trips to the dumps are reduced due to high waste storage capacity of these tru cks. Finally, the proposedmodels are evaluated on synthetic and real data from the city municipality of St.Petersburg, Russia. The models demonstrate consistency and correctness

Inventory routing for dynamic waste collection

Marco schutten,Martijin Mes,Arturo Perez Rivera(2014): We consider the problem of collecting waste from sensor equipped underground containers. These sensors enable the use of a dynamic collection policy. The problem, which is known as a reverse inventory routing problem, involves decisions regarding routing and container selection. In more dense networks, the latter becomes more important. To cope with uncertainty in deposit volumes and with fluctuations due to daily and seasonal effects, we need an anticipatory policy that balances the workload over time. We propose a relatively simple heuristic consisting of several tunable parameters depending on the day of the week. We tune the parameters of this policy using optimal learning techniques combined with simulation. We illustrate our approach using a real life problem instance of a waste collection company, located in The Netherlands, and perform experiments on several other instances. For our case study, we show that costs savings up to 40% are possible by optimizing the parameters.

Capacitated location of collection sites in an urban waste management system

Giapalo ghiani,Demetri Lagana,Emanuele Manni(2012) :Urbanwaste management is becoming an increasingly complex task, absorbing a huge amount of resources, and having a major environmental impact. The design of a waste management system consists in various activities, and one of these is related to the location of waste collection sites. In this paper, we propose an integer programming model that helps decision makers in choosing the sites where to locate the unsorted waste collection bins in a residential town, as well as the capacities of the bins to be located at each collection site. This model helps in assessing tactical decisions through constraints that force each collection area to be capacitated enough to fit the expected waste to be directed to that area, while taking into account Quality of Service constraints from the citizens point of view. Moreover, we propose an effective constructive heuristic approach whose aim is to provide a good solution quality in an extremely reduced computational time.

Iot based waste management for smart city

Prakash,Prabu.v(2015): Our country is facing vast challenges in the environment due to waste generation some of them were inadequate waste collection, transport, treatment, and disposal. One of the important challenges is from its inception till its disposal. Our country can cope with the current systems by an increasing urban population with the volumes of waste, and this on the public and environmental health pollution. Unhygienic conditions are created due to the flooding of the dustbins each day. This paper is for the commenting of the challenges, barriers, and opportunities for the betterment of collection, and segregation in the field of waste management.

When it senses the nearby trash the dustbin that function automatically, and it is built through a prototype. Dustbins are placed all over the city, and deliveredwith low cost embedded method to help in tracking of the garbage bins. The Blynk appLindicates through SMS as soon as dustbin has reached its maximum level, to the unwanted management department.

Iot based smart garbage system for efficient food waste management

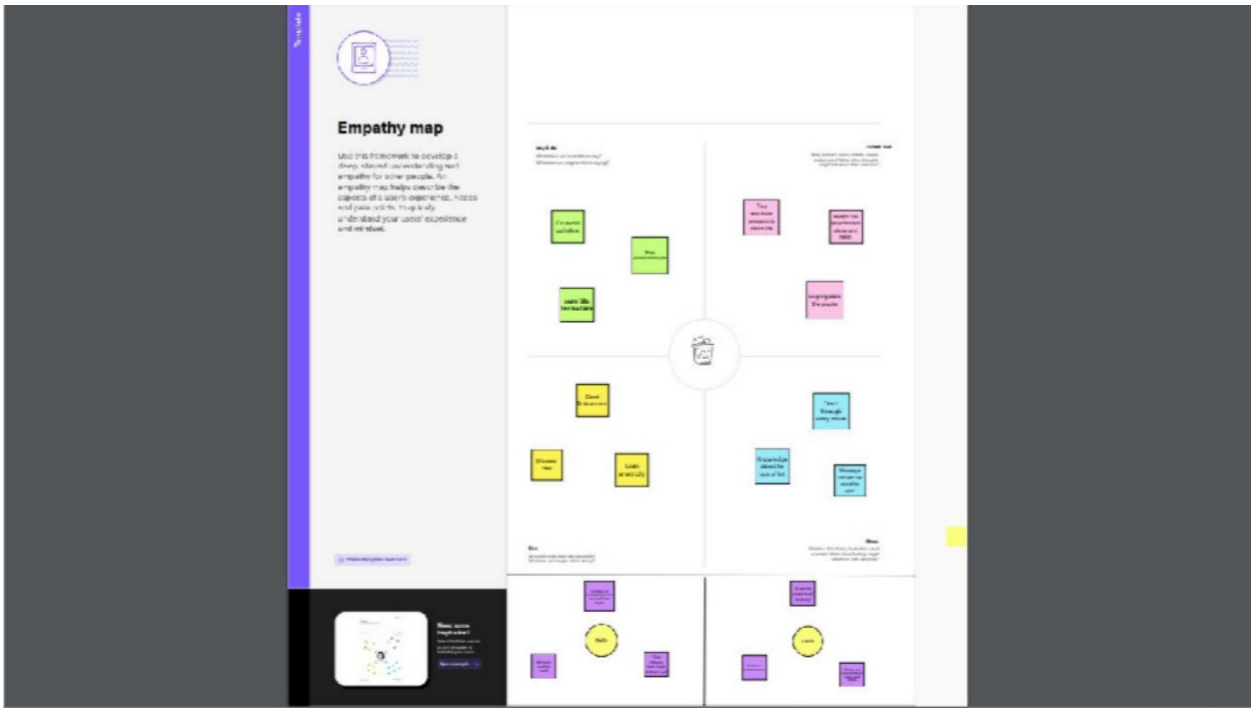
Zhongmei Zhou(2014): Owing to a paradigm shift toward Internet of Things (IoT), researches into IoT services have been conducted in a wide range of fields. As a major application field of IoT, waste management has become one such issue. The absence of efficient waste management has caused serious environmental problems and cost issues.Therefore, in this paper, an IoT-based smart garbage system (SGS) is proposed to reduce the amount of food waste. In an SGS, battery-based smart garbage bins (SGBs) exchange information with each other using wireless mesh networks, and a router and server collect and analyze the information for service provisioning.Furthermore, the SGS includes various IoT techniques considering user convenience and increases the battery lifetime through two types of energy- efficient operations of the SGBs: stand-alone operation and cooperation- based operation. The proposed SGS had been operated as a pilot project in Gangnam district, Seoul, Republic of Korea, for a one-year period. The experimentshowed that the average amount of food waste could be reduced by 33%.

