CODING FOR MONITORING

TEAM ID : PNT2022TMID07334

PROJECT NAME: SMART WASTE MANAGENMENT SYSTEM FOR MERTOPOLITAN CITIES

```
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-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)
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-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 = [], [], []
    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-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()
    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-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([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)
    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})
```