Tarp Digital Assignment 10

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: TF2

Backend Connection Code

```
import random
import json
import os
import traceback
import tornado.httpserver
import tornado.ioloop
import tornado.web
from motor import motor_tornado
from tornado.gen import coroutine
from tornado.options import define, options
from models import users, patient
from passlib.hash import pbkdf2_sha256
import tornado.escape
import jwt
from functools import wraps
from oauth2client import client as auth_client
from oauth2client import crypt
from io import BytesIO
from PIL import Image
import base64
import requests

define("port", default=8080, help="runs on the given port", type=int)


class MyAppException(tornado.web.HTTPError):
    pass


def protected(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        token = tornado.escape.json_decode(args[0].request.body)
        decoded_token = jwt.decode(token['token'], "abc")
        if decoded_token['valid']:
            return f(*args, **kwargs)
        else:
            raise MyAppException(reason="Invalid Token",
status_code=401)
    return wrapper


def Authenticated(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        # try:
        #     token = args[0].get_argument('token')
        # except tornado.web.MissingArgumentError:
        token = tornado.escape.json_decode(args[0].request.body)
        decoded_token = jwt.decode(token['token'], "abc")
        if decoded_token['user'] in args[0].settings['logged_in']:
            return f(*args, **kwargs)
        else:
            raise MyAppException(reason='User is not logged in.',
status_code=301)
    return wrapper


class BaseHandler(tornado.web.RequestHandler):

    def db(self):
        clientz = self.settings['db_client']
        db = clientz.tornado
        return db

    def write_error(self, status_code, **kwargs):
        self.set_header('Content-Type', 'application/json')
        if self.settings.get("serve_traceback") and "exc_info" in
kwargs:
            # in debug mode, try to send a traceback
            lines = []
            for line in
traceback.format_exception(*kwargs["exc_info"]):
                lines.append(line)
            self.write(json.dumps({
                        'status_code': status_code,
                        'message': self._reason,
                        'traceback': lines,
                }))
        else:
            self.write(json.dumps({
                    'status_code': status_code,
                    'message': self._reason,
                }))


class AuthHandler(BaseHandler):
    @coroutine
    def post(self):
        self.set_header('Content-Type', 'application/json')
        login_data = tornado.escape.json_decode(self.request.body)
        if not login_data:
            raise MyAppException(reason="Invalid Credentials",
status_code=400)
        db_client = self.db()
        hash = yield db_client.auth.find_one({"user":
login_data['user']})
        if not hash:
            raise MyAppException(reason="Invalid Credentials",
status_code=400)
        if 'google' in hash:
            self.write(json.dumps({
                'message': 'Invalid Credentials',
                'status_code': 400,
            }))
        if pbkdf2_sha256.verify(login_data['pass'], hash["pass"]):
            flag = {'status_code': 200, 'message': 'valid', 'portal':
hash['portal']}
        else:
            raise MyAppException(reason="Invalid Credentials",
status_code=400)
        if str(flag['portal']) == "1":
            response = yield db_client.patient.find_one({"user":
login_data['user']})
        else:
            response = yield db_client.doctor.find_one({"user":
login_data['user']})
        response.pop('_id', None)
        response['valid'] = True
        token = jwt.encode(response, "abc")
        self.write(json.dumps({
            'message': 'Logged in Successfully.',
            'status_code': 200,
            'token': token.decode('utf-8')
        }))


class SignUpHandler(BaseHandler):
    @coroutine
    def post(self):
        self.set_header('Content-Type', 'application/json')
        signup_data = tornado.escape.json_decode(self.request.body)
        if signup_data is None:
            raise MyAppException(reason='Invalid Data Given.',
status_code=400)
        db_client = self.db()
        database_auth = db_client["auth"]
        database_details = None
        if str(signup_data['portal']) == "1":
            database_details = db_client["patient"]
            signup_data['ap_details'] = list()

        elif str(signup_data['portal']) == "0":
            database_details = db_client["doctor"]
            signup_data['plist'] = list()
            signup_data['availability'] = True
        signup_data['ap_inactive'] = list()

        if database_details is None:
            raise MyAppException(reason='Portal Not Set.',
status_code=400)
        find_email = yield database_auth.find_one({"user":
signup_data['user']})
        find_user = yield database_details.find_one({'email':
signup_data['email']})
        if find_email:
            raise MyAppException(reason='User Exists', status_code=400)
        if find_user:
            raise MyAppException(reason='Email Exists',
status_code=400)
        if 'google' in signup_data:
            hash_pass = 'google'
        else:
            hash_pass = pbkdf2_sha256.hash(signup_data['pass'])
        signup_data.pop('pass', None)
        signup_data['valid'] = True
        token = jwt.encode(signup_data, "abc")
        signup_data.pop('valid', None)
        if 'google' in signup_data:
            database_auth.insert_one({"user": signup_data['user'],
"pass": hash_pass, "portal": signup_data['portal'],
                                      "email": signup_data['email'],
"google": True})
        else:
            database_auth.insert_one({"user": signup_data['user'],
"pass": hash_pass, "portal": signup_data['portal'],
                                      "email": signup_data['email']})
        database_details.insert_one(signup_data)
        self.write(json.dumps({
            'status_code': 200,
            'message': 'Sign-up successful.',
            'token': token.decode('utf-8')
        }))


class my404handler(BaseHandler):
    def get(self):
        self.set_header('Content-Type', 'application/json')
        self.write(json.dumps({
                'status_code': 404,
                'message': 'illegal call.'
        }))


class OauthLogin(BaseHandler):
    @coroutine
    def post(self):
        self.set_header('Content-Type','application/json')
        token = tornado.escape.json_decode(self.request.body)
        try:
            idinfo = auth_client.verify_id_token(token['token'],