Iot based smart waste management system using wireless sensor

network and embedded linux board

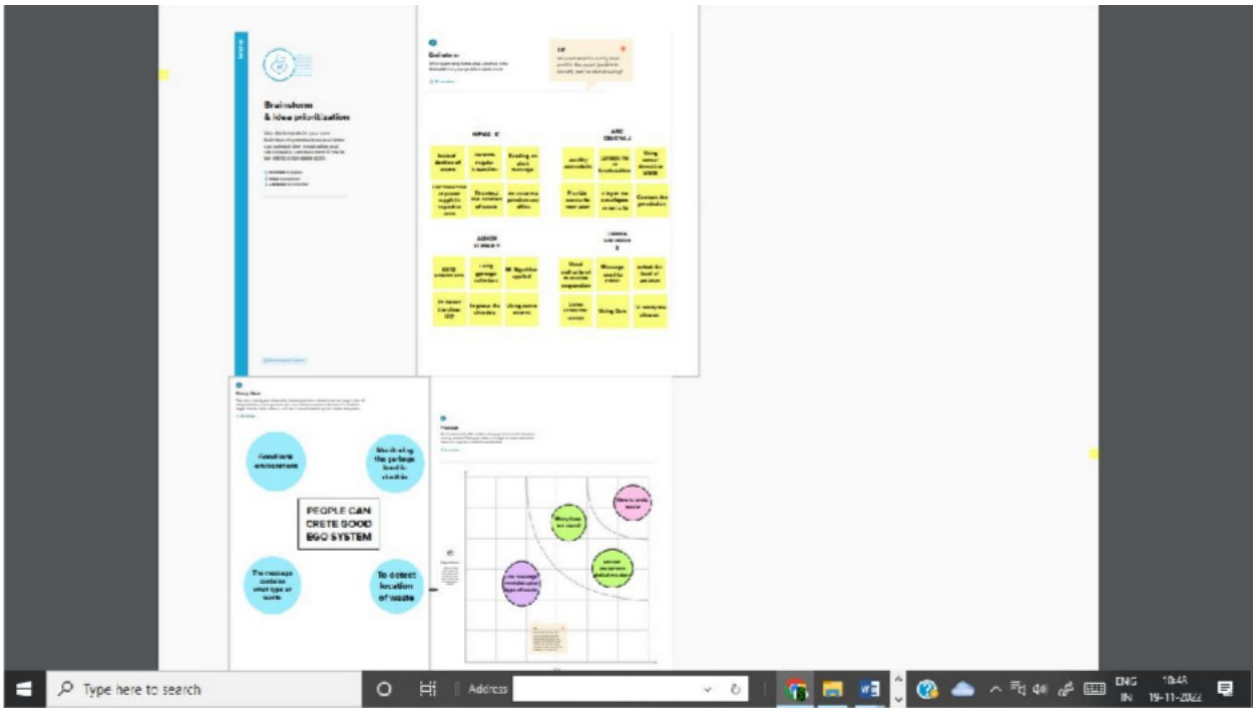
Shri S K Singh(2012): In many places, we can see that the municipal waste bins are overflowing because of it is not collected and cleaned before it fill and overflow. The consequences for this overflowing waste bin is very severe. It can lead to land pollution, air pollution, which will lead to spread of deseases to humans and animals around it. Thus there should be a alert system that can monitor the bin and can give the condition or level of filling of the waste bin to the municipality using wireless sensor network. So that the bin can be cleaned on time and the environment can be kept safe. This paper presents the smart waste management system of the garbage system that identifies fullness and level of the waste bin using the wireless sensor network(WSN). which is embedded to the Linux board to inform the information to the authorized person for the cleaning of the waste bin in time. Here we use Raspberry Pi as the embedded Linux board. The board architecture is designed based on the arm 11 microcontroller architecture.

EMPATHYMAP



.

Brainstorm & idea prioritization



PROPOSED SOLUTION

PROPOSED SOLUTION TEMPLATE

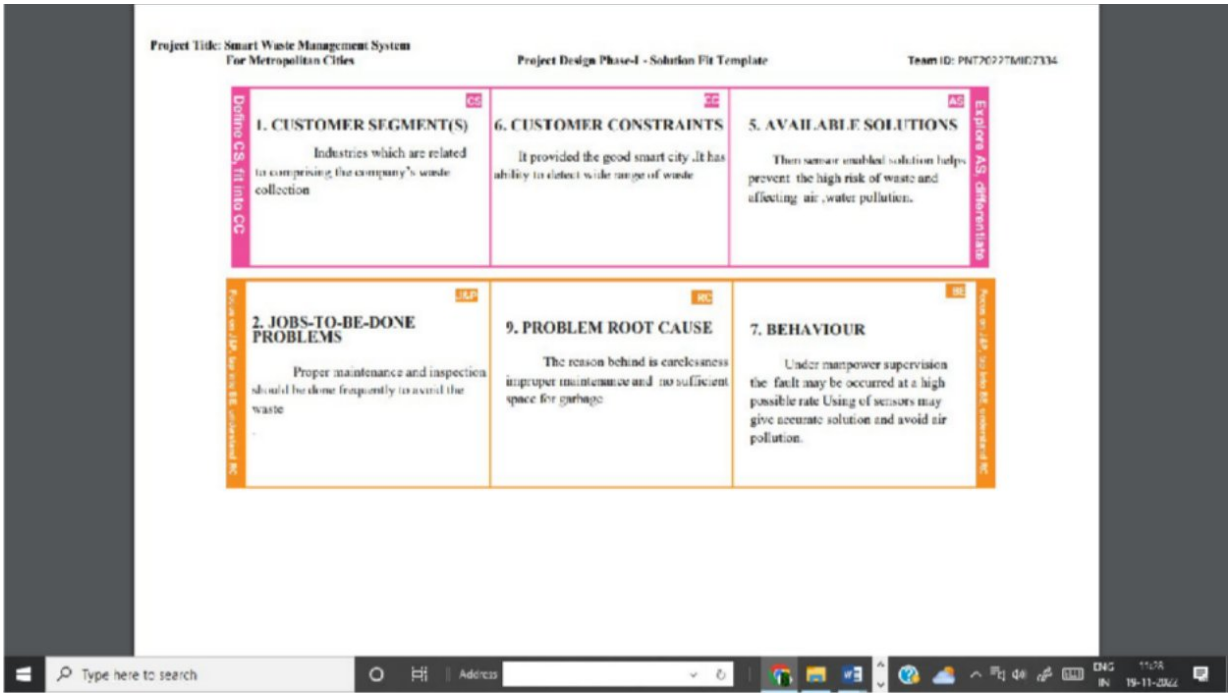
Project team shall fill the following information in proposed solution template.

SNO	PARAMETER	DESCRIPTION
1	Problem Statement (Problem to be solved)	This project deals with the problem of waste management in smart cities,where the collection system is not optimized.This project enables the organizations to meet their needs of smart garbage management system allows the person to know the fill level of eachgarbage bin in a locality or city at all times,to give a cost - effective and time saving route to the truck driver
2	Idea / Solution description	The proposed system would be able to automate the solid waste monitoring process and

		management of the over all collection process using IOT(Internet of Things). The Proposed system consists of main subsystem namely Smart Trash System(STS) and Smart Monitoring and Controlling Hut(SMCH). In the proposed system, the received signal indicates the waste bin status at the monitoring and controlling system.
3	Novelty / Uniqueness	We are going to establish SWM in our college but the real hard thing is that cleaner do not know to operate these thing practically so here our team planned to build a wrist band to them, that indicate via light blinking when the dustbin fill and this is uniqueness we made here beside from project constrain.
4	Social Impact / Customer Satisfaction	From the public perception s worst impacts of present solid waste disposal practices are seen direct social impacts such as neighbourhood of landfills to communities breeding of pests and loss in property values
5	Business Model (Revenue Model)	comprising the company's waste collection ,transfer,recycling and resource recovery and disposal services ,which are operated and managed locally by the company's various subsidiaries ,which focus on distinct geographic areas and corporate and

		other activities, including its development and moperation of landfill gas to energy facilities in the INDIA ,and its recycling brokerage services,as well as various corporate functions .
6	Scalability of the Solution	In this regard ,smart city design has been increasingly studied and discussed around the world to solve this problem. Following this approach ,this paper presented an efficient IoT based and real time waste management model for improving the living environment in cities focused on a citizen perspective. The proposed system uses sensor and communication technologies where waste data is collected from the smart bin in real time and then transmitted to an online platform where citizens can access check the availability the compartments scattered around a city.

PROBLEM SOLUTION FIT PHASE



SOLUTION REQUIREMENTS

Solution Requirements (Functional & Non-functional)

Functional Requirements:

Following are the functional requirements of the proposed solution.

Fr no	Functional Requirement	Sub Requirement
FR-1	Detailed bin inventory.	All monitored bins and stands can be seen on the map, and you can visit them at any time via the Street View feature from Google. Bins or stands are visible on the map as green, orange or red circles. You can see bin details in the

		Dashboard – capacity, waste type, last measurement, GPS location and collection schedule or pick recognition.
FR-2	Real time bin monitoring.	The Dashboard displays real-time data on fill-levels of bins monitored by smart sensors. In addition to the % of fill-level, based on the historical data, the tool predicts when the bin will become full, one of the functionalities that are not included even in the best waste management software.. Sensors recognize picks as well; so you can check when the bin was last collected. With real-time data and predictions, you can eliminate the overflowing bins and stop collecting half-empty once
FR-3	Expensive bins	We help you identify bins that drive up your collection costs. The tool calculates a rating for each bin in terms of collection costs. The tool considers the average distance depo-bindischarge in the area. The tool assigns bin a rating (1-10) and calculates distance from depo-bin discharge.

