Python Code for – Detection, Page Routes

// app.py

```
from flask import Flask, render_template, Response, jsonify, request
import cv2
import numpy as np

# for accessing session storage
from flask import session, redirect

# cloudant imports
from cloudant.client import Cloudant

# sub-imports
# from object_detection import Detect

# connecting client with cloudant db
client = Cloudant.iam('5e67dcf0-6dd2-49ef-ba49-548e2376d5fa-bluemix',
                      'T0BBzOvBQK6JyezcCq1xelsmRiuVe-AQ1PwdufX_3XCL',
                      connect = True)

db = client.create_database('veye_users')

app=Flask(__name__)

class Detect:

    def __init__(self, video_source,
                      classes,
                      config,
                      weights,
                      frame_title,
                      wait_key,
                      threshold,
                      suppression_threshold,
                      yolo_image_size):


                      self.video_source = video_source
```

```
                self.classes = classes
                self.config = config
                self. weights = weights
                self.frame_title = frame_title
                self.wait_key = wait_key
                self.threshold = threshold
                self.suppression_threshold = suppression_threshold
                self.yolo_image_size = yolo_image_size
                self.detect_count = 0

    def find_objects(self, model_outputs, YOLO_IMAGE_SIZE, THRESHOLD,
SUPPRESSION_THRESHOLD):
        bounding_box_locations = []
        class_ids = []
        confidence_values = []

        for output in model_outputs:
            for prediction in output:
                class_probabilities = prediction[5:]
                class_id = np.argmax(class_probabilities)
                confidence = class_probabilities[class_id]

                if confidence > THRESHOLD:
                    w, h = int(prediction[2] * YOLO_IMAGE_SIZE), int(prediction[3] *
YOLO_IMAGE_SIZE)
                    # the center of the bounding box (we should transform these
values)
                    x, y = int(prediction[0] * YOLO_IMAGE_SIZE - w / 2),
int(prediction[1] * YOLO_IMAGE_SIZE - h / 2)
                    bounding_box_locations.append([x, y, w, h])
                    class_ids.append(class_id)
                    confidence_values.append(float(confidence))

        box_indexes_to_keep = cv2.dnn.NMSBoxes(bounding_box_locations,
confidence_values, THRESHOLD, SUPPRESSION_THRESHOLD)

        return box_indexes_to_keep, bounding_box_locations, class_ids,
confidence_values
```

```
   def mark_detected_objects(self, img, bounding_box_ids,
all_bounding_boxes, class_ids, confidence_values, width_ratio,
                    height_ratio):
       for index in bounding_box_ids:
          bounding_box = all_bounding_boxes[index]
          x, y, w, h = int(bounding_box[0]), int(bounding_box[1]),
int(bounding_box[2]), int(bounding_box[3])

          # we have to transform the locations and coordinates because the
image is resized

          x = int(x * width_ratio)
          y = int(y * height_ratio)
          w = int(w * width_ratio)
          h = int(h * height_ratio)

          # OpenCV deals with BGR blue green red (255,0,0) then it is the blue
color
          # we are not going to detect every objects just PERSON and CAR
          # if class_ids[index] == 2:
          #     cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
          #     class_with_confidence = 'CAR' + str(int(confidence_values[index] *
100)) + '%'
          #     cv2.putText(img, class_with_confidence, (x, y-10),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.5, (255, 0, 0), 1)

          if class_ids[index] == 0:

             self.detect_count += 1

             cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
             class_with_confidence = f'drowning' +
str(int(confidence_values[index] * 100)) + '%'
             cv2.putText(img, class_with_confidence, (x, y-10),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.5, (255, 0, 0), 1)

# find_objects
# mark_detected_objects
```

```
  def generate_frames(self):
    capture = cv2.VideoCapture(self.video_source)

    neural_network = cv2.dnn.readNetFromDarknet(self.config, self.weights)

    neural_network.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
    neural_network.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

    YOLO_IMAGE_SIZE = self.yolo_image_size

    while True:
      frame_grabbed, frame = capture.read()

      if not frame_grabbed:
        break
      else:

        original_width, original_height = frame.shape[1], frame.shape[0]

        # the image into a BLOB [0-1] RGB - BGR
        blob = cv2.dnn.blobFromImage(frame, 1 / 255, (YOLO_IMAGE_SIZE,
YOLO_IMAGE_SIZE), True, crop=False)
        neural_network.setInput(blob)

        layer_names = neural_network.getLayerNames()
        # YOLO network has 3 output layer - note: these indexes are starting
with 1
        output_names = [layer_names[index - 1] for index in
neural_network.getUnconnectedOutLayers()]

        self.detect_count = 0

        outputs = neural_network.forward(output_names)
        predicted_objects, bbox_locations, class_label_ids, conf_values =
self.find_objects(outputs,
                                                    self.yolo_image_size,
                                                    self.threshold,

self.suppression_threshold)
```

```
            self.mark_detected_objects(frame, predicted_objects,
bbox_locations, class_label_ids, conf_values,
                        original_width / YOLO_IMAGE_SIZE, original_height /
YOLO_IMAGE_SIZE)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()



        yield (b'--frame\r\n'
        b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

# global declaration
source = Detect(video_source = './media/swimming_pool1.mp4',
                classes = ['drowning'],
                config = './config/yolov3_testing.cfg',
                weights = './weights/yolov3_training_3000.weights',
                frame_title = 'YOLO V3 Object Detection',
                wait_key = 10,
                threshold = 0.5,
                suppression_threshold = 0.4,
                yolo_image_size = 320)

@app.route('/counter', methods=['POST'])
def counter():
   return jsonify('', render_template('counter.html', dyn_var =
source.detect_count))

@app.route('/video')
def video():

   frame = source.generate_frames()

   return Response(frame,
   mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/detection', methods=["GET", "POST"])
def detection():
```

```
    if (session.get("user_token")):
        return render_template('detection.html', dyn_var = source.detect_count)
    return render_template("login_redirect.html", dyn_message = "You need to
login first!")


