Grant Novota

HTML GUI Development Project

Assumptions/Notes:

* Using Python3.8 on an Ubuntu machine. Viewed in firefox.
* Database is sqlite3 and interactions are through a python script using sqlalchemy.
* Generated graphs of temperature and humidity that update on a set time interval and show upper and lower bounds for alarms.
* Alarms are pop-ups.
* Humidity and temperature metrics are displayed in a table format in the UI.
* Made a change to show the total number of humidity/temperature samples and use the entire dataset for max, min, and average values.
* Encountered issues with handling static files with tornado, so decided to not include graphs.

Required Libraries:

* sqlalchemy
* tornado
* numpy

Code:

The project code is made up of 3 parts: the pseudo sensor code “pseudoSensor.py”, the database interaction code “db.py”, and the main code “main.py”

pseudoSensor.py:

import random

# pseudo temp and humidity sensor

class PseudoSensor:

    h\_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10, 10]

    t\_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]

    h\_range\_index = 4

    t\_range\_index = 5

    humVal = 0

    tempVal = 0

    def \_\_init\_\_(self):

        self.humVal = self.h\_range[self.h\_range\_index]

        self.tempVal = self.t\_range[self.t\_range\_index]

    def generate\_values(self):

        self.humVal = self.h\_range[self.h\_range\_index] + random.uniform(0, 10)

        self.tempVal = self.t\_range[self.t\_range\_index] + random.uniform(0, 10)

        self.h\_range\_index += 1

        if self.h\_range\_index > len(self.h\_range) - 1:

            self.h\_range\_index = 0

        self.t\_range\_index += 1

        if self.t\_range\_index > len(self.t\_range) - 1:

            self.t\_range\_index = 0

        return self.humVal, self.tempVal

db.py:

# database functions

from datetime import datetime, timezone

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy import Column, ForeignKey, Integer, String, Float, Boolean, DateTime

from sqlalchemy import Index

from sqlalchemy.orm import relationship, backref, sessionmaker

from sqlalchemy import create\_engine, select

Base = declarative\_base()

# Tables

class Temperature(Base):

    \_\_tablename\_\_ = 'temperature'

    id = Column(Integer, primary\_key = True, autoincrement=True)

    # fahrenheit

    value\_f = Column(Float)

    # celsius

    value\_c = Column(Float)

    time = Column(DateTime)

class Humidity(Base):

    \_\_tablename\_\_ = 'humidity'

    id = Column(Integer, primary\_key = True, autoincrement=True)

    value = Column(Float)

    time = Column(DateTime)

# general db functions

def create(database):

    # an engine that the session will use for resources

    engine = create\_engine(database)

    # create a configured session class

    Session = sessionmaker(bind=engine)

    # create a session

    session = Session()

    return engine, session

def result\_dict(r):

    return dict(zip(r.keys(), r))

def result\_dicts(rs):

    return list(map(result\_dict, rs))

def database\_dump(session):

    Database = [Temperature, Humidity]

    for table in Database:

        stmt = select('\*').select\_from(table)

        result = session.execute(stmt).fetchall()

        print(result\_dicts(result))

    return

def create\_tables(engine):

    Base.metadata.create\_all(engine)

    return

def init\_session():

    engine, session = create("sqlite:///db.sqlite3")

    create\_tables(engine)

    return session

def close(conn):

    conn.close()

    return

def delete\_obj(session, obj):

    session.delete(obj)

    session.commit()

    return

# add rows to tables

def add\_temp(session, value\_f, value\_c, time):

    temp = Temperature(value\_f=value\_f, value\_c=value\_c, time=time)

    session.add(temp)

    session.commit()

    return

def add\_humidity(session, value, time):

    humidity = Humidity(value=value, time=time)

    session.add(humidity)

    session.commit()

    return

def get\_all\_temps(session, type):

    temp\_list = []

    temp\_times = []

    temps = session.query(Temperature).all()

    for temp in temps:

        if type == "f":

            temp\_list.append(temp.value\_f)

        else:

            temp\_list.append(temp.value\_c)

        temp\_times.append(temp.time)

    return temp\_list, temp\_times

def get\_all\_humids(session):

    humid\_list = []

    humid\_times = []

    humids = session. query(Humidity).all()

    for humid in humids:

        humid\_list.append(humid.value)

        humid\_times.append(humid.time)

    return humid\_list, humid\_times

def get\_latest\_temp(session):

    return session.query(Temperature).order\_by(Temperature.id.desc()).first()

def get\_latest\_humidity(session):

    return session.query(Humidity).order\_by(Humidity.id.desc()).first()

main.py:

# main code

import sys

import time

import numpy as np

from datetime import datetime

import tornado.ioloop

import tornado.web

import os

import json

# my libraries

import db

from psuedoSensor import PseudoSensor

# init database

session = db.init\_session()

tornadoPort = 8888

cwd = os.getcwd() # used by static file server

current\_temp = 0.0

current\_humidity = 0.0

# alarm limits

temp\_min\_limit = 30.0

temp\_max\_limit = 80.0

humid\_min\_limit = 30.0

humid\_max\_limit = 70.0

# alarms

temp\_min\_alarm = False

temp\_max\_alarm = False

humid\_min\_alarm = False

humid\_max\_alarm = False

# allow cross-origin requests

class BaseHandler(tornado.web.RequestHandler):

    def set\_default\_headers(self):

        print("setting headers!!!")

        self.set\_header("Access-Control-Allow-Origin", "\*")

        self.set\_header("Access-Control-Allow-Headers", "x-requested-with")

        self.set\_header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS')

        # HEADERS!

        self.set\_header("Access-Control-Allow-Headers", "access-control-allow-origin,authorization,content-type")

    def options(self):

        # no body

        self.set\_status(204)

        self.finish()

# send the index file

class IndexHandler(BaseHandler):

    def get(self, url = '/'):

        self.render('index.html')

    def post(self, url ='/'):

        self.render('index.html')

# handle commands sent from the web browser

class CommandHandler(BaseHandler):

    #both GET and POST requests have the same responses

    def get(self, url = '/'):

        print("get")

        self.handleRequest()

    def post(self, url = '/'):

        print("post")

        self.handleRequest()

    # handle both GET and POST requests with the same function

    def handleRequest(self):

        # is op to decide what kind of command is being sent

        op = self.get\_argument('op', None)

        global temp\_min\_limit, temp\_max\_limit, humid\_min\_limit, humid\_max\_limit

        global temp\_min\_alarm, temp\_max\_alarm, humid\_min\_alarm, humid\_max\_alarm

        #received a "checkup" operation command from the browser:

        if op == "checkup":

            print("checkup called")

            #make a dictionary

            status = {"server": True }

            #turn it to JSON and send it to the browser

            self.write( json.dumps(status) )

        elif op == "sample once":

            print("sample once called")

            single\_sample()

            #make a dictionary

            global current\_temp, current\_humidity

            status = {"server": True, "current\_temp": current\_temp, "current\_humidity": current\_humidity,

                "temp\_max\_limit": temp\_max\_limit, "humid\_max\_limit": humid\_max\_limit,

                "temp\_min\_limit": temp\_min\_limit, "humid\_min\_limit": humid\_min\_limit,

                "temp\_max\_alarm": temp\_max\_alarm, "humid\_max\_alarm": humid\_max\_alarm,

                "temp\_min\_alarm": temp\_min\_alarm, "humid\_min\_alarm": humid\_min\_alarm}

            #turn it to JSON and send it to the browser

            self.write( json.dumps(status) )

        elif op == "sample multi":

            print("multi sample called")

            max = 10

            print("take 10 samples:")

            for i in range(max):

                print('sample', i+1)

                single\_sample()

                time.sleep(1)

            global current\_temp, current\_humidity

            status = {"server": True, "current\_temp": current\_temp, "current\_humidity": current\_humidity,

                "temp\_max\_limit": temp\_max\_limit, "humid\_max\_limit": humid\_max\_limit,

                "temp\_min\_limit": temp\_min\_limit, "humid\_min\_limit": humid\_min\_limit,

                "temp\_max\_alarm": temp\_max\_alarm, "humid\_max\_alarm": humid\_max\_alarm,

                "temp\_min\_alarm": temp\_min\_alarm, "humid\_min\_alarm": humid\_min\_alarm}

            #turn it to JSON and send it to the browser

            self.write( json.dumps(status) )

        elif op == "calc metrics":

            print("calc metrics called")

            metrics = calc\_metrics()

            metrics["server"] = True

            status = metrics

            #turn it to JSON and send it to the browser

            self.write( json.dumps(status) )

        elif op == "set max temp":

            value = self.get\_argument('value', None)

            temp\_max\_limit = float(value)

            print("max temp value:", value)

        elif op == "set max humidity":

            value = self.get\_argument('value', None)

            humid\_max\_limit = float(value)

            print("max humidity value:", value)

        elif op == "set min temp":

            value = self.get\_argument('value', None)

            temp\_min\_limit = float(value)

            print("min temp value:", value)

        elif op == "set min humidity":

            value = self.get\_argument('value', None)

            humid\_min\_limit = float(value)

            print("min humidity value:", value)

        elif op == "create error":

            status = {}

            self.write( json.dumps(status) )

        elif op == "stop server":

            stop\_tornado()