FR-4	Adjust bin distribution.	Ensure the most optimal distribution of bins. Identify areas with either dense or sparse bin distribution. Make sure all trash types are represented within a stand. Based on the historical data, you can adjust bin capacity or location where necessary.
FR-5	Eliminate unefficient picks.	Eliminate the collection of half-empty bins. The sensors recognize picks. By using real-time data on fill-levels and pick recognition, we can show you how full the bins you collect are. The report shows how full the bin was when picked. You immediately see any inefficient picks below 80% full.

Non-Functional Requirement

Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

NFR NO	Non-Functional Requirement	Description
1	Usability	IoT device verifies that

		usability is a special and importantperspective to analyze user requirements, which can further improve the design quality. In the design process with user experience as the core, the analysis of users' product usability can indeed help designers better understand users' potential needs in waste management, behavior and experience.
2	Security	Use a reusable bottles Use reusable grocery bags Purchase wisely and recycle Avoid single use food and drink containers.
3	Reliability	Smart waste management is also about creating better working conditions for waste collectors and drivers. Instead of driving the same collection routes and servicing empty bins, waste collectors will spend their time more efficiently, taking care of bins that need servicing.
4	Performance	The Smart Sensors use ultrasound technology to measure the fill levels (along with other data) in bins several times a day. Using a variety of IoT networks (

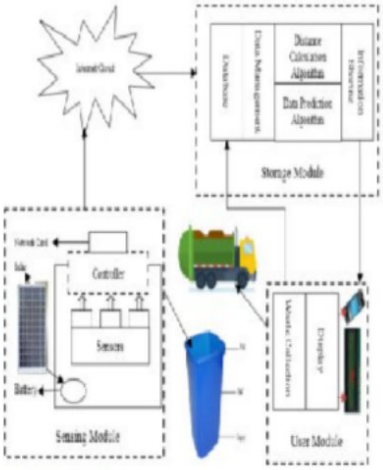
		(NB-IoT,GPRS), the sensors send the data to Sensoneo’s Smart Waste Management Software System, a powerful cloud-based platform, for datadriven daily operations, available also as a waste management app. Customers are hence provided data-driven decision making, and optimization of waste collection routes, frequencies, and vehicle loads resulting in route reduction by at least 30%.
5	Availability	By developing & deploying resilient hardware and beautiful software we empower cities, businesses, and countries to manage waste smarter.

DATA FLOW DIAGRAM

DATA FLOW DIAGRAM

SMART WASTE MANAGEMENT SYSTEM FOR METROPOLITAN CITIES

TEAM ID: PNT2022TMID07334

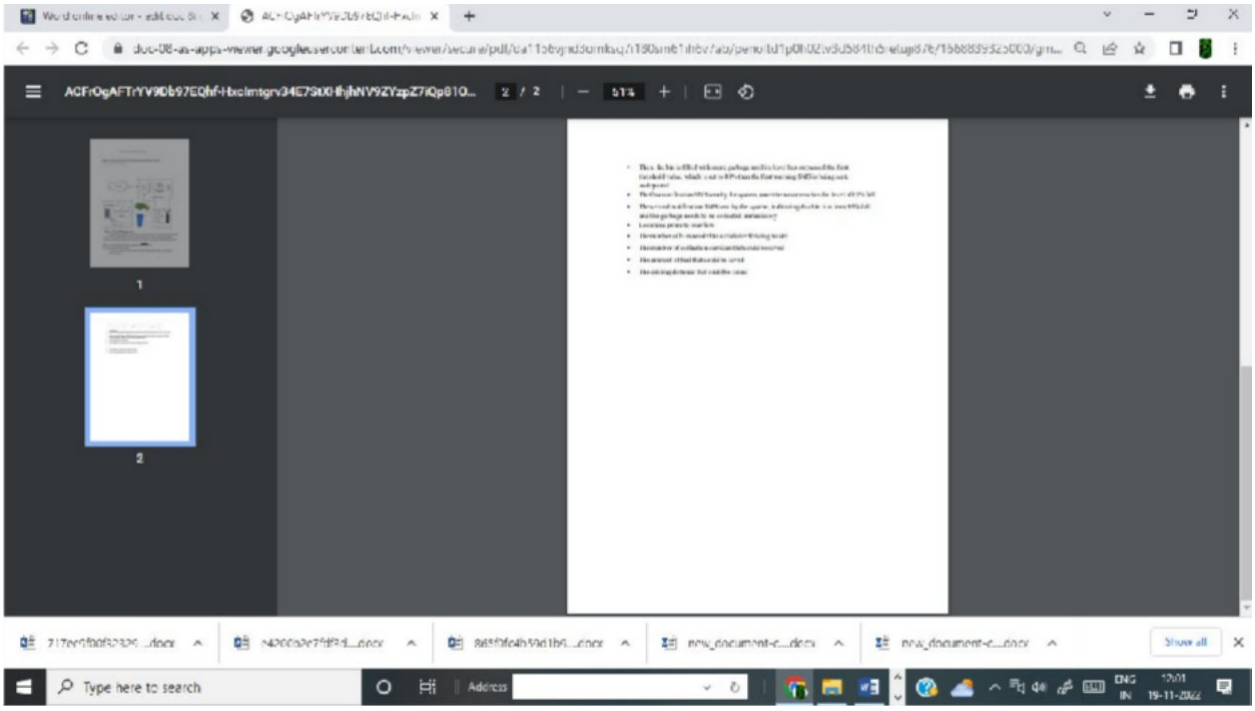


Data Flow Diagrams:

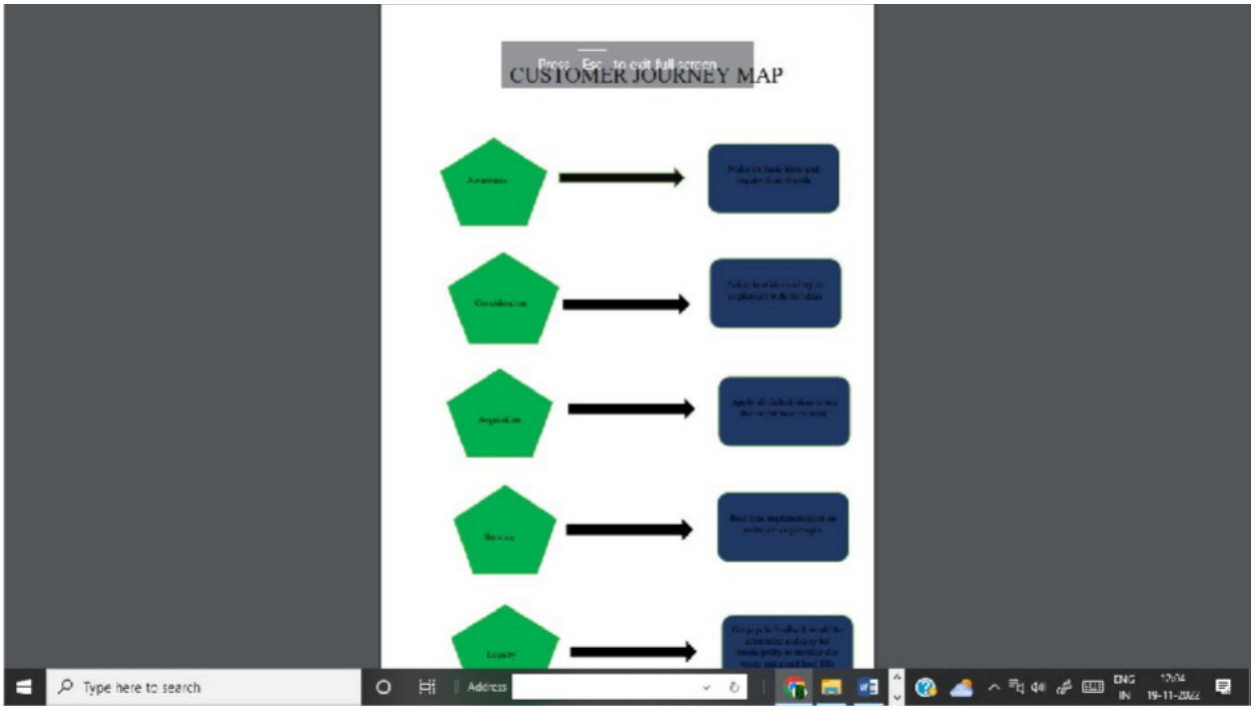
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