# login & registration
@app.route('/validate_login', methods=["GET", "POST"])
def validate_login():
    if request.method == "POST":
        email = request.form.get("user_login_email")
        password = request.form.get("user_login_password")

        session["login_username"] = email
        session["login_password"] = password

        test_login = {
            '_id': email,
            'pword': password
        }

        # test_login = {
        #     '_id': 'veye_admin',
        #     'pword': 'veye_admin'
        # }

        if (test_login['_id'] and test_login['pword']) in db:
            session["user_token"] = db[test_login['_id']]['_rev']

            print(f"username: {session.get('login_username')}; password:
{session.get('login_password')}")

            return render_template('login_modules/login_success.html',
            dyn_message = "You're in!")

    return render_template('/login.html', dyn_message = "check your u/name
or p/word")

@app.route('/logout')
```

```
def logout():
    if session.get("login_username"): session.pop("login_username")
    if session.get("login_password"): session.pop("login_password")
    if session.get("user_taken"): session.pop("user_token")
    return redirect("/")

@app.route('/about')
def about():
    return render_template("about.html")

@app.route('/register_intro', methods=["GET", "POST"])
def register_intro():
    return render_template('register_user/register_intro.html')

@app.route('/register_name', methods=["POST", "GET"])
def register_name():
    # if request.method == "POST":
    # register_user_name = request.form.get("user_name")

    # session["register_user_name"] = register_user_name
    # print(f"name set: {session['register_user_name']}")

    return render_template('register_user/register_name.html')

@app.route('/register_email', methods=["GET", "POST"])
def register_email():
    if request.method == "POST":

    # retrieve user_name from name page
        register_user_name = request.form.get("user_name")

        session["register_user_name"] = register_user_name
        print(f"name set: {session['register_user_name']}")



    return render_template('register_user/register_email.html')

@app.route('/register_password', methods=["GET", "POST"])
def register_password():
```

```
    if request.method == "POST":

        # retrieve user_email from email page
        register_user_email = request.form.get("user_email")

        session["register_user_email"] = register_user_email
        print(f"email set: {session['register_user_email']}")

    return render_template('register_user/register_password.html')

@app.route('/register_phoneNumber', methods=["GET", "POST"])
def register_phoneNumber():
    if request.method == "POST":

        # retrieve user_pass from password page
        register_user_pword = request.form.get("user_pass")

        session["register_user_pword"] = register_user_pword
        print(f"pword set: {session['register_user_pword']}")

    return render_template('register_user/register_phoneNumber.html')

@app.route('/register_outro', methods=["GET", "POST"])
def register_outro():
    if request.method == "POST":
    # retrieve user_phone from phoneNumber page
        register_user_phoneNumber = request.form.get("user_phone")

        session["register_user_phone"] = register_user_phoneNumber
        print(f"phone number set: {session['register_user_phone']}")



    register_new_document = {
        '_id': str(session.get("register_user_email")),
        'name': str(session.get("register_user_name")),
        'pword': str(session.get("register_user_pword")),
        'phoneNumber': str(session.get("register_user_phone"))
    }
```

```
    new_document = db.create_document(register_new_document)

    if new_document.exists():
        print(register_new_document)
        return render_template('register_user/register_outro.html',
        dyn_message = "You're in!")



    return render_template('register_user/register_outro.html',
    dyn_message = "Oops! Seems like there was a problem while registering
you in. Contact Administrator.")

@app.route('/')
def login():
    return render_template('login.html', dyn_message = "")

if __name__ == "__main__":
    app.config["SESSION_PERMANENT"] = False
    app.config["SESSION_TYPE"] = "filesystem"
    app.secret_key = "veye"
    app.run(debug=True)
```