```python
# Authored By: Jayant Sogikar
import ast
from nis import cat
import geocoder
from datetime import datetime
import json
from math import asin, cos, pi, sqrt
import os
from flask import Flask, jsonify, url_for, redirect, request, Blueprint
import pymongo
from bson.json_util import dumps
from bson import ObjectId
from flask_caching import Cache
from dataGenerator.barcodelookup import getBarcodeInfo
from dataGenerator.closestSupplierInfo import getSupplierInfo
from dataGenerator.distanceEmission import Emission
from dataGenerator.geoRun import calcTotalDistance
from predictor.predict import predictCategory
import redis
from flask_cors import CORS
from google_images_search import GoogleImagesSearch
from hellosign_sdk import ApiClient, ApiException, Configuration, apis, models
import requests

subscription_key = "8e2959e6bd2247c28b7c9059fe237b60"
endpoint = "https://sih-vision.cognitiveservices.azure.com//vision/v3.2/analyze?visualFeatures=Tags"



API_KEY = "AIzaSyAVhuJbVRBxIjerR-TuR97vY_ubUeKEfp0"
CX = "145dcb7386f4248c6"
gis = GoogleImagesSearch(API_KEY, CX)


config = {
    "CACHE_TYPE": "RedisCache",
    "CACHE_REDIS_HOST": "localhost",
    "CACHE_REDIS_PORT": "6379",
    "CACHE_REDIS_URL": 'redis://localhost:6379'
}

mongoClinet = pymongo.MongoClient('mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.3.1')
redisclient  = redis.StrictRedis("localhost",port=6379,db=0,charset="utf-8", decode_responses=True)

helloSignConfig = Configuration(
    username="096f2e433b09835cb775448649e17799bb699a49a682ec660218bbc059c8d625",
)


db = mongoClinet['new']

app = Flask(__name__)
CORS(app)

cache = Cache(app, config = config)

def getImage(name):
    API_KEY = "AIzaSyAs0NhZ3xEGNMMXefHuvrHU_3_kkPHYgMM"
    CX = "f76f632302fc44e4c"

    res = requests.get("https://www.googleapis.com/customsearch/v1", params={"key" : API_KEY, "cx" : CX, "q" : name})


    for x in json.loads(res.content)["items"]:

        if "pagemap" in x and "cse_thumbnail" in x["pagemap"] and len(x["pagemap"]["cse_thumbnail"]) > 0 and int(x["pagemap"]["cse_thumbnail"][0]["width"]) < 420:
            return x["pagemap"]["cse_thumbnail"][0]["src"]



    return "https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/No_image_available.svg/1024px-No_image_available.svg.png"

@app.route('/')
def home_page():
    return 'Hello World'


@app.route('/getProduct')
def get_product():
    """
    GET request | Returns product details
    Query Params : barcode
    Returns : json containing all the details
    """
    details = request.args
    barcode = details["barcode"]

    mongoCollection = db['products']

    return dumps(mongoCollection.find_one({'_id' : barcode}))

@app.route('/getProductsByCategory')
def get_product_by_category():
    """
    GET request | Returns product details
    Query Params : categories
    Returns : json containing all the details
    """
    details = request.args
    cats = []
    for x in details["categories"].split(','):
        cats.append(x.strip())

    mongoCollection = db['products']

    return dumps(list(mongoCollection.find({"category" : {"$in" : cats}})))

# Generate a key for product name caching
def productNameKey():
    data = request.args

    return str(data["searchTerm"]).lower()

@app.route('/getProductByName')
@cache.cached(timeout=100, key_prefix=productNameKey)
def get_product_by_name():
    """
    GET request | Returns list of product details
    Query Params : searchTerm
    Returns : json containing all the details
    """
    details = request.args
    searchTerm = details["searchTerm"]

    mongoCollection = db['products']

    res = mongoCollection.find(
        { '$text': { '$search': searchTerm } },
        { 'score': { '$meta': "textScore" } }
    ).sort([('score', {'$meta': 'textScore'})])

    return dumps(list(res))

@app.route('/getProductFromManufacturer')
def get_product_from_manufacturer():
    """
    GET request | Returns list of product details
    Query Params : mid
    Returns : json containing all the details
    """
    details = request.args
    mid = details["mid"]
```

```
143
144     mongoCollection = db['products']
145
146     return dumps(list(mongoCollection.find({"manufacturer" : {"$in" : [mid]}})))
147 # Generate a key for supplier caching
148 def supplierKey():
149     data = request.args
150
151     return str(data["searchTerm"]).lower()
152
153
154 @app.route('/getSuppliers')
155 @cache.cached(timeout=50, key_prefix=supplierKey)
156 def get_supplier():
157     """