A smart waste management platform uses analytics to turn the data gather in your bins into actionable insights to help you improve your waste services.
You can receive data or metric such as:

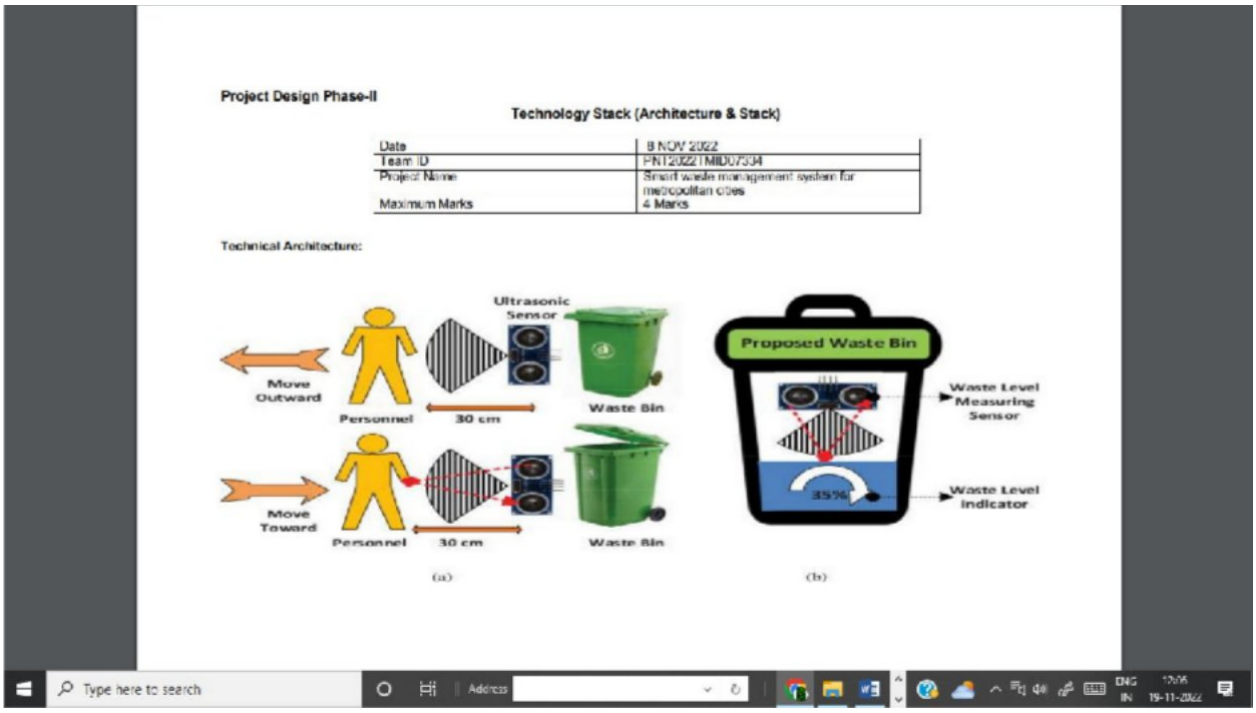
- The first test conducted is the situation where the garbage bin is empty or its garbage level is very low.



