**Irusha’s Code - Reviewed by Dominik, Derek, and Amr**

#!/usr/bin/env python

from \_\_future\_\_ import print\_function  
import os  
import sys  
import random  
  
  
import socket  
import pickle  
import time  
from threading import Thread  
import json, socket  
import sqlite3  
  
  
#Google Email API  
import httplib2  
  
from apiclient import discovery  
import oauth2client  
  
  
  
#Custom files  
import UDPFunc  
import databaseFunc as dbFunc  
import googleEmailApiFunc as emailAPI  
import serverModl  
  
  
import sched, time  
  
DEF\_HALF\_HOUR\_IN\_SECONDS = 10 #1800  
DEF\_1\_DAY\_IN\_SECONDS = 86400  
DEF\_2\_DAYS\_IN\_SECONDS = 172800  
  
  
  
  
  
  
class serverController():  
  
 def \_\_init\_\_(self, dataBasePath, IP, PORT,servModel):  
  
 self.DEF\_DB\_PATH = dataBasePath #'lifeBandDB.db'  
 self.DEF\_IP = IP#'172.17.148.20'  
 self.DEF\_PORT = PORT#5005  
 #self.DEF\_PORT\_SEND = PORT\_Send  
 self.model = servModel  
  
  
  
  
  
 def networkHandler(self,conn,receivedData,(receivedIP,receivedPORT)):   
 """  
 Function:   
 Serve the requests that is received by Phone or wearable  
  
 Input arguments:  
 args[0] : data that is contained in UDP packets  
 args[1] : Address from the connection  
 """  
  
 #database = sqlite3.connect(DEF\_DB\_PATH)  
  
 dataDecoded = json.loads(receivedData)  
   
 if dataDecoded['id'] == "phone":  
 print (str(time.ctime())+"Phone data Received from "+str(receivedIP))  
  
 if dataDecoded['command'] == 'getLatestData':  
 print ('\t'+"Sending latest Data")  
 conn.sendto(json.dumps(self.model.getLatestDataFromDB()), (receivedIP,self.DEF\_PORT))  
  
  
 elif dataDecoded['command'] == 'getPulseDataSet':  
 print ('\t'+"Sending past Data")  
  
  
 elif dataDecoded['command'] == 'addEmergencyContact':  
 print ('\t'+"Adding Emergency Data")  
 self.model.emergContactChangeToDB('add','emergList',dataDecoded['data'])  
  
  
 elif dataDecoded['command'] == 'remEmergencyContact':  
 print ('\t'+"Removing Emergency Data")  
 self.model.emergContactChangeToDB('rem','emergList',dataDecoded['data'])  
  
  
 elif dataDecoded['id'] == "wearable":  
 print (str(time.ctime())+"Wearable data Received from "+str(receivedIP))  
  
 if dataDecoded['command'] == 'addPulseData':  
 print ('\t'+"Adding pulse data to database")  
 self.model.addSensorDataToDB('pulse',dataDecoded['data'])  
  
  
 elif dataDecoded['command'] == 'addRespData':  
 print ('\t'+"Adding respiratory data to database")  
 self.model.addSensorDataToDB('resp',dataDecoded['data'])  
  
   
 elif dataDecoded['command'] == 'addAccelData':  
 print ('\t'+"Adding accelerometer data to database")  
 self.model.addSensorDataToDB('accell',dataDecoded['data'])  
  
  
 elif dataDecoded['command'] == 'truePositiveAlarm':  
 print ('\t'+"Adding True Positive Alarm to database")  
 self.model.addAlarmToDB('TRUE')  
  
  
 elif dataDecoded['command'] == 'falsePositiveAlarm':  
 print ('\t'+"Adding False Positive Alarm to database")  
 self.model.addAlarmToDB('FALSE')  
  
  
  
   
  
  
  
  
 def emailHandler(self):  
 """  
 Function:   
 Creates a connection with the Google Email API and sends a message  
 Upon alert  
   
 Input arguments:  
 None  
  
 Output variables:  
 None  
 """  
  
 print("Email Sender started")  
 credentials = emailAPI.get\_credentials()  
 http = credentials.authorize(httplib2.Http())  
 service = discovery.build('gmail', 'v1', http=http)  
  
 message = emailAPI.CreateMessage('LifeBandCenter@gmail.com', 'irusha.dilshan@gmail.com', 'Test123', 'Hey!')  
 emailAPI.SendMessage(service, 'me', message)  
  
   
  
 def runServer(self):  
 server = UDPFunc.createUDPSocket(self.DEF\_IP,self.DEF\_PORT)  
  
 #timerThread = Thread(target = timerSched, args = [])  
 #timerThread.start()  
   
 #thread.start\_new\_thread( maintainDatabaseSize,(None,None))   
 try:  
 while True:  
 #print("Email Thread Creation")  
 #thread = Thread(target = emailHandler, args = [])  
 #thread.start()  
 #print("Email Thread Creation")  
 #Accept each communication  
 data, addr = UDPFunc.recvUDP(server)  
 #Create a new thread for each connection that is made  
 thread = Thread(target = self.networkHandler, args = (server,data,addr))  
 thread.start()  
 except (KeyboardInterrupt, SystemExit):  
 closeTCP(conn)   
 server.close()

**SYSC 3010 - Group B - Code Review Team Checklist**

✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓

* Functionality
  + Overall functionality works according to the project specifications
  + Input and Output network communications conform to established data protocol
  + Code straightforward to understand
  1. N/A Sensors working in the right way
  + No sections of code incomplete
* Comments
  + X Meta data at the top (Name, Date Modified)
  + Comments are comprehensible and add something to the maintainability of the code
  + Comments are neither too numerous nor verbose
  + Comments are in the right place and they are useful
* Code smells
  + Tabs and brackets are consistent
  + Code as modular as possible
  + Repetitive code has been factored out
  + Command classes have been designed to undertake one task only
  + The code does not use unjustifiable static methods/blocks
  + Loops have a set length and correct termination conditions
  + Any unusual behavior or edge-case handling described
  + No hardcoded or Magic numbers present within reason
* Performance
  + No unnecessary loops
  + No possible replacement of recursive functions with sequential functions
  + the code was designed to perform as fast as possible
* Scope
  + No possible replacement of global variables to function variables
  + Variable types have been generalized where possible
  + N/A Down casting used properly
* Unit Tests
  + N/A Unit tests are present and correct
  + The code is unit testable
  + Testing the part that is implemented and checked for right performance
* Error/Exception handling
  + Common errors have been checked for
  + N/A No zombie threads running
  + N/A Any security concerns have been addressed
  + data goes through many filters that makes sure the data does not contain errors
* Logging/Debugging Info
  + Logging used appropriately (proper logging level and details)
  + Are all data inputs checked (for the correct type, length, format, and range) and encoded?
* Code Library Usage
  + Frameworks have been used appropriately
  + Can any of the code be replaced with library functions?
  + the code was designed using the libraries and there is no code to be replaced