**the file: in\_server.py**# Avinoam Troen  
# in\_server  
from utils import \*  
from flask import Flask, request  
from mysql\_stuff.input\_data\_to\_mysql import insert\_record\_oneScout  
import mysql.connector  
import servers.mysql\_stuff.mysql\_utils as mysql\_utils  
  
  
# page director - gets the correct function based on url and is in charge of threading and directing everything  
app = Flask(\_\_name\_\_)  
  
  
# the main function  
@app.route(PATH\_FOR\_INPUT, methods=['POST'])  
def input\_one\_scout():  
 """  
 This function is called when an http request is sent to the server at the url of app.route  
 For the function to succeed the request needs to be of type post with a json containing all the relevant fields  
 :return:  
 The function returns 200 status code for success and 400 or 500 status code (with a short message) for failure.  
 """  
 try:  
 json\_data = request.json  
  
 print('\ninput\_one\_scout: initiating\n')  
 print('input\_one\_scout: about to print oneScout Json\n')  
 for key in json\_data:  
 print(key, json\_data[key])  
 print('\ninput\_one\_scout: done\n')  
  
 val = (json\_data['compName'], json\_data['matchType'], json\_data['matchNumber'], json\_data['teamNumber'],  
 json\_data['scouterName'], json\_data['whenCaptured'], json\_data['scouterTeamNumber'],  
 json\_data['ballsInUpperAuto'],  
 json\_data['ballsInLowerAuto'], json\_data['ballsMissedAuto'], json\_data['passedLine'],  
 json\_data['ballsHumanShotAuto'], json\_data['ballsHumanScoredAuto'], json\_data['autoMalfunction'],  
 json\_data['autoFreeText'], json\_data['ballsInUpperTele'], json\_data['ballsInLowerTele'],  
 json\_data['ballsMissedTele'], json\_data['levelClimbed'], json\_data['climbSuccessful'],  
 json\_data['climbTime'], json\_data['defensiveDefenseLevel'], json\_data['offensiveDefenseLevel'],  
 json\_data['wasDefendedLevel'], json\_data['goodTeamMateLevel'], json\_data['wasBroken'],  
 json\_data['freeText'], json\_data['generalImpression'], json\_data['robotNoFunction'],  
 json\_data['systemNoFunction'])  
  
 # print out the values we will put into the DB for debugging/log purposes  
 print('input\_one\_scout: about to input the following into the DB')  
 print(val)  
  
 # open a connection to the DB  
 the\_db = mysql.connector.connect(  
 host=mysql\_utils.HOST,  
 user=mysql\_utils.USER,  
 password=mysql\_utils.PASSWORD,  
 database=mysql\_utils.DB\_NAME  
 )  
 # insert oneScout to DB  
 insert\_record\_oneScout(the\_db, val)  
  
 # if we got here the input was successful - another log message  
 print('input\_one\_scout: successful input to DB')  
  
 # if we got to here we were successful and can return 200 - a success status code  
 return '', 200  
 # this is useful in case the request json was missing fields (400, bad request)  
 except KeyError:  
 # log error  
 print('input\_one\_scout: 400')  
 return 'KeyError', 400  
 # the code shouldn't get here, but if it does at least we'll get back the type of error (500, internal server error)  
 except Exception as error:  
 # log error  
 print('input\_one\_scout: 500')  
 return type(error).\_\_name\_\_, 500  
  
  
# runs from here  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host=IPADDR, port=INPUT\_SERVER\_PORT, debug=True)

**the file: out\_data.py**from flask import jsonify  
  
  
# all functions in this module which return data from sql will return a  
# jsonified object which can be sent  
  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*remove defaults\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# generic helper function to remove all of a value from a list  
def remove\_all\_occurrences(list\_obj, value):  
 """  
 generic helper function to remove unwanted default values  
 :param list\_obj:  
 :param value:  
 :return:  
 """  
 while value in list\_obj:  
 list\_obj.remove(value)  
  
  
def cut\_dict(data\_dict, num\_of\_rounds):  
 """  
 takes a data dict of scouts (each value in dict is a list of values for the particular field) and cuts the  
 dict to have only the latest results (latest num\_of\_rounds results)  
 :param data\_dict: data dict of scouts  
 :param num\_of\_rounds: int - cuts by this  
 :return: None  
 """  
 for key in data\_dict:  
 if len(data\_dict[key]) > num\_of\_rounds:  
 data\_dict[key] = data\_dict[key][-num\_of\_rounds:]  
  
  
# remove default unuseful data like -1  
def remove\_unused\_values(data\_dict):  
 """  
 clears a dict with info from scouts of deafault - garbage data  
 :param data\_dict: dict of scouts in which each value is a list  
 :return: None  
 """  
 # Defaults  
 DEF\_STR = ""  
 DEF\_NUM\_CHOICE = -1  
 DEF\_ZERO = 0  
 DEF\_TIME = "1989-03-20 00:00:00"  
  