CUSTOMER JOURNEY MAP



TECHNOLOGY ARCHITECTURE



CODING

```
from __future__import division
from __future__import print_function
from django.shortcuts import render
from django.contrib import auth
import requests
import json
import urllib.request
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
from datetime import date
from datetime import datetime
import pyrebase
import json


config = {
  'apiKey': "AIzaSyB6s7DSe9M6MZk7g77cMTuoqIO6d-ebKwI",
  'authDomain' : "garbage-monitoring.firebaseapp.com",
  'databaseURL' : "https://garbage-monitoring.firebaseio.com",
  'projectId' : "garbage-truck-monitoring",
  'storageBucket' : "garbage-monitoring.appspot.com",
  'messagingSenderId' : "549306067582",
```

```
  'appId' : "1:549306067582:web:bbaeac9ec829045099c62f",
  'measurementId' : "G-X9JCRW3TR0"
}
firebase = pyrebase.initialize_app(config)
authe = firebase.auth()
database=firebase.database()

def signIn(request):
  return render(request, "sign.html")

def postsign(request):
  email=request.POST.get('email')
  passw = request.POST.get("pass")
  try:
    user = authe.sign_in_with_email_and_password(email,passw)
  except:
    message="invalid credentials"
    return render(request,"sign.html",{"messg":message})
  print(user['idToken'])
  session_id=user['idToken']
  request.session['uid']=str(session_id)
  return render(request, "welcome.html",{"e":email})

def logout(request):
  auth.logout(request)
  return render(request,'sign.html')
from django.shortcuts import render
from django.contrib import auth
import json

import pyrebase

from datetime import date

def get_latlong(request) :
  from django.shortcuts import render
  from django.contrib import auth


  import json

  import pyrebase

  config = {
    'apiKey': "AIzaSyA3W-x4zqHwfCJ2xgzLvuO1MVPlWwp_XJI",
    'authDomain': "garbage-monitoring.firebaseapp.com",
    'databaseURL': "https://garbage-monitoring.firebaseio.com",
    'projectId': "garbage-truck-monitoring",
    'storageBucket': "garbage-monitoring.appspot.com",
    'messagingSenderId': "549306067582",
```

```
    'appId': "1:549306067582:web:bbaeac9ec829045099c62f",
    'measurementId': "G-X9JCRW3TR0"
  }
  firebase = pyrebase.initialize_app(config)

  db = firebase.database()
  bin = db.child("Bin").get().val()
  bin2 = db.child("BinPerLevel").get().val()
  lat, lon, cap = [], [], []
  cap_70, cap_20, cap_20_70 = [], [], []
  print(bin)
  for i in bin:
    height = (int(db.child("Bin").child(i).child("height").get().val()))
    lati = (float(db.child("Bin").child(i).child("latitude").get().val()))
    long = (float(db.child("Bin").child(i).child("longitude").get().val()))
    print(i)
    try :
      data = db.child("BinPerLevel").child(i).child("2020-01-21").get().val()
      last = next(reversed(data))
      height2 = db.child("BinPerLevel").child(i).child("2020-01-21").child(last).child("height").get().val()
      perc = (int(height2) / int(height)) * 100
      if perc >= 70:
        cap_70.append([lati, long])
      elif perc <= 20:
        cap_20.append([lati, long])
      else:
        cap_20_70.append([lati, long])
    except:
      pass
  cap_20 = json.dumps(cap_20)
  cap_20_70 = json.dumps(cap_20_70)
  cap_70 = json.dumps(cap_70)
  print(cap_70)
  print(cap_20_70)
  print(cap_20)

  # return render(request, 'neww.html',{"cap_70":cap_70,"cap_20_70":cap_20_70,"cap_20":cap_20})
  return render(request, 'latlong.html', {"cap_70": cap_70, "cap_20_70": cap_20_70, "cap_20": cap_20})


def get_latlong2(request) :
  """
  from django.shortcuts import render
  from django.contrib import auth

  import json

  import pyrebase
```

```
    config = {
        'apiKey': "AIzaSyB6s7DSe9M6MZk7g77cMTuoqIO6d-ebKwI",
        'authDomain': "garbage--monitoring.firebaseapp.com",
        'databaseURL': "https://garbage--monitoring.firebaseio.com",
        'projectId': "garbage-truck-monitoring",
        'storageBucket': "garbage-monitoring.appspot.com",
        'messagingSenderId': "549306067582",
        'appId': "1:549306067582:web:bbaeac9ec829045099c62f",
        'measurementId': "G-X9JCRW3TR0"
    }
    firebase = pyrebase.initialize_app(config)

    db = firebase.database()
    bin = db.child("Bin").get().val()
    lat, lon, cap = [], [], []
    cap = []
    print(bin)
    for i in bin:
        lati = (float(db.child("Bin").child(i).child("latitude").get().val()))
        long = (float(db.child("Bin").child(i).child("longitude").get().val()))
        cap.append([lati, long])

    cap = json.dumps(cap)
    print(cap)
    return render(request,'new2.html',{"cap":cap})
"""

    from django.shortcuts import render
    from django.contrib import auth

    import json

    import pyrebase

    config = {
        'apiKey': "AIzaSyA3W-x4zqHwfCJ2xgzLvuO1MVPlWwp_XJI",
        'authDomain': "garbage-monitoring.firebaseapp.com",
        'databaseURL': "https://garbage--monitoring.firebaseio.com",
        'projectId': "garbage-truck-monitoring",
        'storageBucket': "garbage-monitoring.appspot.com",
        'messagingSenderId': "549306067582",
        'appId': "1:549306067582:web:bbaeac9ec829045099c62f",
        'measurementId': "G-X9JCRW3TR0"
    }
    firebase = pyrebase.initialize_app(config)

    db = firebase.database()
    bin = db.child("Bin").get().val()
    bin2 = db.child("BinPerLevel").get().val()
```

```
  lat, lon, cap = [], [], []
  cap_70, cap_20, cap_20_70 = [], [], []
  for i in bin:
     height = (int(db.child("Bin").child(i).child("height").get().val()))
     lati = (float(db.child("Bin").child(i).child("latitude").get().val()))
     long = (float(db.child("Bin").child(i).child("longitude").get().val()))
     print(i,height)
     try:
        data = db.child("BinPerLevel").child(i).child(str(date.today())).get().val()
        print(data)
        last = next(reversed(data))
        height2 = db.child("BinPerLevel").child(i).child(str(date.today())).child(last).child("height").get().val()
        print("here",height2)
        perc = (float(height2) / float(height)) * 100
        print(perc)
        if perc >= 70.0 :
           cap_70.append([lati, long])
        elif perc <= 20.0 :
           cap_20.append([lati, long])
        else:
           cap_20_70.append([lati, long])
     except:
        pass
  cap_20 = json.dumps(cap_20)
  cap_20_70 = json.dumps(cap_20_70)
  cap_70 = json.dumps(cap_70)
  print(cap_70)
  print(cap_20_70)
  print(cap_20)

  return render(request, 'marker.html', {"cap_70": cap_70, "cap_20_70": cap_20_70, "cap_20": cap_20})


def create_bin(request):

   return render(request,'CreateBin.html')

def post_create_bin(request):
   lat = str(request.POST.get('lat'))
   lon = str(request.POST.get('lon'))
   lat = lat.replace(".","-")
   lon = lon.replace(".", "-")
   id = lat + "|" + lon
   lat = lat.replace("-", ".")
   lon = lon.replace("-", ".")
   print(id)
   capacity = request.POST.get('capacity')
   height = request.POST.get("height")
   # idtoken= request.session['uid']
```

```
    # a = authe.get_account_info(idtoken)
    # a = a['users']
    # a = a[0]
    # a = a['localId']
    #

    data = {
        "latitude":lat,
        'longitude':lon,
        'capcity':capacity,
        'height' :height
    }
    database.child('Bin').child(id).set(data)
    # name = database.child('users').child(id).child('details').child('name').get().val()
    bins = database.child('Bin').get().val()
    print(bins)
    name = "vinal"
    return render(request,'welcome.html', {'e':name})

def create_depot(request):
    return render(request,'CreateDepot.html')

def post_create_depot(request):
    lat = str(request.POST.get('lat'))
    lon = str(request.POST.get('lon'))
    lat = lat.replace(".","-")
    lon = lon.replace(".", "-")
    id = lat + "|" + lon
    lat = lat.replace("-", ".")
    lon = lon.replace("-", ".")
    print(id)
    # idtoken= request.session['uid']
    # a = authe.get_account_info(idtoken)
    # a = a['users']
    # a = a[0]
    # a = a['localId']

    data = {
        "latitude":lat,
        'longitude':lon,
    }
    database.child('Depot').child(id).set(data)
    # name = database.child('users').child(id).child('details').child('name').get().val()
    name = "vinal"
    return render(request,'welcome.html', {'e':name})


def create_vehicle(request):
    return render(request, 'CreateVehicle.html')
```

```
def post_create_vehicle(request):
    if request.method == 'POST':
        vehicle_no = str(request.POST.get('vehicleNo'))
        capacity = str(request.POST.get('capacity'))
        idtoken = request.session['uid']
        print(vehicle_no)
        # a = authe.get_account_info(idtoken)
        # a = a['users']
        # a = a[0]
        # a = a['localId']

        data = {
            'capacity': capacity
        }
        print(data)
        database.child('Vehicle').child(vehicle_no).set(data)
        print(database.child('Vehicle').get().val())
        name = "vinal"
        #return render(request, 'welcome.html', {'e': name})
    return render(request, 'welcome.html')

def create_driver(request):
    return render(request, 'CreateDriver.html')

def post_create_driver(request):
    mobileNo = request.POST.get('mobile')
    name = request.POST.get('name')
    age  =request.POST.get('age')
    address =request.POST.get('address')
    gender =request.POST.get('gender')
    password = request.POST.get('password')
    date =request.POST.get('joiningdate')
    #idtoken= request.session['uid']
    # a = authe.get_account_info(idtoken)
    # a = a['users']
    # a = a[0]
    # a = a['localId']


    data = {
        "name":name,
        'age':age,
        'gender': gender,
        'address':address,
        'joining date': date,
        'password': password
    }
```

```
    database.child('Driver').child(mobileNo).set(data)
    name = database.child('users').child(id).child('details').child('name').get().val()
    return render(request,'welcome.html', {'e':name})


def check(request):

#- - - - - - - - - - - - - - - - - Driver
    timestamps = database.child('Driver').get().val()
    lis_time=[]
    for i in timestamps:

        lis_time.append(i)

    lis_time.sort(reverse=True)
    print("hello")
    print(lis_time)
    address = []
    age = []
    gender = []
    date = []
    name = []
    for i in lis_time:

        n=database.child('Driver').child(i).child('name').get().val()
        name.append(n)
        ag=database.child('Driver').child(i).child('age').get().val()
        age.append(ag)
        addr=database.child('Driver').child(i).child('address').get().val()
        address.append(addr)
        gen=database.child('Driver').child(i).child('gender').get().val()
        gender.append(gen)
        da=database.child('Driver').child(i).child('joining date').get().val()
        date.append(da)

    print(name)
    print(address)
    print(age)
    print(gender)
    print(date)


    comb_lis = zip(name,address,age,gender,date)

    #- - - - - - - - - - - - - - - - - Bin
    bindetails = database.child('Bin').get().val()
    bin_details=[]
    for i in bindetails:
```

```
        bin_details.append(i)

    bin_details.sort(reverse=True)

    latitude = []
    longitude = []
    capacity = []

    for i in bin_details:

        lat=database.child('Bin').child(i).child('latitude').get().val()
        latitude.append(lat)
        lon=database.child('Bin').child(i).child('longitude').get().val()
        longitude.append(lon)

        cap=database.child('Bin').child(i).child('capacity').get().val()
        capacity.append(cap)


    comb_lis_bin = zip(latitude,longitude,capacity)


    return render(request,'check.html',{'comb_lis':comb_lis,'comb_lis_bin':comb_lis_bin,'e':"Palak"})

def check_queries(request):
    citizendetails = database.child('Citizen').get().val()