self.settings['client_id'])
        except crypt.AppIdentityError:
            idinfo = None
        if idinfo is None:
            raise MyAppException(reason="Invalid Token",
status_code=400)
        elif idinfo['iss'] not in ['accounts.google.com',
'https://accounts.google.com']:
            raise MyAppException(reason="wrong Issuer",
status_code=400)
        db = self.db()
        find_email = yield db.auth.find_one({"email": idinfo['email']})
        if find_email:
            if str(find_email['portal']) == "1":
                det = yield db.patient.find_one({"email":
idinfo['email']})
            else:
                det = yield db.doctor.find_one({"email":
idinfo['email']})
            det.pop('_id', None)
            det['valid'] = True
            tokenz = jwt.encode(det, "abc")
            self.write(json.dumps({
                "message": "Logged In",
                "token": tokenz.decode('utf-8'),
                "status_code": 200
            }))
        else:
            details = dict(email=idinfo['email'], name=idinfo['name'],
google=True)
            tokenz = jwt.encode(details, "abc")
            self.write(json.dumps({
                "message": "Signup required",
                "status_code": 301,
                "token": tokenz.decode('utf-8')
            }))


class LogoutHandler(BaseHandler):
    @coroutine
    @protected
    def post(self):
        self.set_header('Content-type', 'application/json')
        token = tornado.escape.json_decode(self.request.body)
        decoded_token = jwt.decode(token['token'], "abc")
        self.write(json.dumps({
            'message': 'Logged Out.',
            'status_code': 200
        }))


class NewTokenGenerator(BaseHandler):
    @coroutine
    @protected
    def post(self):
        self.set_header('Content-type', 'application/json')
        token = tornado.escape.json_decode(self.request.body)
        decoded_token = jwt.decode(token['token'], "abc")
        db = self.db()
        if str(decoded_token['portal']) == "1":
            response = yield db.patient.find_one({"user":
decoded_token['user']})
        else:
            response = yield db.doctor.find_one({"user":
decoded_token['user']})
        response.pop('_id', None)
        response['valid'] = True
        refreshed_token = jwt.encode(response, "abc")
        self.write(json.dumps({
            'status_code': 200,
            'message': 'Token Generated.',
            'token': refreshed_token.decode('utf-8')
        }))
```

```
class AppointmentHandler(BaseHandler):
    @coroutine
    @protected
    def post(self):
        self.set_header('Content-type', 'application/json')
        token = tornado.escape.json_decode(self.request.body)
        data = jwt.decode(token['token'], "abc")
        data['description'] = token['description']
        flag = yield patient.make_appointment(self.db(), data)
        if flag['status_code'] == 200:
            self.write(json.dumps(flag))
        else:
            raise MyAppException(reason=flag['message'],
status_code=flag['status_code'])


class ResolveAppointment(BaseHandler):
    @coroutine
    @protected
    def post(self):
        """
        removes appointment for doctor
        makes the status of the appointment done for patient
        """
        self.set_header('Content-type','application/json')
        token = tornado.escape.json_decode(self.request.body)
        db = self.db()
        user = token['user']
        doctor = token['doctor']
        doc = yield db.doctor.find_one({"user": doctor})
        pat = yield db.patient.find_one({"user": user})

        for record in doc['plist']:
            if pat['user'] == record['user']:
                doc['plist'].remove(record)
                doc['ap_inactive'].append(record)
        plist = doc['plist']
        ap_inactive1 = doc['ap_inactive']

        for record in pat['ap_details']:
            if doc['user'] == record['user']:
                pat['ap_details'].remove(record)
                pat['ap_inactive'].append(record)
        ap_details = pat['ap_details']
        ap_inactive2 = pat['ap_inactive']

        db.patient.update({'_id': pat['_id']}, {'$set': {'ap_details':
ap_details, 'ap_inactive': ap_inactive2}}, upsert=False)
        db.doctor.update({'user': doc['user']}, {'$set': {'plist':
plist, 'ap_inactive': ap_inactive1}}, upsert=False)
        self.write(json.dumps({
            'status_code': 200,
            'message': 'Updated Successfully.'
        }))


class MLHandler(tornado.web.RequestHandler):
    def set_default_headers(self):
        self.set_header("Access-Control-Allow-Origin", "*")
        self.set_header("Access-Control-Allow-Headers", "x-requested-
with")
        self.set_header('Access-Control-Allow-Methods', 'POST, GET,
OPTIONS')
        self.set_header('Content-type','application/json')

    @coroutine
    def post(self):
        self.write(json.dumps({"confidence": random.randint(1,48),
"result": "benign"}))
#         file_body = self.request.files['filefieldname'][0]['body']
#         img = Image.open(BytesIO(file_body))
#         img.save("current.jpg")
#         f = open("current.jpg", "rb")
#         images = [{'content': base64.b64encode(f.read()).decode('UTF-
8')}]
#         x = requests.post("https://skindoc-
10ef5.appspot.com/melanoma/predict", json=images)
#         f.close()
#         self.write(json.dumps(x.json()[0]))


class AdminFeature(tornado.web.RequestHandler):
    def get(self):
        self.write(json.dumps({'message': self.settings['logged_in']}))


if __name__ == "__main__":
    options.parse_command_line()
    client =
motor_tornado.MotorClient("mongodb://samyak:samyak1@ds247944.mlab.com:4
7944/tornado")
    app = tornado.web.Application(
        handlers=[
            (r"/", AuthHandler),
            (r"/login", AuthHandler),
            (r"/Signup", SignUpHandler),
            (r"/signup", SignUpHandler),
            (r"/logout", LogoutHandler),
            (r"/appoint", AppointmentHandler),
            (r"/admin", AdminFeature),
            (r"/new", NewTokenGenerator),
            (r"/oauth",OauthLogin),
            (r"/resolve", ResolveAppointment),
            (r"/mlpredict", MLHandler)
        ],
        default_handler_class = my404handler,
        debug = True,
        cookie_secret = "abc",
        login_url = "/login",
        db_client = client,
        ap_details = dict(),
        client_id = "def",
        template_path=os.path.join(os.path.dirname(__file__),
"template"),
        static_path=os.path.join(os.path.dirname(__file__), "static"),
    )
    http_server = tornado.httpserver.HTTPServer(app)
    http_server.listen(os.environ.get("PORT", options.port))
    tornado.ioloop.IOLoop.instance().start()
```

