# NFO5A python project :

## Snapch'UTT

# Summary :

# Introduction

The idea behind Snapch'UTT is to create a database for a social network : we want to store the information of the users, such as their name, their age, their city etc. We were also asked to include a "following" system : a user can follow another one. Finally, we had to add a suggestion feature : based on the common followers and areas of interest, the program should propose 5 suggested users for every user.

The architecture of the database is as follows : the database is a 26-elements list, one element for each letter (in our case, we used a dictionary with the letter as the keys). Each of these elements is a list of users whose names begin with the associated letter. Finally, every user element contains his information : name, age, areas of interests(list), user that he follows(also a list), etc.

For the interface, we wanted to have a GUI, and we figured a local website would be a great solution. Basically, the python file dedicated to this website (server.py) constantly listens for any http requests. Its role is to send back any files requested, and to ask the database to update any fields.

We decided to implement a saving feature. For this, we used "pickle", a python library that allows us to store any object (a dictionary in our case) to a file and retrieve it later.

# The Algorithm and Functions

For the whole program we estimated 8hours of (only) coding, without counting the front-end development (html/js).

## In database.py :

- **CreateUser()**

For the whole program we estimated 30min of coding.

We call this function to create a user. It works first of all by checking if the username we enter for the user is available by calling CheckForPseudo(). If the username is available we create it with the information input by the user, and then we store him in the GlobalList of all the user with the function AddtoGlobal() and his username in a list of all the username (quickier to check) and it returns True. If the username is not available it just returns False.

After finishing, it calls the function save() to store the new data in a file.

- **CheckForPseudo()**

For this function we estimated 15 minutes of coding.

This function is pretty self-explanatory : it just looks in the ListofUsername if the username is available(if it's not in the list) and returns True if it is. Else it returns False.

- **AddtoGlobal()**

For this function we estimated 15minutes of coding.

This function adds the user in the global dictionary where they are all stored. To do this it checks the first letter of the name of the user and, depending on that, adds it to the corresponding “box” which are all letters of the alphabet.

- **FindUsername()**

For this function we estimated 10minutes of coding.

This function finds the user variable corresponding to the username we input. To do that it calls everyUser(), and then search in the list for the user with the right username, and then return him.

- **everyUser()**

For this function we estimated 15 minutes of coding.

This function goes through the global dictionary and stores all the users in a list to simplify the search.

- **UppdateUser()**

For this function we estimated 45minutes of coding.

This function first calls FindUsername() to find the user corresponding to the username inputted, then for all the other input of the user it changes the corresponding value in the variable and saves the update calling save().

At the end it saves the update in a file calling save()

- **DeleteUser()**

For these functions we estimated 30minutes of coding.

This function first searches in the user dictionary for the user corresponding to the username entered and deletes it. Then it goes through the list of followers of each user and if the user is in it deletes

it.After all it goes through the list of username and deletes the corresponding username.

- **Follow() UnFollow()**

for these functions we estimated 30minutes of coding.

The first function works by adding a follower to another follower list depending on the input and the other is to suppress one of the users from another users list. After the changes it saves the function by calling save().

- **ReturnFollowers()**

For this function we estimated 15min of coding.

This function calls FindUsername() for the username inputted and then returns the list of followers of this user.

- **ReturnFollow()**

For this function we estimated 30min of coding.

This function searches in the list of all the users, everyone who has the user corresponding to the username inputted in their followers list and then it returns the list of all of them.

- **FindName(), FindField(), FindYear()**

For these functions we estimated 45minutes of coding.

These functions all work the same but search something different. The user input what he wants to search, then the function calls everyUser() and searches for all the users corresponding to the inputs and returns the list of them.
FindName() search in the user's name , FindField() in the user's field of study, FindYear() in the user's year of study.

- **FindArea()**

For this function we estimated 25min of coding.

This function works by searching in the list of areas of all the users if they are the same as the input or if the input is included in the list of the user. Every user corresponding to one of these two conditions is added to a list that we return at the end of the functions.

- **SearchforUser()**

For this function we estimated 20min of coding.

This function first checks if the input is equal to nothing, if it is we return all the users in a list. if it's equal to something we call FindName() FindField() FindYear() and FindArea() and input the input of SearchforUser(). Each function will return a list of users that we will combine in a final list that we will sort to have only each user one time.

- **SuggestionSharedFollowers()**

For this function we estimated 40min of coding.