 # remove irrelevant data  
 remove\_all\_occurrences(data\_dict['compNameS'], DEF\_STR)  
 remove\_all\_occurrences(data\_dict['matchTypeS'], DEF\_STR)  
 remove\_all\_occurrences(data\_dict['matchNumberS'], DEF\_ZERO)  
 remove\_all\_occurrences(data\_dict['teamNumberS'], DEF\_ZERO)  
 remove\_all\_occurrences(data\_dict['scouterNameS'], DEF\_STR)  
 remove\_all\_occurrences(data\_dict['whenCapturedS'], DEF\_TIME)  
 remove\_all\_occurrences(data\_dict['scouterTeamNumberS'], DEF\_ZERO)  
 remove\_all\_occurrences(data\_dict['autoFreeTextS'], DEF\_STR)  
 remove\_all\_occurrences(data\_dict['climbTimeS'], DEF\_ZERO)  
 remove\_all\_occurrences(data\_dict['defensiveDefenseLevelS'], DEF\_NUM\_CHOICE)  
 remove\_all\_occurrences(data\_dict['offensiveDefenseLevelS'], DEF\_NUM\_CHOICE)  
 remove\_all\_occurrences(data\_dict['wasDefendedLevelS'], DEF\_NUM\_CHOICE)  
 remove\_all\_occurrences(data\_dict['goodTeamMateLevelS'], DEF\_NUM\_CHOICE)  
 remove\_all\_occurrences(data\_dict['wasBrokenS'], DEF\_NUM\_CHOICE)  
 remove\_all\_occurrences(data\_dict['freeTextS'], DEF\_STR)  
 remove\_all\_occurrences(data\_dict['generalImpressionS'], DEF\_NUM\_CHOICE)  
  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* reorder  
def get\_scores(data\_dict, num\_of\_rounds):  
 """  
 gets the scores on three metrics:  
 1) offensive score  
 2) defensive score  
 3) general score  
 :param db: the db to use  
 :param data\_dict: scouts to run on  
 :param num\_of\_rounds: num of rounds back to look  
 :return: dict of:  
 offensiveScore  
 defensiveScore  
 generalScore  
 defenseLevel  
 """  
 # cut data dict by num of rounds (if less than don't touch)  
 cut\_dict(data\_dict, num\_of\_rounds)  
 # remove garbage values  
 remove\_unused\_values(data\_dict)  
  
 # calculate the stuff  
 climbs = data\_dict['levelClimbedS']  
 climb = [climbs.count(0), climbs.count(1), climbs.count(2), climbs.count(3), climbs.count(4)]  
  
 print(climb)  
 if sum(climb) != 0:  
 taxi\_points = (sum(data\_dict['passedLineS']) \* 2) / sum(climb) # sum climb is the num of games returned  
 else:  
 taxi\_points = 0  
 auto\_points = avg(data\_dict['ballsInUpperAutoS']) \* 4 + avg(data\_dict['ballsInLowerAutoS']) \* 4  
 tele\_points = avg(data\_dict['ballsInUpperTeleS']) \* 2 + avg(data\_dict['ballsInLowerTeleS'])  
 if len(climb) >= 5 and sum(climb) != 0:  
 print('climb true..')  
 climb\_points = (climb[1] \* 4 + climb[2] \* 6 + climb[3] \* 10 + climb[4] \* 15) / sum(climb)  
 else:  
 climb\_points = 0  
  
 offensiveScore = taxi\_points + auto\_points + tele\_points + climb\_points + 4 \* (  
 avg(data\_dict['offensiveDefenseLevelS']) / 7)  
 defensiveScore = taxi\_points + min([auto\_points, 4]) + climb\_points + 10 \* (  
 avg(data\_dict['defensiveDefenseLevelS']) / 7)  
 generalScore = offensiveScore + defensiveScore  
 defenseLevel = data\_dict['defensiveDefenseLevelS']  
  
 try:  
 scores\_dict = {'teamNumber': data\_dict['teamNumberS'][0], 'offensiveScore': offensiveScore,  
 'defensiveScore': defensiveScore, 'generalScore': generalScore, 'defenseLevel': defenseLevel}  
 except IndexError:  
 scores\_dict = {'teamNumber': 0, 'offensiveScore': offensiveScore,  
 'defensiveScore': defensiveScore, 'generalScore': generalScore, 'defenseLevel': defenseLevel}  
 return scores\_dict  
  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* "Expected" next round \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
def avg(num\_list):  
 """  
 :param num\_list: a list of numbers  
 :return: the average, if the list is empty - will return 0  
 """  
 if num\_list:  
 res = (sum(num\_list) / len(num\_list))  
 return res  
 # returns 0 if list is empty  
 return 0  
  