Classification Coding for Skin Cancer

```
from tornado.gen import coroutine
from random import choice
from datetime import datetime
class users:
    def __init__(self, email, user, name):
        self.email = email
        self.user = user
        self.name = name

class patient(users):
    @staticmethod
    @coroutine
    def make_appointment(db, user):
        resp = db.doctor.find({"$where": "this.plist.length<3"})
        listOfDoc = []
        while (yield resp.fetch_next):
            ele = resp.next_object()
            if 'qualifications' not in ele:
                ele['qualifications'] = None
            if 'description' not in ele:
                ele['description'] = None
            if 'name' not in ele:
                ele['name'] = ele['user']
            listOfDoc.append(
                dict(email=ele['email'], user=ele['user'],
plist=ele['plist'], availability=ele['availability'],
                     qualifications=ele['qualifications'],
description=ele['description'], name=ele['name']))
        print(user)
        if len(user['ap_details']) >= 3:
            print(user)
            return {'message': 'Maximum Appointments reached.',
'status_code': 400}
        if not listOfDoc:
            return {'message': 'Doctors Not Available.',
                    'status_code': 400}
        dbdoc = choice(listOfDoc)
        plist = [i['user'] for i in dbdoc['plist']]
        print(plist)
        while user['user'] in plist:
            listOfDoc.remove(dbdoc)
            if not listOfDoc:
                return {'message': 'Doctors Not Available.',
                        'status_code': 400}
            dbdoc = choice(listOfDoc)
            if not dbdoc['availability']:
                listOfDoc.remove(dbdoc)
                if not listOfDoc:
                    return {'message': 'Doctors Not Available.',
                            'status_code': 400}
                dbdoc = choice(listOfDoc)
            plist = [i['user'] for i in dbdoc['plist']]
        print(dbdoc)
        if 'name' not in user:
            user['name'] = user['user']

        user_details = dict(user=user['user'], name=user['name'],
description=user['description'],
                            datetime=datetime.now().strftime("%d-%m-%Y
%H:%M"))
        doctor_details = dict(user=dbdoc['user'], name=dbdoc['name'],
qualifications=dbdoc['qualifications'],
                              datetime=datetime.now().strftime("%d-%m-
%Y %H:%M"),description=dbdoc['description'])
        dbdoc['plist'].append(user_details)
        plist = dbdoc['plist']
        ap_list = user['ap_details']
        ap_list.append(doctor_details)
        db.patient.update({'user': user['user']}, {'$set':
{'ap_details': ap_list}}, upsert=False)
        db.doctor.update({'user': dbdoc['user']}, {'$set': {'plist':
plist}}, upsert=False)
        return {'status_code': 200, 'message': 'Updated Successfully.'}
```

