#Author : Authors of Learn & Practice AI by Projects

#Purpose : RabbitMQ server side application to serve the user request for IRIS model's prediction request.

#Only one instance of this application will be active to listen to the users' prediction request, enqueue the request

#into message queue and give prediction result to the caller.

#import packages

from fastapi import FastAPI

import uvicorn

import pika

import json

#shared queue name

SHARED\_REQUEST\_QUEUE = 'SHARED\_REQUEST\_CLASSIFICATION\_MODEL\_QUEUE';

#Constant string for Job Id and prediction request parameters.

JOBID\_KEY = 'JOBID';

SEPAL\_LENGTH = 'SEPAL\_LENGTH';

SEPAL\_WIDTH = 'SEPAL\_WIDTH'

PETAL\_LENGTH = 'PETAL\_LENGTH';

PETAL\_WIDTH = 'PETAL\_WIDTH';

#FastAPI instance created.

fastAPIInstance = FastAPI();

#Job Id counter initialized to 0.

jobIDCounter = 0;

#dictionary object crated.

dictPredictionResult = {};

#connection parameters required to connect to RabbitMQ

requestConnectionParameters = pika.ConnectionParameters(host='localhost');

#BlockingConnection creates layer on top of Pika's Async core to return response.

requestConnection = pika.BlockingConnection(requestConnectionParameters);

#create request channel

requestChannel = requestConnection.channel();

#define the request queue

requestChannel.queue\_declare(queue=SHARED\_REQUEST\_QUEUE, durable=True);

print(" [\*\*\*\*] Server application started...Press CTRL+C to stop the server.")

#construct the json request using given job id and prediction request

def getJSONRequest(job\_id, sepal\_length:float, sepal\_width:float, petal\_length:float, petal\_width:float):

data ={};

data[JOBID\_KEY] = job\_id;

data[SEPAL\_LENGTH] = sepal\_length;

data[SEPAL\_WIDTH] = sepal\_width;

data[PETAL\_LENGTH] = petal\_length;

data[PETAL\_WIDTH] = petal\_width;

jsonData = json.dumps(data);

return jsonData;

#API for querying the prediction result for the given Job Id.

#This API will be called by the Client appliaction to know the status of prediction request.

@fastAPIInstance.get('/prediction-result')

def prediction\_result(job\_id:int):

global dictPredictionResult;

if job\_id not in dictPredictionResult:

return 'Unknown Job Id....';

return dictPredictionResult[job\_id];

#API for updating the predicted result detail in the server application

#This API will be called by the worker application to assign the predicted result against the job id.

@fastAPIInstance.get('/update-predicted-result')

def updated\_predicted\_result(job\_id:int, predicted\_result:str):

global dictPredictionResult;

#updating the prediction result for the given Job Id.

dictPredictionResult[job\_id] = predicted\_result;

print("<-<-<-Updated prediction Result for Job Id {} is {} ".format(job\_id,dictPredictionResult));

return "{Status : Success}";

#API for submitting the prediction request to the server application.

#This API will be called by the client application to submit the prediction request.

@fastAPIInstance.get('/predict-class')

def predict\_class(sepal\_length:float, sepal\_width:float, petal\_length:float, petal\_width:float):

global jobIDCounter;

#increment the Job ID value.

jobIDCounter = jobIDCounter + 1;

global dictPredictionResult;

#call getJSONRequest to construct the JSON object from the reqeust list.

jsonRequest = getJSONRequest(jobIDCounter, sepal\_length, sepal\_width, petal\_length, petal\_width);

#print the request detail in JSON format.

print("->->->Received prediction request : {}".format(jsonRequest));

#Enqueue the task in JSON format. This task will be picked by worker application for processing.

requestChannel.basic\_publish(exchange='', routing\_key=SHARED\_REQUEST\_QUEUE, body=jsonRequest,

properties=pika.BasicProperties(delivery\_mode=2, # make message persistent

));

#store the "IN-PROGRESS" value against the current job id. This "IN-PROGRESS" value will be relaced with the correct

#predicted value once the result is available.

dictPredictionResult[jobIDCounter] = 'IN-PROGRESS';

return jsonRequest;