  
# return average (of relevant data) of scouts - a json  
def get\_avg(data\_dict, num\_of\_rounds):  
 # cut data dict by num of rounds (if less than don't touch)  
 cut\_dict(data\_dict, num\_of\_rounds)  
 # remove garbage values  
 remove\_unused\_values(data\_dict)  
  
 # make calculations and update avg\_dict  
 avg\_dict = dict()  
 # not sending idS, compNames, matchTypes, matchNumbers, scouterNames, whenCaptureds, scouterTeamNumbers  
 # in the results here  
 # notice the format things are sent in as it will be crucial when unpacking  
 if data\_dict['teamNumberS']: # checks it's not an empty list  
 avg\_dict['teamNumber'] = data\_dict['teamNumberS'][0] # getting 0 cuz they are all meant to be the same  
  
 avg\_dict['ballsInUpperAuto'] = avg(data\_dict['ballsInUpperAutoS'])  
 avg\_dict['ballsInLowerAuto'] = avg(data\_dict['ballsInLowerAutoS'])  
 avg\_dict['ballsMissedAuto'] = avg(data\_dict['ballsMissedAutoS'])  
  
 avg\_dict['passedLine'] = sum(data\_dict['passedLineS'])  
  
 avg\_dict['ballsHumanShotAuto'] = sum(data\_dict['ballsHumanShotAutoS'])  
 avg\_dict['ballsHumanScoredAuto'] = sum(data\_dict['ballsHumanScoredAutoS'])  
 avg\_dict['autoMalfunction'] = sum(data\_dict['autoMalfunctionS'])  
  
 avg\_dict['autoFreeText'] = '\n'.join(data\_dict['autoFreeTextS'])  
  
 avg\_dict['ballsInUpperTele'] = avg(data\_dict['ballsInUpperTeleS'])  
 avg\_dict['ballsInLowerTele'] = avg(data\_dict['ballsInLowerTeleS'])  
 avg\_dict['ballsMissedTele'] = avg(data\_dict['ballsMissedTeleS'])  
  
 climbs = data\_dict['levelClimbedS']  
 avg\_dict['levelClimbed'] = [climbs.count(0), climbs.count(1), climbs.count(2), climbs.count(3), climbs.count(4)]  
  
 avg\_dict['climbSuccessful'] = sum(data\_dict['climbSuccessfulS'])  
 avg\_dict['climbTime'] = sum(data\_dict['climbTimeS'])  
  
 avg\_dict['defensiveDefenseLevel'] = avg(data\_dict['defensiveDefenseLevelS'])  
 avg\_dict['offensiveDefenseLevel'] = avg(data\_dict['offensiveDefenseLevelS'])  
 avg\_dict['wasDefendedLevel'] = avg(data\_dict['wasDefendedLevelS'])  
 avg\_dict['goodTeamMateLevel'] = avg(data\_dict['goodTeamMateLevelS'])  
 avg\_dict['wasBroken'] = avg(data\_dict['wasBrokenS'])  
  
 avg\_dict['freeText'] = '\n'.join(data\_dict['freeTextS'])  
  
 avg\_dict['generalImpression'] = avg(data\_dict['generalImpressionS'])  
 avg\_dict['robotNoFunction'] = sum(data\_dict['robotNoFunctionS'])  
 avg\_dict['systemNoFunction'] = sum(data\_dict['systemNoFunctionS'])  
  
 scores = get\_scores(data\_dict, num\_of\_rounds)  
 results\_dict = {'scores': scores, 'avg\_dict': avg\_dict}  
 return jsonify(results\_dict)  
  
# return min/max - a json  
  
  
# return time past adjusted score - a json  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Match expectation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# return  
  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ranked Top Teams \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Comp expectation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**the file: out\_server.py**# Avinoam Troen  
# out\_server  
from utils import \*  
from flask import Flask, request  
from mysql\_stuff.get\_things\_sql import \*  
from out\_data import \*  
import mysql.connector  
import servers.mysql\_stuff.mysql\_utils as mysql\_utils  
  
# page director - gets the correct function based on url and is in charge of threading and directing everything  
app = Flask(\_\_name\_\_)  
  
  
# get team avg  
@app.route(PATH\_FOR\_SCOUT\_TEAM\_OUTPUT + AVG, methods=['POST'])  
def output\_team\_avg():  
 """  
 This function is called when an http request is sent to the server at the url of app.route above  
 For the function to succeed the request needs to be of type post with a json containing all the relevant fields  
 :return:  
 The function returns 200 status code for success, and also a json containing the "average" of the team requested  
 If the function fails to do this it will return 400, 404 or 500 status code (with a short message).  
 """  
 try:  
 # attempt to pull request params out of json  
 json\_data = request.json  
 # print json data to log  
 print("output\_team\_avg: json with request params")  
 print(json\_data)  
 try:  
 num\_of\_rounds = json\_data['num\_of\_rounds']  
 teamNumber = json\_data['teamNumber']  
 # if fails return status code 400 (bad request)  
 except KeyError:  
 # log error  
 print('output\_team\_avg: 400')  
 return 'Did not have necessary params in json', 400  
  
 # open a connection to the DB  
 the\_db = mysql.connector.connect(  
 host=mysql\_utils.HOST,  
 user=mysql\_utils.USER,  
 password=mysql\_utils.PASSWORD,  
 database=mysql\_utils.DB\_NAME  
 )  
 # get raw data from DB (all of the scouts pertaining to this given team)  
 scouts, foundRecords = get\_team\_records(the\_db, teamNumber)  
 print(scouts)  
  
 # return processed data if valid  
 if foundRecords: # if the dict scouts is not empty  
 res = get\_avg(scouts, num\_of\_rounds)  
 # print res before sending to client  
 print('output\_team\_avg: sending the below json back to the client')  
 print(res)  
 # send json with team average back to client  
 return res, 200  
  
 # return error if invalid (no scouts were found)  
 # log error  
 print('output\_team\_avg: 404')  
 return 'Error - scouts not found', 404  
 # code shouldn't get here but if it does this will catch all errors and return the error type to the client  
 except Exception as error:  
 # log error  
 print('output\_team\_avg: 500')  
 return type(error).\_\_name\_\_, 500  
  
  
# get team list  
@app.route(PATH\_FOR\_TEAM\_LIST\_OUTPUT, methods=['POST'])  
def output\_team\_list():  
 """  
 This function is called when an http request is sent to the server at the url of app.route above  
 For the function to succeed the request needs to be of type post with a json containing all the relevant fields  
 :return:  
 The function returns 200 status code for success, and also a json containing a list of all teams in the requested  
 competition  
 If the function fails to do this it will return 400, 404 or 500 status code (with a short message).  
 """  
 try:  
 # log for starting  
 print('output\_team\_list: starting')  
 # pull request params from json  
 json\_data = request.json  
 try:  
 num\_of\_rounds = json\_data['num\_of\_rounds']  
 compName = json\_data['compName']  
 # return error if necessary params are missing  
 except KeyError:  
 # log error  
 print('output\_team\_list: 404')  
 return 'Did not have necessary params in json', 400  
  
 # open a connection to the DB  
 the\_db = mysql.connector.connect(  
 host=mysql\_utils.HOST,  
 user=mysql\_utils.USER,  
 password=mysql\_utils.PASSWORD,  
 database=mysql\_utils.DB\_NAME  
 )  
 # get list of the teams in the competition  
 teams = get\_teams\_of\_comp(the\_db, compName)  
 # for each team get data (attack score, defense score, general score)  
 team\_list = [] # list will be filled with dicts (each dict containing the teams different scores)  
 for teamNumber in teams:  
 scouts, foundRecords = get\_team\_records(the\_db, teamNumber)  
 if foundRecords:  
 team\_list.append(get\_scores(scouts, num\_of\_rounds))  
 # return list to client (as a json) if not empty = success  
 if team\_list:  
 # log success  
 print('output\_team\_list: success, returning following json')  
 print(jsonify({'team\_list': team\_list}), 200)  
 return jsonify({'team\_list': team\_list}), 200  
 # return error if invalid - no relevant data was found  
 # log error  
 print('output\_team\_list: 404')  
 return 'Error - scouts not found', 404  
 # code shouldn't get here but if it does this will catch all errors and return the error type to the client  
 except Exception as error:  
 # log error  
 print('output\_team\_list: 500')  
 return type(error).\_\_name\_\_, 500  
  
  
# runs from here  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host=IPADDR, port=OUTPUT\_SERVER\_PORT, debug=True)

**the file: utils.py**DEV = True  
if DEV:  
 IPADDR = "0.0.0.0" # i think this should work for everything so ill leave dev always true for now  
else:  
 IPADDR = "184.72.229.230"  
INPUT\_SERVER\_PORT = 3173  
OUTPUT\_SERVER\_PORT = 3713  
HOST\_NAME = "localhost"  
PATH\_FOR\_INPUT = '/input\_server/one\_scout/json'  
PATH\_FOR\_SCOUT\_OUTPUT = '/output\_server/one\_scout/json'  
PATH\_FOR\_SCOUT\_TEAM\_OUTPUT = '/output\_server/scout\_team/json'  
AVG = '/avg'  
  
PATH\_FOR\_TEAM\_LIST\_OUTPUT = '/output\_server/team\_list/json'

**the file: \_\_init\_\_.py**