This function calls FindUsername() to see the inputted Username correspond to which User1. Then for each other user in the User1 followers list we search their followers list and make a list with all the last ones.

Once we have this list we count how many times each user appears in this one and we make a list of lists where we have a user and his score for each user. Then we return this list of lists.

- **AreaSuggestion()**

For this function we estimated 30min of coding.

This function calls FindUsername() to see the inputted Username correspond to which user1. Then it searches in everyUser() and for each user we look how many areas of interest they have in common with user1 and for each area in common we attribute them a point. Then we

make a list of lists where we have a user and his score for each user. Then we return this list of lists.

- **Suggestion()**

For this function we estimated 1 hour of coding.

This function calls FindUsername() to see the inputted Username correspond to which user1. Then it calls AreaSuggestion() and SuggestionSharedFollowers() and combine the two list obtained by the functions to have one list with a score for each user corresponding to the the score of SuggestionSharedFollowers() plus the score of AreaSuggestion().

Then we sort the final list depending on the score (the biggest the first) and return the list of only the 5 users with the biggest score in a list ( this list only contains the user without their scores).

- **save()**

For this function we estimated 15min of coding.

This function saves the content of the global List ,and of each user in it , and the list of username in a file using the pickle library.

## In server.py :

- **Serv.do_GET()**

Every GET request that arrives at the server goes through this function. We then test each route (path) to see which file we need to

send back. For the response, we start by opening the right file, then we can send the 200-SUCCESS http code, end the headers part and finally send the file converted to bytes beforehand.

- **Serv.do_POST()**

Every POST request that arrives at the server goes through this function. We then test each route (path) to see what we need to send back (most of the time it is a JSON object).

For the response, we start by reading the content of the request, we then do the appropriate action (such as editing a user, or editing one for example), then we can send the 200-SUCCESS http code. After that, we can send the good headers (Content-Type:application/json if we send a JSON object, or Access-Control-Allow-Origin:* for any object for example) end the headers part and finally send the object, converted to bytes beforehand.

# Encountered Problems

- There were a lot of small difficulties with the web site part :
    - the CORS policy is always a bit problematic, in our case, the post requests were blocked by it.
      We solved the problem by adding the right header : `Access-Control-Allow-Origin:*` right after sending the 200-success response code and reading the content of the request.
    - The conversion between a python dictionary and a json object (to send via the response of a request) was also not easy, mainly because an instance of a python class can't be serialized to JSON.
      To fix this, we created the method export inside the class User. This function converts the class instance to a python dictionary.
- The project was also a bit difficult to understand, there were a lot of fonctionnalites and some weren't clear or hard to realise, like the function follower or who are we following. After some reflection and visualisation of what the program should do and how we were fixed.
- The variables within the program were also hard to name and to differentiate because all of them were of the same type and sometimes even their function and goal were the same.
      We solved that by giving clear names to these kinds of variables and by doing functions for each program feature and even divided some features in part to have a function for each one regrouping into a bigger function.

# Instruction Manual for the program

To launch the program, you just need to start the “server.py” file, which will include by itself the database file. The program can then run in the background.
The required python packages (http, pickle and json) are part of the default modules installed with python

While it is running and to see the actual interface, open a web browser and go to the following address : *localhost:8080* or *http://127.0.0.1:8080* (these are the same addresses).

From then, the interface is pretty self-explanatory : the web site is basically a list of users. For each user, there are, in this order, the name (first + last), the nickname, the age, the field of study, the city. Then you can see 4 buttons : the first allows you to see the users who follow the selected user, the second is a list of the users followed by the selected one, the third is the same list but with the possibility to edit it and the last displays the five suggested users for the selected one. After these, you can see, by hovering with the pointer, every area of interest of the user. Finally, there is an edit and a delete button.

On the top right of the screen, you will find a text box that allows you to search for a user. You can input the name, an area of interest, a year of study or a field of study.

On the bottom, you have the possibility to refresh the list and to add a new user.

# Conclusion

To conclude, we can say that our program respects what was asked pretty well.
To optimise it, it could be a good idea to implement something like a feed, like in real social networks : every user could post something that will appear in the feed to the people following him. We could even add a like/dislike/comments section, and why not media like photos or video. A direct message chat would also be possible.
Apart from that, another architecture of the database can also be a good idea, the 26-elements list with one element for every letter is not really adapted for python, especially since it would only be optimized for a research by name, which is clearly not the main feature of the program.