    print(citizendetails)
    citizen_details=[]

    for i in citizendetails:
        citizen_details.append(i)

    date_wise = []
    for i in citizen_details:
        date_wise.append([])
        data = database.child('Citizen').child(i).get().val()
        count = 0
        for j in data:
            date_wise[0].append(data)
            count+=1

    citizen_details.sort(reverse=True)
    print(date_wise)

    citizen_id = []
    address = []
    image = []
    name = []
    query = []
```

```
    date = []
    for key,value in citizendetails.items():


        for keysecond,valuesecond in value.items():
            print(keysecond)
            citizen_id.append(key)
            date.append(keysecond)
            print(date)
            address.append(valuesecond['address'])
            name.append(valuesecond['name'])
            query.append(valuesecond['query'])



    comb_lis_query = list(zip(name,address,query,date,citizen_id))

    return render(request,'checkQueries.html',{'comb_lis_query':comb_lis_query})

def create_distance_matrix(data):
    addresses = data["addresses"]
    API_key = data["API_key"]
    # Distance Matrix API only accepts 100 elements per request, so get rows in multiple requests.
    max_elements = 100
    num_addresses = len(addresses) # 16 in this example.
    # Maximum number of rows that can be computed per request (6 in this example).
    max_rows = max_elements // num_addresses
    # num_addresses = q * max_rows + r (q = 2 and r = 4 in this example).
    q, r = divmod(num_addresses, max_rows)
    dest_addresses = addresses
    distance_matrix = []
    # Send q requests, returning max_rows rows per request.
    for i in range(q):
        origin_addresses = addresses[i * max_rows: (i + 1) * max_rows]
        response = send_request(origin_addresses, dest_addresses, API_key)
        distance_matrix += build_distance_matrix(response)
    # Get the remaining remaining r rows, if necessary.
    if r > 0:
        origin_addresses = addresses[q * max_rows: q * max_rows + r]
        response = send_request(origin_addresses, dest_addresses, API_key)
        distance_matrix += build_distance_matrix(response)
    return distance_matrix

def send_request(origin_addresses, dest_addresses, API_key):
    def build_address_str(addresses):
    # Build a pipe-separated string of addresses
        address_str = ''
        for i in range(len(addresses) - 1):
            address_str += addresses[i] + '|'
```

```
        address_str += addresses[-1]
        return address_str

    request = 'https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial'
    origin_address_str = build_address_str(origin_addresses)
    dest_address_str = build_address_str(dest_addresses)
    request = request + '&origins=' + origin_address_str + '&destinations=' + \
                        dest_address_str + '&key=' + API_key
    jsonResult = urllib.request.urlopen(request).read()
    response = json.loads(jsonResult)
    return response

def build_distance_matrix(response):
    distance_matrix = []
    for row in response['rows']:
        row_list = [row['elements'][j]['distance']['value'] for j in range(len(row['elements']))]
        distance_matrix.append(row_list)
    return distance_matrix

def get_routes(manager, routing, solution, num_routes):
    """Get vehicle routes from a solution and store them in an array."""
    # Get vehicle routes and store them in a two dimensional array whose
    # i,j entry is the jth location visited by vehicle i along its route.
    routes = []
    for route_nbr in range(num_routes):
        index = routing.Start(route_nbr)
        route = [manager.IndexToNode(index)]
        while not routing.IsEnd(index):
            index = solution.Value(routing.NextVar(index))
            route.append(manager.IndexToNode(index))
        routes.append(route)
#     print(routes)
    return routes

def get_vehicle_capacities():
    vehicles = database.child('Vehicle').get().val()
    print('Vehicles',vehicles)
    vehicle_cap = []
    vehicle_key = []
    for i in vehicles:
        vehicle_cap.append(int(vehicles[i]['capacity']))
        vehicle_key.append(i)
    print("Vehicle Cap",vehicle_cap)
    return vehicle_cap,vehicle_key

def get_bin_cap():
    bins = database.child('Bin').get().val()
    print('Bins',bins)
    bin_cap = []
```

```
  for i in bins:
    bin_cap.append(int(bins[i]['capcity']))
  print("Bin Cap",bin_cap)
  return bin_cap
  #a = [0, 1, 1, 4, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
 # return bin_cap

def get_bin_address():
  bin_addr = []
  bins = database.child('Bin').get().val()
  binVal = database.child('Bin').get().key()
  print(bins)
  #bins = bins.remove(0)
  for i in bins:
    s = ""
    s = str(bins[i]['latitude'])+","+str(bins[i]['longitude'])
    bin_addr.append(s)
  print(bin_addr)
  return bin_addr

def get_depot_location():
  depot = database.child('Depot').get().val()
  print("DepotNew: ",depot)
  depotarr = []
  s = ""
  for i in depot:
    s = str(depot[i]['latitude']) + "," + str(depot[i]['longitude'])
  depotarr.append(s)
  return depotarr

# def get_dumping_location():
#     dumping = database.child('DumpG').get().val()
#     print("Dumping: ",dumping)
#     dumparr = []
#     s = ""
#     for i in dumping:
#         s = str(dumping[i]['latitude']) + "," + str(dumping[i]['longitude'])
#     dumparr.append(s)
#     return dumparr

def generate_routes(request):
  data = {}
  data['API_key'] = 'YOUR_KEY'
  # data['addresses'] = ['19.312251,72.8513579', # depot
  #                  '19.3844215,72.8221597',
  #                  '19.3084312,72.8489729',
  #                  '19.3834291,72.8280696',
  #                  '19.3834291,72.8280696',
  #                  '19.2813741,72.8559049',
```

```
#                 '19.2527913,72.8506576',
#                 '19.2813741,72.8559049',
#                 '19.2864772,72.8486726',
#                 '19.2794676,72.8775643',
#                 '19.3726195,72.8255362',
#                 '19.3726195,72.8255362',
#                 '19.3726195,72.8255362',
#                 '19.3720507,72.8268628',
#                 '19.3720507,72.8268628',
#                 '19.3720419,72.8268988',
#                 ]
# data['addresses'] = ['19.8597909,75.3091889']
data['addresses'] =  get_depot_location()
data['addresses'] = data['addresses'] + get_bin_address()
print("addr",data['addresses'])
"""Solve the CVRP problem."""

"""Stores the data for the problem."""
datap = {}
datap['distance_matrix'] = create_distance_matrix(data)
print("dm", datap['distance_matrix'])

# datap['distance_matrix'] = [
#     [0, 31895, 878, 31603, 31603, 5342, 5342, 5342, 3010, 5834, 33590, 33590, 33590, 33821, 33821, 33821],
#     [32453, 0, 32024, 999, 999, 29059, 29059, 29059, 31272, 26481, 2985, 2985, 2985, 3217, 3217, 3217],
#     [878, 32094, 0, 31803, 31803, 4913, 4913, 4913, 2581, 6033, 33789, 33789, 33789, 34020, 34020, 34020],
#     [32453, 1359, 32023, 0, 0, 29058, 29058, 29058, 31271, 26480, 1986, 1986, 1986, 2218, 2218, 2218],
#     [32453, 1359, 32023, 0, 0, 29058, 29058, 29058, 31271, 26480, 1986, 1986, 1986, 2218, 2218, 2218],
#     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516, 31516],
#     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516, 31516],
#     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516, 31516],
#     [3481, 31233, 3052, 30942, 30942, 4052, 4052, 4052, 0, 5172, 32928, 32928, 32928, 33160, 33160, 33160],
#     [5972, 26678, 5543, 26387, 26387, 2577, 2577, 2577, 4791, 0, 28373, 28373, 28373, 28605, 28605, 28605],
#     [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
#     [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
#     [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
#     [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
#     [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
#     [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
# ]
#datap['demands'] = [0, 1, 1, 4, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
datap['demands'] = [0]
datap['demands'] = datap['demands']+ get_bin_cap()
total_bin_cap = sum(datap['demands'])
print("demands", datap['demands'])
#datap['vehicle_capacities'], datap['vehicle_key'] = [15,15,15,15],[1,2,3,4]
datap['vehicle_capacities'],datap['vehicle_key'] = get_vehicle_capacities()
total_veh_cap = sum(datap['vehicle_capacities'])
print("veh_cap", datap['vehicle_capacities'])
```

```
datap['num_vehicles'] = len(datap['vehicle_capacities'])
print("n", datap['num_vehicles'])
datap['depot'] = 0

cap_diff = total_veh_cap-total_bin_cap

# Create the routing index manager.
manager = pywrapcp.RoutingIndexManager(len(datap['distance_matrix']),
                                       datap['num_vehicles'], datap['depot'])



# Create Routing Model.
routing = pywrapcp.RoutingModel(manager)

#     print(routing)

# Create and register a transit callback.
def distance_callback(from_index, to_index):
    """Returns the distance between the two nodes."""
    # Convert from routing variable Index to distance matrix NodeIndex.
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return datap['distance_matrix'][from_node][to_node]

transit_callback_index = routing.RegisterTransitCallback(distance_callback)

# Define cost of each arc.
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

# Add Capacity constraint.
def demand_callback(from_index):
    """Returns the demand of the node."""
    # Convert from routing variable Index to demands NodeIndex.
    from_node = manager.IndexToNode(from_index)
    return datap['demands'][from_node]

demand_callback_index =
    routing.RegisterUnaryTransitCallback(demand_callback)
routing.AddDimensionWithVehicleCapacity(
    demand_callback_index,
    0,  # null capacity slack
    datap['vehicle_capacities'], # vehicle maximum capacities
    True,  # start cumul to zero
    'Capacity')

# Setting first solution heuristic.
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
```

```
# Solve the problem.
print("cap_diff: ",cap_diff)
if (cap_diff>0):
    assignment = routing.SolveWithParameters(search_parameters)
    # solution = routing.SolveWithParameters(search_parameters)
    print('ASSIGNMENT:',assignment)
else:
    print("More trucks are needed of capacity: ",cap_diff)
    return render(request, 'routesError.html',{'capErr':'1','cap_diff':cap_diff})
# Print solution on console.
if assignment:
    # print_solution(data, manager, routing, assignment)
    routes = get_routes(manager, routing, assignment, datap['num_vehicles'])
    # Display the routes.
    Routes = []
    for i, route in enumerate(routes):
        # print('Route', i, route)
        Route = []
        for j in range(len(route)):
            Route.append(data['addresses'][route[j]].split(","))
        Routes.append(Route)
    print(Routes)
else:
    return render(request,'routesError.html',{'capErr':'0'})

vehicle_route = dict()
j = 0
for i in datap['vehicle_key']:
    vehicle_route[i] = Routes[j]
    d =  dict()
    for k in range(len(Routes[j])):
        lat= Routes[j][k][0]
        long = Routes[j][k][1]
        d[k] = {
            "latitude": lat,
            "longitude": long
        }
    print(d)
    sdate = str(date.today())
    st = str(datetime.time(datetime.now()))
    stime = (st[0:5])
    database.child('Route').child(sdate).child(stime).child(i).set(d)
    j = j+1
print(vehicle_route)
truckRoutes = []
#<I love this code>
for key,val in vehicle_route.items():
    print("Key",key)
```

```
        for xy in val:
            x = xy[0]
            y = xy[1]
            print(x,y)
    #</I love this code>
    # test = [ [k,v] for k, v in vehicle_route.items() ]
    # print(test)
    test = json.dumps(vehicle_route)