IPR Justification

“Intellectual property (IP)” refers to inventions, devices, new varieties of designs and other properties that are produced through “mental or creative labour” by human beings, and the law regulating intellectual property is “highly politicised.” “Intellectual property rights (IPR)” is a catch-all term used to describe the legal status and protection that allows people to own intellectual properties – the intangible products of their creativity and innovation imbedded in physical objects – in the form that they own physical properties. According to the official interpretation of World Intellectual Property Organisation (WIPO), IPR comprises those legal rights, by which the products of intellectual activity over a range of endeavours are defined. For the purposes of the TRIPs Agreement, IPR refers to copyright and related rights, trademarks, geographical indications, industrial designs, patents, integrated circuit layout-designs, protection of undisclosed information and anti-competitive practices in contractual licenses.

In our project, we have new features like:-

- Previously the models used in the field of Skin Cancer are old and basically based on small CNN and deep learning models and which have less accuracy and identification of the problem. In our case we have used the fundamentals of CNN and introduced new models of CNN like RainForest Classifier, HaarCascading etc.
- The Project also includes the GANs (Generative Adversarial Network). This helps in making more images for the classification of the data. This helped in project not only

to detect one type of cancer but also the different other cases formed and the way to handle them.

- We also formed a Chat between Patient and Doctor which will be able to appoint time for the patient to meet the respective doctor. This will reduced the rush in the hospital and the doctors will be extra benefitted by this app as they may get chats from other region or district nearby.
- The dataset used is an authenticated one and thus will provide true results. This thing was clarified by the Doctor we met at Chetinadu Hospital in VIT Vellore and the Bala Skin Clinic near PVR, Vellore.
- The marketability of the product is Hugh and the access to the product is the need for the call of the hour. This is important as the India as a mass becomes negligence in terms of the checking for the Skin rushes and the People suffer. This is a step in that field. The recent apps that are present in the PlayStore or AppStore are out dated and cannot detect as they don’t have much of the evolution in them. Since they are based on static data.

All this represents the uniqueness of our project and the understanding why an IPR is needed for the same.