158     GET request | Returns supplier details
159     Query Params : searchTerm, latitude, longitude
160     Returns : json containing all the details
161     """
162
163     details = request.args
164
165     term = details["searchTerm"]
166     lat = float(details["latitude"])
167     long = float(details["longitude"])
168
169     res = getSupplierInfo(term,lat,long)
170
171     return json.dumps(res)
172
173 # Generate a key for barcode caching
174 def barcodeKey():
175     data = request.args
176
177     return str(data["barcode"]).lower()
178
179 @app.route('/getProductNameByBarcode')
180 @cache.cached(timeout=0, key_prefix=barcodeKey)
181 def get_product_name_by_barcode():
182     """
183     GET request | Returns barcode product
184     Query Params : barcode
185     Returns : json containing all the details
186     """
187
188     details = request.args
189
190     barcode = details["barcode"]
191
192     return getBarcodeInfo(barcode)
193
194 @app.route('/getProductDetailsByBarcode')
195 def get_product_details_by_barcode():
196     """
197     GET request | Returns barcode product
198     Query Params : barcode
199     Returns : json containing all the details
200     """
201
202     details = request.args
203
204     barcode = details["barcode"]
205
206     mongoCollection = db['products']
207
208     res = mongoCollection.find_one({'_id' : barcode})
209
210     if res:
211         return dumps(res)
212
213     redisRes = redisclient.get("flask_cache_" + barcode)
214
215     if redisRes:
216         name = json.loads(redisRes)["productName"]
217     else:
218         name = getBarcodeInfo(barcode)["productName"]
219
220         redisclient.set("flask_cache_" + barcode, json.dumps({"productName":name,"status":"Passed"}))
221
222     predisRed = redisclient.get("flask_cache_cat" + name.lower())
223
224     imgUrl = getImage(name)
225
226     # t1 = ThreadWithReturnValue(target = getImage, args = (name,))
227
228     # t1.start()
229
230     if predisRed:
231         cats = json.loads(predisRed)
232     else:
233         cats = predictCategory(name)[:10]
234
235         redisclient.set("flask_cache_cat" + name.lower(), json.dumps(cats))
236
237
238
239     catId = '+'.join(cats[:5])
240
241     res = db['categoryEmission'].find_one({'_id' : catId})
242
243     emission = 150
244
245     rating = 2.5
246
247     if res:
248         emission = res['totalEmission'] / res['totalManufacturers']
249
250     # imgUrl = t1.join()
251
252     return dumps({"_id" : "NA", "weight" : -1, "price" : -1,  "category" : cats, "categoryID" : catId,"image_url" : imgUrl, "manufacturer" : [], "name" : name, "rating" : 2.5, "emission" : emission,
253
254 @app.route('/getAllRoutes')
255 def get_all_routes():
256     """
257     GET request | Returns all routes from A to B and its emission
258     Query Params : fromAddress, toAddress
259     Returns : json containing all the details
260     """
261
262     details = request.args
263
264     return calcTotalDistance(details["fromAddress"], details["toAddress"])
265
266
267 @app.route('/getManufacturers')
268 def get_manufacturers():
269     """
270     Depreciated
271     GET request | Returns all Manufacturers
272     Query Params : searchTerm
273     Returns : json containing all the details
274     """
275
276     details = request.args
277
278     searchTerm = details["searchTerm"]
279
280     mongoCollection = db['manufacturers']
281
282     res = mongoCollection.find(
283         { '$text': { '$search': searchTerm } },
284         { 'score': { '$meta': "textScore" } }
```

```
    ).sort([('score', {'$meta': 'textScore'})])

    return dumps(list(res))

@app.route('/getManufacturer')
def get_manufacturer():
    """
    GET request | Returns the Manufacturer details
    Query Params : mid
    Returns : json containing all the details
    """

    details = request.args

    mid = details["mid"]

    mongoCollection = db['manufacturers']

    res = mongoCollection.find_one({"_id" : mid})

    return dumps(res)

# Generate a key for category caching
def categoryKey():
    data = request.args

    return str("cat" + data["searchTerm"]).lower()

@app.route('/getCategories')
@cache.cached(timeout=0,key_prefix=categoryKey)
def get_categories():
    """
    GET request | Returns category through name
    Query Params : searchTerm
    Returns : json array having top 10 categories
    """

    details = request.args

    searchTerm = details["searchTerm"]

    res = predictCategory(searchTerm)

    return dumps(res[:10])