    vehicles = database.child('Vehicle').get().val()
    print(vehicles)
    vehicleId = []
    for i in vehicles:
        vehicleId.append(int(i))
    return render(request,'generatedRoutes_copy.html',{'route':test,'veh1':
datap['vehicle_key'][0],'vehicleId':vehicleId})




def real_time(request):
    date = datetime.now().strftime("%Y-%m-%d")

    return render(request,'realTime_test.html',{'date':date})


def show_vehicles(request):
    vehicles = database.child('Vehicle').get().val()
    print(vehicles)
    vehicleId = []
    for i in vehicles:
        vehicleId.append(int(i))
    return render(request,'showVehicles.html',{'vehicleId':vehicleId})

def g_routes(request,vId):
    print(vId)
    sdate = str(date.today())
    p = database.child('Route').child(sdate).get().val()
    print("P",p)
    lastIndex = ""
    for i in p:
        lastIndex = i
        print(i)
    j = p[lastIndex][str(vId)]
    routes = json.dumps(j)
    print(routes)
    return render(request,'showRoutes.html',{'route' :routes,'vId':vId})
```

```
# import datetime

def updateFeedback(request):

    comment = request.POST.get('feedback')
    id =request.POST.get('id')
    date = datetime.now().strftime("%Y:%m:%d:%H:%M:%S")
    # console.log()
    print(comment, "------", id ,"- - - - ", date)
    data = {
        "feedback":comment,

    }
    # current = date.year-date.month-date.day,"-",date.hour,"-",date.minute,"-",date.second
    print(date)
    database.child('Feedback').child(id).child(date).set(data)
    return render(request,'welcome.html', {'e':"palak  "})




def create_dump(request):
    return render(request,'CreateDump.html')

def  post_create_dump(request):
    lat = str(request.POST.get('lat'))
    lon = str(request.POST.get('lon'))
    lat = lat.replace(".","-")
    lon = lon.replace(".", "-")
    id = lat + "|" + lon
    lat = lat.replace("-", ".")
    lon = lon.replace("-", ".")
    print(id)
    # idtoken= request.session['uid']
    # a = authe.get_account_info(idtoken)
    # a = a['users']
    # a = a[0]
    # a = a['localId']

    data = {
        "latitude":lat,
        'longitude':lon,
    }
    database.child('DumpG').child(id).set(data)
    # name = database.child('users').child(id).child('details').child('name').get().val()
    name = "vinal"
```

```
    return render(request,'welcome.html', {'e':name})



########test######
def get_vehicle_capacities_test():
    vehicles = database.child('Vehicle').get().val()
    print('Vehicles',vehicles)
    vehicle_cap = []
    vehicle_key = []
    for i in vehicles:
        vehicle_cap.append(int(vehicles[i]['capacity']))
        vehicle_key.append(i)
    print("Vehicle Cap",vehicle_cap)
    return vehicle_cap,vehicle_key

def get_bin_cap_test(bin_addr):
    bins = database.child('Bin').get().val()
    print('Bins',bins)
    bin_cap = []
    for i in bin_addr:
        bin_cap.append(int(bins[i]['capcity']))
    print("Bin Cap",bin_cap)
    return bin_cap
    #a = [0, 1, 1, 4, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
   # return bin_cap

def get_bin_address_test():
    config = {
        'apiKey': "AIzaSyA3W-x4zqHwfCJ2xgzLvuO1MVPlWwp_XJI",
        'authDomain': "garbage-truck-monitoring.firebaseapp.com",
        'databaseURL': "https://garbage-truck-monitoring.firebaseio.com",
        'projectId': "garbage-truck-monitoring",
        'storageBucket': "garbage-truck-monitoring.appspot.com",
        'messagingSenderId': "549306067582",
        'appId': "1:549306067582:web:bbaeac9ec829045099c62f",
        'measurementId': "G-X9JCRW3TR0"
    }
    firebase = pyrebase.initialize_app(config)

    db = firebase.database()
    bin = db.child("Bin").get().val()
    bin2 = db.child("BinPerLevel").get().val()
    lat, lon, cap = [], [], []
    cap_70, cap_20, cap_20_70 = [], [], []
    for i in bin:
        height = (int(db.child("Bin").child(i).child("height").get().val()))
        lati = (float(db.child("Bin").child(i).child("latitude").get().val()))
        long = (float(db.child("Bin").child(i).child("longitude").get().val()))
```

```
        print(i, height)
        try:
            data = db.child("BinPerLevel").child(i).child(str(date.today())).get().val()
            print(data)
            last = next(reversed(data))
            height2 = db.child("BinPerLevel").child(i).child(str(date.today())).child(last).child("height").get().val()
            print("here", height2)
            perc = (float(height2) / float(height)) * 100
            print(perc)
            if perc >= 70.0:
                cap_70.append(str(lati) + ',' + str(long))
            elif perc <= 20.0:
                cap_20.append(str(lati) + ',' + str(long))
            else:
                cap_20_70.append(str(lati) + ',' + str(long))
        except:
            pass
    bin_addr = cap_20_70 + cap_70
    print("OG length:",len(bin))
    print("New bins: length: ",bin_addr,len(bin_addr))
    return bin_addr



def generate_routes_test(request):
    data = {}
    data['API_key'] = 'YOUR_KEY'
    # data['addresses'] = ['19.312251,72.8513579',  # depot
    #                      '19.3844215,72.8221597',
    #                      '19.3084312,72.8489729',
    #                      '19.3834291,72.8280696',
    #                      '19.3834291,72.8280696',
    #                      '19.2813741,72.8559049',
    #                      '19.2527913,72.8506576',
    #                      '19.2813741,72.8559049',
    #                      '19.2864772,72.8486726',
    #                      '19.2794676,72.8775643',
    #                      '19.3726195,72.8255362',
    #                      '19.3726195,72.8255362',
    #                      '19.3726195,72.8255362',
    #                      '19.3720507,72.8268628',
    #                      '19.3720507,72.8268628',
    #                      '19.3720419,72.8268988',
    #                      ]
    # data['addresses'] = ['19.8597909,75.3091889']
    data['addresses'] = get_depot_location()
    bin_addr = get_bin_address_test()
    data['addresses'] = data['addresses'] + bin_addr
    print("addr",data['addresses'])
```

```
  """Solve the CVRP problem."""

  """Stores the data for the problem."""
  datap = {}
  datap['distance_matrix'] = create_distance_matrix(data)
  print("dm", datap['distance_matrix'])

  # datap['distance_matrix'] = [
  #     [0, 31895, 878, 31603, 31603, 5342, 5342, 5342, 3010, 5834, 33590, 33590, 33590, 33821, 33821,
33821],
  #     [32453, 0, 32024, 999, 999, 29059, 29059, 29059, 31272, 26481, 2985, 2985, 2985, 3217, 3217,
3217],
  #     [878, 32094, 0, 31803, 31803, 4913, 4913, 4913, 2581, 6033, 33789, 33789, 33789, 34020, 34020,
34020],
  #     [32453, 1359, 32023, 0, 0, 29058, 29058, 29058, 31271, 26480, 1986, 1986, 1986, 2218, 2218,
2218],
  #     [32453, 1359, 32023, 0, 0, 29058, 29058, 29058, 31271, 26480, 1986, 1986, 1986, 2218, 2218,
2218],
  #     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516,
31516],
  #     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516,
31516],
  #     [5258, 29590, 4828, 29298, 29298, 0, 0, 0, 4076, 3026, 31285, 31285, 31285, 31516, 31516,
31516],
  #     [3481, 31233, 3052, 30942, 30942, 4052, 4052, 4052, 0, 5172, 32928, 32928, 32928, 33160,
33160, 33160],
  #     [5972, 26678, 5543, 26387, 26387, 2577, 2577, 2577, 4791, 0, 28373, 28373, 28373, 28605,
28605, 28605],
  #      [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
  #      [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
  #      [34313, 2611, 33883, 2294, 2294, 30918, 30918, 30918, 33131, 28340, 0, 0, 0, 511, 511, 511],
  #      [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
  #      [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
  #      [34219, 2517, 33789, 2200, 2200, 30824, 30824, 30824, 33037, 28247, 511, 511, 511, 0, 0, 0],
  # ]
  #datap['demands'] = [0, 1, 1, 4, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
  datap['demands'] = [0]
  for l in range(len(bin_addr)):
      bin_addr[l] = bin_addr[l].replace(".",'-')
      bin_addr[l] = bin_addr[l].replace(",",'|')

  datap['demands'] = datap['demands']+ get_bin_cap_test(bin_addr)
  total_bin_cap = sum(datap['demands'])
  print("demands", datap['demands'])
  #datap['vehicle_capacities'], datap['vehicle_key'] = [15,15,15,15],[1,2,3,4]
  datap['vehicle_capacities'],datap['vehicle_key'] = get_vehicle_capacities_test()
  total_veh_cap = sum(datap['vehicle_capacities'])
  print("veh_cap", datap['vehicle_capacities'])
  datap['num_vehicles'] = len(datap['vehicle_capacities'])
```

```
print("n", datap['num_vehicles'])
datap['depot'] = 0


cap_diff = total_veh_cap-total_bin_cap

# Create the routing index manager.
manager = pywrapcp.RoutingIndexManager(len(datap['distance_matrix']),
                                       datap['num_vehicles'], datap['depot'])



# Create Routing Model.
routing = pywrapcp.RoutingModel(manager)

#     print(routing)

# Create and register a transit callback.
def distance_callback(from_index, to_index):
    """Returns the distance between the two nodes."""
    # Convert from routing variable Index to distance matrix NodeIndex.
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return datap['distance_matrix'][from_node][to_node]

transit_callback_index = routing.RegisterTransitCallback(distance_callback)

# Define cost of each arc.
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

# Add Capacity constraint.
def demand_callback(from_index):
    """Returns the demand of the node."""
    # Convert from routing variable Index to demands NodeIndex.
    from_node = manager.IndexToNode(from_index)
    return datap['demands'][from_node]

demand_callback_index =
    routing.RegisterUnaryTransitCallback(demand_callback)
routing.AddDimensionWithVehicleCapacity(
    demand_callback_index,
    0,  # null capacity slack
    datap['vehicle_capacities'], # vehicle maximum capacities
    True,  # start cumul to zero
    'Capacity')

# Setting first solution heuristic.
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
```

```
# Solve the problem.
print("cap_diff: ",cap_diff)
if (cap_diff>0):
    assignment = routing.SolveWithParameters(search_parameters)
    # solution = routing.SolveWithParameters(search_parameters)
    print('ASSIGNMENT:',assignment)
else:
    print("More trucks are needed of capacity: ",cap_diff)
    return render(request, 'routesError.html',{'capErr':'1','cap_diff':cap_diff})
# Print solution on console.
if assignment:
    # print_solution(data, manager, routing, assignment)
    routes = get_routes(manager, routing, assignment, datap['num_vehicles'])
    # Display the routes.
    Routes = []
    for i, route in enumerate(routes):
        # print('Route', i, route)
        Route = []
        for j in range(len(route)):
            Route.append(data['addresses'][route[j]].split(","))
        Routes.append(Route)
    print(Routes)
else:
    return render(request,'routesError.html',{'capErr':'0'})

vehicle_route = dict()
j = 0
for i in datap['vehicle_key']:
    vehicle_route[i] = Routes[j]
    d =  dict()
    for k in range(len(Routes[j])):
        lat= Routes[j][k][0]
        long = Routes[j][k][1]
        d[k] = {
            "latitude": lat,
            "longitude": long
        }
    print(d)
    sdate = str(date.today())
    st = str(datetime.time(datetime.now()))
    stime = (st[0:5])
    database.child('Route').child(sdate).child(stime).child(i).set(d)
    j = j+1
print(vehicle_route)
truckRoutes = []
#<I love this code>
for key,val in vehicle_route.items():
    print("Key",key)
    for xy in val:
```

```
        x = xy[0]
        y = xy[1]
        print(x,y)
    #</I love this code>
    # test = [ [k,v] for k, v in vehicle_route.items() ]
    # print(test)
```