# Appendix

References :

- NF05A Courses
- Open classrooms : https://openclassrooms.com/
- Doxygen documentation : https://www.doxygen.nl/manual/index.html

## CODE : database.py

```
import pickle
##
# @file database.py
# @brief This file allows us to store and edit all the informations of
the users
# @author THARAUD Valentin & SAUVAGE Eli
# @version 1.0
# @date 07/06/2021

##
# @mainpage Snapch'UTT documentation
# @section intro Introduction
#  This project was created for the NF05A course <br/>
# Original instructions : <a
href="../NF05A_Project_Snapch_UTT_Python.pdf">NF05A_Project_Snapch_UTT_
Python.pdf</a>
# @section install Installation
# @subsection run To run the project
# you will have to start the database.py file.
# For this, and with python installed on your computer, just type
# @code{.bat}
# py database.py
# @endcode
```

```
## Each user of the database will we a type User variable
class User:
    ## This initialyze the user by default
    #  @param self The object pointer.
    def __init__(self, name="none", age=0, year=0,
field="none",pseudo="none",city="none" ):# we create a class "User"
each user of the database will be a "User"
        ##the name of the user
        self.Name=name
        ##the age of the user
        self.Age=age
        ##the year of study of the user
        self.YearofStudy=year
        ##the field of study of the user
        self.FieldofStudy=field
        ##the username of the user
        self.Username=pseudo
        ##the city of the user
        self.City=city
        ##a list of the areas of interest of the user
        self.Areaofinterests=[]
        ##ab list of user followed by this user
        self.followers = []
    ##this function is used to transform an object user to a simple
dict <br/>
    # the followers field for example has to be modified to contain
only the usernames, because we won't be able to transform it into json
otherwise
    # @param self The object pointer.
    # @return a dict with fields that aren't objects (only strings or
ints)
    def export(self):
        res = dict(self.__dict__)
        res["iFollow"] = sorted([follower.Username for follower in
res["followers"]])#convert user object reference to only usernames
        res["followers"] = sorted([follow.Username for follow in
ReturnFollow(self.Username)])
        res["suggestions"] = Suggestion(self.Username)
        return res
    ##Every user will be saved in the database using his username
    # @param self The object pointer.
    # @return the username
    def __repr__(self):
```

```
        return self.Username


filename = 'save.pkl'
##loads the informations from the save file
# @return the dict stored in the file (basically the database ordered
by the first letter as asked)
def load():
    try:
        infile = open(filename,'rb')
        new_dict = pickle.load(infile)
        infile.close()
    except:#if file doesn't exist or is corrupted, we erase everything
and start back to an empty list
        print("error while reading file")
        res = {"GlobalList":{}, "ListofUsername":[]}
        for c in range(65,91):res["GlobalList"][chr(c)] = []
        return res
    return new_dict


content = load()
GlobalList = content["GlobalList"]
ListofUsername=content["ListofUsername"]
## Save the information in a file
def save():#We save the information in a file using this function
    tosave = {"GlobalList":GlobalList,
"ListofUsername":ListofUsername}#We save the content of the
globalList(all the user and their informations) and the list of
Username
    outfile = open(filename, "wb")
    pickle.dump(tosave, outfile)
    outfile.close()
## This function return the User with the corresponding username
# @param Username
# @return The User with the corresponding username
def FindUsername(username):#This function return the User with the
corresponding username
    for u in everyUser():
        if u.Username.lower()==username.lower():
            return u
##This function return all the Users with the corresponding name in a
list
```

```
# @param Name name of the users we search for
# @return The list of Users with the corresponding name
def FindName(name):
    ListofUser=[]
    for u in everyUser():
        if u.Name.lower()==name.lower():
                ListofUser.append(u)
    return ListofUser
##This function return all the Users with the corresponding field of
study in a list
# @param Field field of study of the users we search for
# @return The list of Users with the corresponding field of study
def FindField(field):
    ListofUser=[]
    for u in everyUser():
        if u.FieldofStudy.lower()==field.lower():
                ListofUser.append(u)
    return ListofUser
##This function return all the Users with the corresponding year of
study in a list
# @param Year year of study of the users we search for
# @return The list of Users with the corresponding year of study
def FindYear(year):
    ListofUser=[]
    for u in everyUser():
        if u.YearofStudy==year:
                ListofUser.append(u)
    return ListofUser