@app.route('/addManufacturer',methods=['POST'])
def add_manufacturer():
    """
    POST request | Adds a Manufacturer
    Json params : [_id,companyName,lat,long,address,phone]
    """

    details = request.json

    mongoCollection = db['manufacturers']

    res = mongoCollection.insert_one(details)

    return jsonify({'_id' : str(res.inserted_id)}) if res.acknowledged else 'Failed'


@app.route('/addProduct',methods=['POST'])
def add_product():
    """
    POST request | Adds a Product
    Json params : [name, weight, price, category, emission, manufacturer, barcode, rawMaterials, components]
    """

    details = request.json
    mongoCollection = db['products']

    # Details init
    name = details["name"]
    cats = details["category"]
    emission = details["emission"]
    mid = details["manufacturer"]
    barcode = details["barcode"]
    rawMaterials = details['rawMaterials']
    weight = details["weight"]
    price = details["price"]
    components = details["components"]

    imgUrl = getImage(name)

    # Create a category ID
    catId = str(hash("+".join(cats).replace(" ", "")))

    # Increment for that category using upsert
    db['categoryEmission'].update_one({'_id' : catId}, {'$inc' : {'totalEmission' : emission, 'totalManufacturers' : 1, }}, upsert=True)

    # Get total value
    query = db['categoryEmission'].find_one({'_id' : catId})

    tot = query['totalEmission']
    ct = query["totalManufacturers"]

    # Get rating by using a stat formula
    rating = 5 * (tot) / (tot + (emission * (ct - 1)))

    res2 = db['manpro'].insert_one({'category' : cats,'categoryID': catId, 'barcode' : barcode, 'name' : name, 'emission' : emission , 'manufacturer' : mid, 'rawMaterials' : rawMaterials, 'components

    # Push the product details
    f = mongoCollection.find_one({'_id' : barcode})

    if f is None:
        mongoCollection.insert_one({'_id' : barcode, 'category' : cats, 'name' : name, 'categoryID': catId,'image_url' : imgUrl, 'weight' : weight, "price" : price, 'rating' : rating})

        for m in rawMaterials:

            mongoCollection.update_one({'_id' : barcode}, {'$push': {'rawMaterials': m}})


        for m in components:

            mongoCollection.update_one({'_id' : barcode}, {'$push': {'components': m}})


    res = mongoCollection.update_one({'_id' : barcode},
    {
        '$set' : {'rating' : rating},
        '$push' : {'manufacturer' : mid},
        '$inc' : {'totalManufacturers' : 1, 'totalEmission' : emission},

    }, upsert= True)

    # Insert into a collection that has mid and pid present linking each person with their product
    res2 = db['manpro'].insert_one({'category' : cats,'categoryID': catId, 'barcode' : barcode, 'name' : name, 'emission' : emission , 'manufacturer' : mid, 'rawMaterials' : rawMaterials, 'components

    # Update each product in the category to maintain dynamic scoring
    for row in mongoCollection.find({'categoryID' : catId}):

        bc = row['_id']

        avgEmission = row['totalEmission'] / row['totalManufacturers']

        rowRating = 5 * (tot) / (tot + (avgEmission * (ct - 1)))


        mongoCollection.update_one({'_id' : bc}, {'$set' : {'rating' : rowRating}})

    # Append product into the manufacturer thingy
```

```
    db["manufacturers"].update_one({'_id' : mid}, {'$push' : {'products' : barcode}})

    details = db["manufacturers"].find_one({'_id' : mid})

    JWT = "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJIZWxsb1dvcmtzIiwiZXhwIjoxNjYyMjMxODM4LCJpYXQiOjE2NjIyMzAwMzgsImlzcyI6IkhlbGxvV29ya3MiLCJqdGkiOiJmNjUwZmZiYy0wZDU5LTQzOTEtODliOS1jMjczZTRmNTF

    headers = {
        'Authorization': f"Bearer {JWT}",
        'Content-Type': 'application/x-www-form-urlencoded',
    }

    data = f'workflow_id=JTvkdP6QUlFrYQs8&participants[participant1_g2xB3p][type]=email&participants[participant1_g2xB3p][value]=1ms19is051@gmail.com&participants[participant1_g2xB3p][full_name]={de

    response = requests.post('https://api.helloworks.com/v3/workflow_instances', headers=headers, data=data)
    print(data)
    print(response.text)
    return jsonify({'status' : str(res.acknowledged and res2.acknowledged)})



# returns the shipment statuses to the manufacturer
@app.route('/getShipments')
def get_shipments():
    """
    GET request | Returns shipment details for a manufacturer
    Query Params : manufacturer
    Returns : json containing all the details
    """

    details = request.args

    manufacturer = details["manufacturer"]

    mongoCollection = db['shipments']

    res = mongoCollection.find({'manufacturer' : manufacturer})

    return dumps(list(res))



@app.route('/addShipment',methods=['POST'])
def add_shipment():
    """
    POST request | Adds a shipment
    Json params : [manufacturer, startLocation, pid, totalWeight, currentLat, currentLong]
    dlang and dlong are destination lat and long
    currentLat and currentLong are the lat and long of where the product is added
    jounrey shall be ["Bangalore", "Delhi", "Kolkata"] basically the location names where all it reaches
    """

    details = request.json

    manufacturer = details['manufacturer']

    details["status"] = "PROCESSING"
    details["timestamp"] = datetime.now()
    details["journey"] = [details["startLocation"]]
    details["transportMode"] = "-"
    details["enroute_to"] = "-"
    details["emission"] = 0 # just the carbon so far used. in update we will update this

    mongoCollection = db['shipments']
    res = mongoCollection.insert_one(details)

    db['manufacturers'].update_one({'_id':  manufacturer}, {"$push":{"currentShipments": res.inserted_id} },upsert=True)

    return dumps({'id' : res.inserted_id})

def distance(lat1, lon1, lat2, lon2):

    p = pi/180
    a = 0.5 - cos((lat2-lat1)*p)/2 + cos(lat1*p) * cos(lat2*p) * (1-cos((lon2-lon1)*p))/2

    return 12742 * asin(sqrt(a))

@app.route('/updateShipment',methods=['POST'])
def update_shipment():
    """
    POST request | Update a shipment
    Json params : [shipmentID, location, transportMode, currentLat, currentLong, enroute_to, status]
    """

    details = request.json

    status = details["status"]

    shipmentID = ObjectId(details["shipmentID"])

    res = db['shipments'].find_one({'_id' : shipmentID})

    prevMode = res["transportMode"]

    emission = 0

    if res["status"] == "TRAVEL":

        prevLat, prevLong = res["currentLat"], res["currentLong"]
        curLat, curLong = details["currentLat"], details["currentLong"]

        if prevMode == "AIR": emission += Emission.airEmission(distance(curLat, curLong,prevLat, prevLong))

        elif prevMode == "WATER" : emission += Emission.waterEmission(distance(curLat, curLong,prevLat, prevLong))

        elif prevMode == "RAIL" : emission += Emission.railEmission(distance(curLat, curLong,prevLat, prevLong))

        elif prevMode == "ROAD" : emission += Emission.roadEmission(distance(curLat, curLong,prevLat, prevLong))



        db['shipments'].update_one(
            {'_id' : shipmentID},
            {
                "$set" : {
                    "transportMode" : details["transportMode"],
                    "status" : status,
                    "currentLat" : details["currentLat"],
                    "currentLong" : details["currentLong"],
                    "enroute_to" : details["enroute_to"]
                },
                '$push' : {'journey' : details["location"]},
                '$inc' : {'emission' : emission },
            }
        )

    else:

        db['shipments'].update_one(
            {'_id' : shipmentID},
            {
                "$set" : {
                    "transportMode" : details["transportMode"],
                    "status" : status,
                    "currentLat" : details["currentLat"],
                    "currentLong" : details["currentLong"],
                    "enroute_to" : details["enroute_to"]
                },
                '$inc' : {'emission' : emission },
            }
        )

        if (status == "DELIVERED"):
```

```
            state = geocoder.osm([details["currentLat"], details["currentLong"]], method='reverse').state

            ship = db["shipments"].find_one({"_id":ObjectId(details["shipmentID"])})

            prod = db["products"].find_one({"_id" : ship["pid"]})

            db["categoryEmission"].update_one({"_id" : prod["categoryID"]}, {"$inc" : {"states." + state.lower() : 1}}, upsert = True)


    return jsonify({'status' : True})



@app.route('/detectImage',methods=['POST'])
def detect_image():
    direc = os.getcwd()
    path = os.path.join(direc,'test.jpg')

    if 'Picture' not in request.files:
        return "someting went wrong 1"

    user_file = request.files['Picture']


    user_file.save(path)

    with open(path, 'rb') as f:
        data = f.read()
        headers = {
            'Ocp-Apim-Subscription-Key': subscription_key,
            'Content-type': 'application/octet-stream'
        }
        response = requests.post(
            endpoint, headers=headers, data=data)