        #operation was not one of the ones that we know how to handle

        else:

            print(op)

            print(self.request)

            raise tornado.web.HTTPError(404, "Missing argument 'op' or not recognized")

    def send\_update(self):

        global current\_temp, current\_humidity

        status = {"current\_temp": current\_humidity, "current\_humidity": current\_humidity }

        self.write( json.dumps(status) )

# adds event handlers for commands and file requests

application = tornado.web.Application([

    #all commands are sent to http://\*:port/com

    #each command is differentiated by the "op" (operation) JSON parameter

    (r"/(com.\*)", CommandHandler ),

    (r"/", IndexHandler),

    (r"/(index\.html)", tornado.web.StaticFileHandler,{"path": cwd}),

    (r"/(.\*\.png)", tornado.web.StaticFileHandler,{"path": cwd }),

    (r"/(.\*\.jpg)", tornado.web.StaticFileHandler,{"path": cwd }),

    (r"/(.\*\.js)", tornado.web.StaticFileHandler,{"path": cwd }),

    (r"/(.\*\.css)", tornado.web.StaticFileHandler,{"path": cwd }),

])

# END OF WEB APP FUNCTIONS

# get sample of data from pseudo sensor

def sample\_data():

    ps = PseudoSensor()

    h,temp\_f = ps.generate\_values()

    temp\_c = (temp\_f - 32) \* 5.0/9.0

    now = datetime.now()

    db.add\_temp(session, temp\_f, temp\_c, now)

    db.add\_humidity(session, h, now)

    # check if we hit an alarm

    global temp\_min\_limit, temp\_max\_limit, humid\_min\_limit, humid\_max\_limit

    global temp\_min\_alarm, temp\_max\_alarm, humid\_min\_alarm, humid\_max\_alarm

    # reset alarms

    temp\_min\_alarm = False

    temp\_max\_alarm = False

    humid\_min\_alarm = False

    humid\_max\_alarm = False

    # set alarms

    if temp\_f > temp\_max\_limit:

        temp\_max\_alarm = True

    elif temp\_f < temp\_min\_limit:

        temp\_min\_alarm = True

    elif h > humid\_max\_limit:

        humid\_max\_alarm = True

    elif h < humid\_min\_limit:

        humid\_min\_alarm = True

    return h, temp\_f

def single\_sample():

    h,t = sample\_data()

    global current\_temp, current\_humidity

    current\_temp = t

    current\_humidity = h

    print('sample', 'temp:', t, 'humidity:', h)

    return

def calc\_metrics():

        temp\_list, temp\_times = db.get\_all\_temps(session, "f")

        humid\_list, humid\_times = db.get\_all\_humids(session)

        metrics = {}

        # set total samples

        metrics["total\_samples"] = str(len(temp\_list))

        # min temp

        metrics["min\_temp"] = str(min(temp\_list))

        # min humidity

        metrics["min\_humidity"] = str(min(humid\_list))

        # max temp

        metrics["max\_temp"] = str(max(temp\_list))

        # max humidity

        metrics["max\_humidity"] = str(max(humid\_list))

        # avg temp

        metrics["avg\_temp"] = str(sum(temp\_list)/len(temp\_list))

        # avg humidity

        metrics["avg\_humidity"] = str(sum(humid\_list)/len(humid\_list))

        return metrics

def start\_tornado():

    application.listen(tornadoPort)

    tornado.ioloop.IOLoop.instance().start()

def stop\_tornado():

    tornado.ioloop.IOLoop.instance().stop()

if \_\_name\_\_ == "\_\_main\_\_":

    #start tornado

    print("Starting server on port number %i..." % tornadoPort )

    print("Open at http://127.0.0.1:%i/index.html" % tornadoPort )

    start\_tornado()

Screenshots:

1. The HTML UI at start up

Graphical user interface, text, application, email

Description automatically generated

1. Error conditions

Error in code:

Graphical user interface, application

Description automatically generated

Disconnected from server:

Graphical user interface, text, application, email

Description automatically generated

1. The UI after its first single data point reading

Graphical user interface, text, application, email

Description automatically generated

1. The UI after it has calculated a 10 point average

Graphical user interface, text, application, email

Description automatically generated

1. The UI after it has seen either a temperature or humidity alarm

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated