from flask import Flask, request, jsonify, render\_template, url\_for

import numpy as np

from PIL import Image

import tensorflow as tf

from tensorflow import keras

import os

import random

import sys

import pickle

import logging

import hashlib

# intialize logger

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

# intialize flask app

app = Flask(\_\_name\_\_, static\_folder='mri-images')

CNN = None  # global variable to hold the CNN model

def load\_model():

    global CNN

    if CNN is not None:

        return  # model already loaded

    script\_dir = os.path.dirname(\_\_file\_\_)

    model\_json\_path = os.path.join(script\_dir, 'models', 'CNN\_structure.json')

    with open(model\_json\_path, 'r') as json\_file:

        model\_json = json\_file.read()

    try:

        # load model

        CNN = tf.keras.models.model\_from\_json(model\_json)

        # load and set model weights

        weights\_path = os.path.join(script\_dir, 'models', 'CNN\_weights.pkl')

        with open(weights\_path, 'rb') as weights\_file:

            weights = pickle.load(weights\_file)

            CNN.set\_weights(weights)

        # compile model

        CNN.compile(optimizer=tf.keras.optimizers.Adamax(learning\_rate=0.001),

                    loss='categorical\_crossentropy',

                    metrics=['accuracy'])

    except Exception as e:

        logger.error(f"Error loading model: {e}")

# function for retrieving prediction from model given an image path

def get\_model\_prediction(image\_path):

    load\_model()

    try:

        # load and preprocess the image

        img = Image.open(image\_path).resize((224, 224))

        # convert grayscale images to RGB

        if img.mode != 'RGB':

            img = img.convert('RGB')

        img\_array = np.expand\_dims(np.array(img), axis=0)

        # predict using the CNN model

        prediction = CNN.predict(img\_array)

        # interpret the prediction

        predicted\_index = np.argmax(prediction[0])

        class\_labels = ['glioma', 'meningioma', 'no tumor', 'pituitary']

        predicted\_class = class\_labels[predicted\_index]

        return predicted\_class

    except Exception as e:

        logger.error(f"Error in get\_model\_prediction: {e}")

        return None

# load html template

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/get-random-image', methods=['GET'])

def get\_random\_image():

    try:

         # select a random directory and then a random image within the image directory

        class\_dirs = ['glioma', 'meningioma', 'notumor', 'pituitary']

        selected\_class = random.choice(class\_dirs)

        image\_dir = os.path.join('mri-images', selected\_class)

        image\_name = random.choice(os.listdir(image\_dir))

        image\_path = os.path.join(image\_dir, image\_name)

        predicted\_label = get\_model\_prediction(image\_path)

        web\_accessible\_image\_path = url\_for('static', filename=f'{selected\_class}/{image\_name}')

        return jsonify({

            'image\_path': web\_accessible\_image\_path,

            'actual\_label': selected\_class,

            'predicted\_label': predicted\_label

        })

    except Exception as e:

        logger.error(f"Error in get-random-image route: {e}")

        return jsonify({'error': 'An error occurred'}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=False)