##This function return all the Users with the corresponding areas of
interst in a list
# @param area list of area of interst of the users we search for
# @return The Users with the corresponding areas of interst
def FindArea(area):
    Test=False
    ListofUser=[]
    if type(area)==list:
        for u in everyUser():
            for i in area:
                if i in u.Areaofinterests:
                    Test=True
                else:
                    Test=False
                    break
```

```
            if Test:
                ListofUser.append(u)
    return ListofUser
## This function add an user to the dictionary GlobalList in the good
section depending on the first letter of his name
# @param User the user type variable we want to add to the GlobalList
def AddtoGlobal(user):#This function add an user to the dictionary
GlobalList in the good section depending on the first letter of his
name
    for c in range(65,91):
        if chr(c)==user.Name[0] or chr(c+32)==user.Name[0]:
            GlobalList[chr(c)].append(user)
##This function check if an username is already taken (return false if
it is)
# @param Username
#@return True if the username does not exist
def CheckForPseudo(username):
    for x in ListofUsername:
        if x == username:
            return False
    return True
##this function initialyze an user, with all his information (only a
unique pseudo and a name are needed) it also add it to the globallist
and save the information in a the file
# @param pseudo,name,age,year,field,city,areaofinterests
# @return True or False depending on if the user as been created or not
def CreateUser(pseudo,
name,age=None,year=None,field=None,city=None,areaofinterests=[]):
    if CheckForPseudo(pseudo):
        user=User(name,age,year,field,pseudo,city)
        user.Areaofinterests=areaofinterests
        ListofUsername.append(user.Username)
        AddtoGlobal(user)
        save()
        return True
    else:
        return False
##This function Update an user, we need to enter his username and we
can uppdate every information it saves the information in a file at the
end
# @param pseudo,name,age,year,field,city,areaofinterests
```

```
def
UppdateUser(username,NewName="none",NewAge=0,NewYear=0,NewField="none",
NewPseudo="none",NewCity="none",NewAreas=None):
    y=FindUsername(username)
    if NewName != "none":
        y.Name=NewName
    if NewAge != 0:
        y.Age=NewAge
    if NewYear != 0:
        y.YearofStudy=NewYear
    if NewField != "none":
        y.FieldofStudy=NewField
    if NewPseudo != "none":
        y.Username=NewPseudo
    if NewCity != "none":
        y.City=NewCity
    if NewAreas != None:
        y.Areaofinterests=NewAreas
    save()#it saves the infoin the file

##This function delete an user from globallist (it delete the user and
all his information but it also search all the people following him and
erase him from their list) it also erase his username from the pseudo
list
# @param Username
def DeleteUser(username):

    for c in GlobalList.keys():
        for user in GlobalList[c]:
            if user.Username == username:
                GlobalList[c].remove(user)
    for u in everyUser():
        for f in u.followers:
            if f.Username==username:
                u.followers.remove(f)
    ListofUsername.remove(username)
    save()#it saves the infoin the file


##this function return a list of evry user of the globalList without
sorting them by letter in a dictionary
# @return a list with every user in the database
def everyUser():
```

```
    res = []
    for c in range(65,91):
        res += GlobalList[chr(c)]
    return res
##this function add an user to the follower list of an other just by
entering their usernames
# @param Username,Usernametofollow The username of the user who will
follow the user who correspond  to the usernametofollow
#@return False if the user Usernametofollow is already in the follower
of the user
def Follow(username,usernametofollow):
    user=FindUsername(username)
    userToFollow=FindUsername(usernametofollow)
    if userToFollow in user.followers:
        return False
    else:
        user.followers.append(userToFollow)
    save()#it saves the info in the file

##this function delete an user of the follower list of an other just by
entering their usernames
# @param Username,usernametounfollow The username of the user who will
unfollow the user who correspond  to the usernametounfollow
#@return False if the user usernametounfollow is not in the follower of
the user
def UnFollow(username,usernametounfollow):
    y=FindUsername(username)
    x=FindUsername(usernametounfollow)
    if x not in y.followers:
        return False
    else:
        y.followers.remove(x)
    save()#it saves the infoin the file
## this function return all the followers of an user in a list
#@param username the username of the user we want the list
def ReturnFollower(username):
    y=FindUsername(username)
    return y.followers

##This foncion call FindArea(), FindField(), FindUsername(), FindName()