    # Print results with confidence score
    # print("Tags in the remote image: ")
    # if (len(tags_result_remote.tags) == 0):
    #     print("No tags detected.")
    # else:
    #     for tag in tags_result_remote.tags:
    #         print("'{}' with confidence {:.2f}%".format(tag.name, tag.confidence * 100))
    # print()

        return jsonify({"object" : response.json()["tags"][0]["name"]})

@app.route('/addInspectionForm',methods=['POST'])
def add_inspection_form():
    """
    POST request | Add an inspection form
    Json params : ["manufacturer", "inputFields", "inputTypes", "formName", "makerName"]
    """

    details = request.json

    mname = details["manufacturer"]

    mid = db["manufacturers"].find(
        { '$text': { '$search': mname } },
        { 'score': { '$meta': "textScore" } }
    ).sort([('score', {'$meta': 'textScore'})])[0]["_id"]

    formName = details["formName"]

    makerName = details["makerName"]

    inputFields = details["inputFields"]

    inputTypes = details["inputTypes"]

    cats = set()

    pids = set(db["manufacturers"].find_one({"_id" : mid})["products"])

    for i in pids:

        pcats = db["products"].find_one({"_id" : i})["category"]

        for c in pcats:

            cats.add(c)


    res = db["forms"].insert_one({"manufacturer" : mid, "inputFields" : inputFields, "inputTypes" : inputTypes, "formName" : formName, "makerName" : makerName, "targetCategories" : list(cats)})

    d = {
        "template_ids": [
            "28b41595c417b0023384b0c04f23e38f169d4fb3"
        ],
        "subject": "E-Signature for audit confirmation",
        "message": "Please sign this form so that we can confirm your submission of the audit format.\nThis shall help us take a step towards a better tomorrow.\nThanks & Regards,\nTeam EcoTag",
        "signers": [
            {
            "role": "Auditor",
            "name": makerName,
            "email_address": "1ms19is051@gmail.com"
            }
        ],
        "custom_fields": [
            {
            "name": "Factory",
            "value": mname,
            "editor": "Auditor",
            "required": True
            },
            {
            "name": "Name1",
            "value": makerName,
            "editor": "Auditor",
            "required": True
            },
            {
            "name": "Name2",
            "value": makerName,
            "editor": "Auditor",
            "required": True
            }
        ],
        "signing_options": {
            "draw": True,
            "type": True,
            "upload": True,
            "phone": False,
            "default_type": "draw"
        },
        "test_mode": True
    }
    headers = {'Content-type': 'application/json'}

    resp = requests.post("https://096f2e433b09835cb775448649e17799bb699a49a682ec660218bbc059c8d625:@api.hellosign.com/v3/signature_request/send_with_template", data = json.dumps(d), headers=headers)


    return json.dumps({"status" : res.acknowledged })



@app.route('/getInspectionForms')
def get_inspection_forms():

    return dumps(list(db["forms"].find()))
```

```

@app.route('/getStates')
def get_states():

    details = request.args

    return dumps(db["categoryEmission"].find_one({"_id" : details["categoryID"]}))


@app.route('/reverseLoc')
def reverse_loc():

    details = request.args

    return dumps({"state" : geocoder.osm([details["lat"], details["long"]], method='reverse').state})


if __name__ == "__main__":
    app.run('0.0.0.0',80,True)
```

```
// Authored By: Jayant Sogikar
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>

int main(){

    // Init a parallel section
    #pragma omp parallel
    {
            // Finding thread number
            int t = omp_get_thread_num();

            // output
            printf("Hello world from thread number %d \n",t);
    }

    return 0;
}

// gcc hw.c -fopenmp
// ./a.out
```

```
# Authored By: Jayant Sogikar
from collections import defaultdict


def solution(map):

    fromA = defaultdict(lambda : float('inf'))
    fromB = defaultdict(lambda : float('inf'))

    r, c = len(map) , len(map[0])
    fromA[(0,0)] = 1

    q = [(0,0,1)]

    while q:
        x, y, score = q.pop(0)



        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
            if 0 <= x + i < r and 0 <= y + j < c and fromA[(x + i,y + j)] == float('inf'):

                fromA[(x + i,y + j)] = score + 1

                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))


    q = [(r - 1, c - 1, 1)]
    fromB[(r-1,c-1)] = 1

    while q:
        x, y, score = q.pop(0)


        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
            if 0 <= x + i < r and 0 <= y + j < c and fromB[(x + i,y + j)] == float('inf'):

                fromB[(x + i,y + j)] = score + 1

                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))


    minVal = float('inf')



    for i in range(r):
        for j in range(c):
            # print(fromA[(i,j)], fromB[(i,j)], fromA[(i,j)] + fromB[(i,j)])
            minVal = min(minVal, fromA[(i,j)] + fromB[(i,j)])


    return minVal - 1




print(solution([[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]))
```