from flask import jsonify, g, request

import jwt

from jwt import DecodeError, ExpiredSignature

from datetime import datetime, timedelta

import calendar

from functools import wraps

import os

SECRET\_KEY = 'xuqgy’

def create\_token(username):

payload = {

'sub': username,

'role':'agent',

'aud': "usr",

'iat': calendar.timegm(datetime.utcnow().timetuple()),

'exp': calendar.timegm((datetime.utcnow() + timedelta(days=(365))).timetuple()),

'iss': 'ipv\_api\_IonicApp'

}

print (calendar.timegm(datetime.utcnow().timetuple()))

print (calendar.timegm((datetime.utcnow() + timedelta(minutes=(30))).timetuple()))

token = jwt.encode(payload, SECRET\_KEY, algorithm='HS256')

return token.decode('unicode\_escape')

def parse\_token(req, permission):

token = req.headers.get('auth')

#print ("auth-->{}".format(token))

return jwt.decode(token, SECRET\_KEY, algorithms='HS256', audience=permission)

def login\_required(\*args1, \*\*kwargs1):

def login\_decorator(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

#print ("request header--{}".format(request.headers))

if not request.headers.get('auth'):

response = jsonify(message='Missing authorization header')

print("No header passed")

response.status\_code = 401

return response

try:

payload = parse\_token(request,kwargs1['permission'])

if payload["role"] != "agent" and "get\_schedules" not in request.url:

response = jsonify(message='invalid role')

response.status\_code = 401

return response

except DecodeError as d:

response = jsonify(message='Token is invalid')

print("DecodeError",d)

response.status\_code = 401

return response

except ExpiredSignature as es:

response = jsonify(message='Token has expired')

print("expired signature",es)

response.status\_code = 401

return response

except jwt.InvalidAudienceError:

response = jsonify(message='Insufficient access level')

print("insufficient access")

response.status\_code = 401

return response

except jwt.InvalidIssuedAtError:

response = jsonify(message='Issued at time doesnt look right')

print("invalid issue time")

response.status\_code

return response

except jwt.InvalidIssuerError:

response = jsonify(message='Doesnt look like we issued this token')

print("token missing")

response.status\_code

return response

# if env.lower()=="prod":

# g.user\_id = cname.lower()

# else:

# g.user\_id="{}\_{}".format(cname.lower(),env.lower())

g.user\_id = payload['sub']

return f(\*args, \*\*kwargs)

return decorated\_function

return login\_decorator

#Api to create token

@bp.route('/generate\_token', methods=['POST'])

def generate\_token():

data = request.get\_json()

username=data[“username”]

response = create\_token(login\_username)

return ( jsonify({'Token': response,

'status\_code': 200,

'success': True}), 200)

##to call API with authentication token

# 1st call generate\_token api with username in request body

#from response take Token and set it in headers

#url = "[http://localhost:7865/generate\_token](http://35.154.77.106:7865/generate_token)'"

#headers = {

'Content-Type': "application/json"

}

#data ={"username":"abc"}

#payload = json.dumps(data)

#response = requests.request("POST",

url,

data=payload, headers=headers)

#auth\_token = json.loads(response.content)['Token']

##now use this in API calls

#url = "[http://localhost:7865/](http://35.154.77.106:7865/generate_token)test\_1"

#headers = {

‘auth’:auth\_token

'Content-Type': "application/json"

}

#data ={}

#payload = json.dumps(data)

#response = requests.request("POST",

url,

data=payload, headers=headers)

## way to use login restriction using decorator

@bp.route('/test\_1', methods=['POST'])

@login\_required(permission='usr')

def get\_stage1\_data():

user\_name= g.user\_id