and FindYear() and search in all the user who correspond to what we are
searching
# @param searchfor the keyword you want to research
```

```
#@return the list of user corresponding
def SearchForUser(searchfor):
    #this function search in all the user info what we want (it could
be a name an area or multiple areas of interst a year or a field of
study or a username) and return all the users corresponding to that
information in a list
    if searchfor == "":
        return everyUser()
    else:

ListofUser=list(set(FindArea(searchfor.split())+FindField(searchfor)+[F
indUsername(searchfor)]+FindName(searchfor)+FindYear(searchfor)))
        ListofUser.remove(None)
        return ListofUser


## this function first search for all the followers of an USER then
search for all the user that the USER followers follow, then it
attibutes a score for evry USER followers followers.
#@param username the username of the user we want the suggestion for
#@return the list of list of the user with their score
def SuggestionSharedfollowers(username):# this function first search
for all the followers of an USER then search for all the user that the
USER followers follow, then it attibutes a score for evry USER
followers followers.
    ListofUser=[]
    ListofFollower=[]
    y=FindUsername(username)
    for j in y.followers:
        for i in j.followers:
            ListofFollower.append(i)
    ListofFollower=list(set(ListofFollower))
    ListofFollower.remove(y)
    for x in range(len(ListofFollower)):
        total=0
        for j in y.followers:
            for i in j.followers:
                if i==ListofFollower[x]:
                    total=total+1
        ListofUser.append([ListofFollower[x],total])
        ListofUser.sort(key=lambda elem:elem[1], reverse=True)
    return ListofUser
```

```
##this function attribute a score for each user depending on how many
areas of interest corespond to the area of the user we input, rach area
in common is a point, it return a list a list(off all the list) of
list(with a user and his score)
#@param username the username of the user we want the suggestion for
#@return the list of list of the user with their score
def AreaSuggestion (username):
    #this function attribute a score for each user depending on how
many areas of interest corespond to the area of the user we input, rach
area in common is a point, it return a list a list(off all the list) of
list(with a user and his score)
    ListofUser=[]
    y=FindUsername(username)
    for u in everyUser():
        if u!=y:
            total=0
            for area in u.Areaofinterests:
                if area in y.Areaofinterests:
                    total=total+1
            ListofUser.append([u.Username,total])
    return ListofUser
## this function first search for all the followers of an USER then
search for all the user that the USER followers follow, then it
attibutes a score for evry USER followers followers.
#@param username the username of the user we want the suggestion for
#@return the list of the five users we suggest
def Suggestion(username):# this function first search for all the
followers of an USER then search for all the user that the USER
followers follow, then it attibutes a score for evry USER followers
followers.
    ListofUser=[]
    ListofFollower=[]
    y=FindUsername(username)
    for j in y.followers:
        for i in j.followers:
            ListofFollower.append(i.Username)
    ListofFollower=list(set(ListofFollower))
    if y.Username in ListofFollower:
        ListofFollower.remove(y.Username)
    for x in range(len(ListofFollower)):
        total=0
        for j in y.followers:
            for i in j.followers:
```

```
                if i.Username==ListofFollower[x]:
                    total=total+1
        ListofUser.append([ListofFollower[x],total])
    ListofUser2=AreaSuggestion(username)#this function attribute a
score for each user depending on how many areas of interest corespond
to the area of the user we input, rach area in common is a point, it
return a list a list(off all the list) of list(with a user and his
score)
    ListofUser=ListofUser+ListofUser2#Next we add the two list
    UserToRemove=[]
    for x in range(len(ListofUser)-1):#for all the users who are in the
two list we add both of the score to have the total
        for i in range(x,len(ListofUser)-1):
            if ListofUser[x][0]==ListofUser[i+1][0]:
                UserToRemove.append(ListofUser[i+1])
    for x in UserToRemove:
        ListofUser.remove(x)
    ListofUser.sort(key=lambda elem: elem[1], reverse=True)
    return [elem[0] for elem in ListofUser[:5] if elem[1]>0]#we only
take the 5 first usernames if their score is > 0

## this function return all the people that follow an user in a list
#@param username the username of the user we want the list
def ReturnFollow(username):
    ListofUser=[]
    for u in everyUser():
        for uF in u.followers:
            if uF.Username==username:
                ListofUser.append(u)
    return ListofUser
```

## CODE : server.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import json
import database

##
# @file server.py
# @brief This file allows the user to interact with the main file
(database) throught a website
# @author THARAUD Valentin & SAUVAGE Eli
# @version 1.0
# @date 07/06/2021

##
# @brief This is the main Class using the http.server package
class Serv(BaseHTTPRequestHandler):
    ##
    # @brief GET requests function
    # every GET request that arrives to the server goes throught here,
we then test for the right route(path) to know what to send back
    # @param self default param for a class method
    # @return None
    def do_GET(self):
        if self.path == '/':
            file_to_open = open("./views/index.html").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/addUser.html":
            file_to_open = open("./views/addUser.html").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/js/index.js":
            file_to_open = open("./views/js/index.js").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/js/jquery.min.js":
```

```
            file_to_open = open("./views/js/jquery.min.js").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/bootstrap.min.css":
            file_to_open = open("./views/bootstrap.min.css").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/js/bootstrap.min.js":
            file_to_open = open("./views/js/bootstrap.min.js").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))

        elif self.path == "/js/poper.min.js":
            file_to_open = open("./views/js/poper.min.js").read()
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes(file_to_open, 'utf-8'))
    ##
    # @brief POST requests function
    # every POST request that arrives to the server goes throught here, we then test for the right route(path) to know what to read, what to modify in the database and to know what to send back
    # @param self default param for a class method
    # @return None
    def do_POST(self):
        if(self.path == "/newMemb"):
            content_length = int(self.headers['Content-Length']) # <--- Gets the size of data
            post_data = self.rfile.read(content_length) # <--- Gets the data itself
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            if newMemb(json.loads(post_data.decode('utf-8'))):
                self.wfile.write(bytes(json.dumps({"response":"ok", "code":0}), "utf-8"))
            else:
```

```
self.wfile.write(bytes(json.dumps({"response":"alreadyTaken",
"code":1}), "utf-8"))
        if(self.path == "/editUser"):
            content_length = int(self.headers['Content-Length']) # <---
Gets the size of data
            post_data = self.rfile.read(content_length) # <--- Gets the
data itself
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            editUser(json.loads(post_data.decode('utf-8')))
            self.wfile.write(bytes(json.dumps({"response":"ok",
"code":0}), "utf-8"))
        elif(self.path == "/delUser"):
            content_length = int(self.headers['Content-Length']) # <---
Gets the size of data
            post_data = self.rfile.read(content_length).decode("utf-8")
# <--- Gets the data itself
            database.DeleteUser(json.loads(post_data)["username"])
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            self.wfile.write(bytes("ok", "utf-8"))

        elif self.path == "/getUsers":
            content_length = int(self.headers['Content-Length']) # <---
Gets the size of data
            post_data = self.rfile.read(content_length).decode("utf-8")
# <--- Gets the data itself
            # print(post_data)
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            users = [user.export() for user in
database.SearchForUser(post_data)]
            # print(users)
            self.wfile.write(bytes(json.dumps(users), "utf-8"))
        elif(self.path == "/follow"):
            content_length = int(self.headers['Content-Length']) # <---
Gets the size of data
```

```
            post_data = self.rfile.read(content_length).decode("utf-8")
# <--- Gets the data itself
            data = json.loads(post_data)
            self.send_response(200)
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            # print(data)
            if database.FindUsername(data["userToFollow"]):
                if data["follow"]:
                    database.Follow(data["user"], data["userToFollow"])
                else:
                    database.UnFollow(data["user"],
data["userToFollow"])
                self.wfile.write(bytes(json.dumps({"response":"ok",
"code":0,
"newList":database.FindUsername(data["user"]).export()["iFollow"]}),
"utf-8"))
            else:

self.wfile.write(bytes(json.dumps({"response":"notFound", "code":1,
"newList":database.FindUsername(data["user"]).export()["iFollow"]}),
"utf-8"))

##
# @brief adds a new member to the database
#
# @param memb dictionnary with the names of the fields of the html form
as keys
# @return True if user created, False is pseudo already taken
def newMemb(memb):
    return database.CreateUser(memb["pseudo"], memb["name"],
memb['age'], memb['year'], memb['field'], memb["city"], memb["areas"])

##
# @brief edit a member to the database
#
# @param memb dictionnary with the names of the fields of the html form
as keys
# @return None
def editUser(user):
    # print(user)
```

```
    database.UppdateUser(user["oldPseudo"], user['name'], user["age"],
user["year"], user["field"], NewCity=user["city"],
NewAreas=user["areas"])

httpd = HTTPServer(('127.0.0.1',8080),Serv)
httpd.serve_forever()
```