import azure.functions as func

from flask import Flask,request,jsonify

import json

import requests

from datetime import timedelta,datetime

import os

from dotenv import load\_dotenv

import logging

from applicationinsights import TelemetryClient

import threading

app = Flask(\_\_name\_\_)

load\_dotenv()

INSTRUMENTATION\_KEY = os.getenv('APPINSIGHTS\_INSTRUMENTATIONKEY')

tc = TelemetryClient(INSTRUMENTATION\_KEY)

@app.route("/github/get\_inactive\_repos", methods=['GET'])

def getInactiveRepositories():

    try:

        function\_key = os.getenv('FUNCTION\_KEY')

        base = os.getenv('GITHUB\_API\_URL')

        base\_url = f"{base}graphql?code={function\_key}"

        github\_url = os.getenv('GITHUB\_API\_URL')

        organization = os.getenv('ORGANIZATION')

        github\_token = os.getenv('GITHUB\_TOKEN')

        HEADERS = {'Authorization': f'Bearer {github\_token}'}

        # condition = request.headers.get('inactive\_days')

        condition = os.getenv('Inactive\_Days', 365)

        message = {'message':'Header not found'}

        def log\_to\_application\_insights(repo\_info):

                tc.track\_event('InactiveRepos', repo\_info)

                tc.flush()

        if condition is None:

            return jsonify({"error": "inactive\_days header is missing"}), 400

        cutoff\_date = (datetime.now() - timedelta(days=int(condition))).isoformat()

        query\_template = """

        {{

            search(query: "org:{organization} pushed:<{cutoff\_date}", type: REPOSITORY, first: 100, after: "{after\_cursor}") {{

                edges {{

                    node {{

                        ... on Repository {{

                            name

                            description

                            url

                            isPrivate

                            pushedAt

                            updatedAt

                            owner {{

                                login

                            }}

                        }}

                    }}

                }}

                pageInfo {{

                    hasNextPage

                    endCursor

                }}

            }}

        }}

        """

        after\_cursor = ""

        all\_repos = []

        logging.info(f"search repo query - {query\_template}")

        while True:

            query = query\_template.format(cutoff\_date=cutoff\_date, organization=organization, after\_cursor=after\_cursor)

            response = requests.post(base\_url, json={'query': query}, headers=HEADERS)

            if response.status\_code == 200:

                logging.info(f"All the inactive repos are retrived with status code - {response.status\_code}")

                result = response.json()

                logging.info(f"Response JSON: {result}")

                repos = result["data"]["search"]["edges"]

                all\_repos.extend(repos)

                page\_info = result["data"]["search"]["pageInfo"]

                if not page\_info["hasNextPage"]:

                    break

                after\_cursor = page\_info["endCursor"]

            else:

                logging.info(f"Query failed to run by returning code of {response.status\_code}. {query}")

                return jsonify(f"Query failed to run by returning code of {response.status\_code}. {query}"), 500

        inactive\_repos = []

        for repo in all\_repos:

            url= github\_url+"repos/"+ organization +"/"+ repo['node']['name']+"/collaborators"

            logging.info(f"collaborators\_url - {url}")

            adminResponse = requests.get(url, headers=HEADERS)

            collaborators = adminResponse.json()

            logging.info(f"Collaborators JSON: {collaborators}")

            admin\_usernames = [collaborator['login'] for collaborator in collaborators if collaborator['permissions']['admin']]

            if(repo['node']['description'] == 'null' or repo['node']['description'] is None):

                description = ''

            else:

                description =  repo['node']['description']

            pushed\_date = repo['node']['pushedAt']

            url = f"{github\_url}repos/{organization}/{repo['node']['name']}?code={function\_key}"

            logging.info(f"archieve url - {url}")

            response = requests.get(url, headers=HEADERS)

            logging.info(f"archieve url status\_code - {response.status\_code}")

            if response.status\_code == 200:

                repo\_info = response.json()

                logging.info(f"Repo info: {repo\_info}")

            else:

                logging.error(f"Failed to retrieve repo info for {repo['node']['name']}. Status code: {response.status\_code}")

                continue

            if pushed\_date and pushed\_date < cutoff\_date and repo\_info['archived'] == False:

                repo\_info = {

                    'Name': repo['node']['name'],

                    "Url": repo['node']['url'],

                    "Updated": repo['node']['updatedAt'],

                    "Pushed": repo['node']['pushedAt'],

                    "Owner": repo['node']['owner']['login'],

                    "Description":description,

                    "env": base,

                    'application': 'GitHub',

                    'operation': 'InactiveRepos'

                }

                inactive\_repos.append(repo\_info)

                log\_to\_application\_insights(repo\_info)

        logging.info(f"Total number of inactive repos fetched: {len(inactive\_repos)}")

        return jsonify(inactive\_repos)

    except Exception as e:

            logging.error(f"Exception in getting inactive github repo.Error message: {e}")

            return jsonify({'error': f"Oops! Something bad happened. Error: {str(e)}"}), 500

def main(req: func.HttpRequest, context: func.Context) -> func.HttpResponse:

    return func.WsgiMiddleware(app.wsgi\_app).handle(req, context)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run()