DEMO VIDEO LINK

https://drive.google.com/file/d/18_3Our1nST_ToAx3mzJm8UlTZBYbPYas/view?usp=drivesdk

ADVANTAGES

It saves time and money by using smart waste collection bins and systems equipped with fill level sensors. As smart transport vehicles go only to the filled containers or bins. It reduces infrastructure, operating and maintenance costs by upto 30%.

➡It decreases traffic flow and consecutively noise due to less air pollution as result ofless waste collection vehicles on the roads. This has become possible due to two way communication between smart dustbins and service operators.

➡It keeps our surroundings clean and green and free from bad odour of wastes, emphasizes on healthy environment and keep cities more beautiful.

➡ It further reduces manpower requirements to handle the garbage collection process.

➡ Applying smart waste management process to the city optimizes management,resources and costs which makes it a "smart city".

➡ It helps administration to generate extra revenue by advertisements on smart devices

DISADVANTAGES:

System requires more number of waste bins for separate waste collection as per population in the city. This results into high initial cost due to expensive smart dustbins compare to other methods.

➡Sensor nodes used in the dustbins have limited memory size.

➡Wireless technologies used in the system such as zigbee and wifi have shorter rangeand lower data speed. In RFID based systems, RFID tags are affected by surrounding metal objects (if any).

➡It reduces man power requirements which results into increase in unemploymentsfor unskilled people.

➡The trainining has to be provided to the people involved in the smart waste management system

CONCLUSION:

This work an IOT enabled Smart Waste Bin with real time monitoring is designed and presented. In addition to the waste level measurement by using ultrasonic sensors, a sensing mechanism based on simple parallel plate capacitance is also developed and presented. Experimental investigations are carried out where the waste level of the smart bins is measured using the parallel plate capacitance and ultrasonic sensors and the statuses of the bins are communicated to the cloud effectively. The results prove the efficiency of the designed smart bins qualitatively. A smart waste management system incorporating robotic smart bins, where the smart bin has the mobility to move to the waste dockyard by localizing itself in the environment, is also proposed in this work. This system could find an application insmart buildings where the waste management could be practiced autonomously in a smarter way. Our future work is to investigate the performance of the proposed traditional and robotic waste management system in outdoor and indoor environment respectively in our institutional campus.