First Session

Date: 10/16/2019

- Overview of assignment and related topics
- Downloading putty as a client application
- Make a paradigm of program

In this session we read assignment accurately and analyzed each part of structures.

And then we have had an overview on Subjects to cover**:** OOP, Inheritance, Async I/O, Error Handling, Exceptions, Testing, TCP/IP, Coroutines.

To have an obvious attitude Putty application is installed it is considered as a temporary client.

A paradigm of Classes in program which needs to handle the project is considered. We need one User class and an Admin class which Admin inherit from User and we should have two module one of them related to server another one related to service and so we will consider another module which related to client. All modules will work as one application Test will consider as Unittest inside of each module.

Second day:

Date: 10/17/2019

- Connect GitHub group to python and check how it works.
- Connect putty to python
- Start to write codes in send-back function in server modules, this codes ask client to enter name, password and privilege

After we connected them successfully, we found that when each member changed the modules in python, other members cannot see the changes. However, we solve this problem by reading more about python and GitHub. We found that each member has to commit after any changes in python, so other members would have a notification of changes, so after updating, other members are able to see anything written in python. And we pass this step successfully.

Then, we tried to install putty as client on our computers and connect it to python through main class in server module. The connection in localhost 127.0.0.1 port 8080.

In next step we read accurately the 14th pdf of python class to understand how we can get input and send sth as an output on putty via python. This is mentioned in page 30. We need StreamReader and StreamWriter method from async class. So now we know how to write something in our putty and ask a client to enter his/her information (name/password/privilege). We made our server module which contains main class. In main class we connected to putty and call send back function which had made in server module.

Every process from putty to python will be done in send back function. Any command which is needed to ask from client was written in the function and we get the answers and put them in a variable, so we can then save them in files if necessary.

Third day

Date: 10/18/2019

- Making portfolio and sharing tasks.
- Defining directory structure
- Working on login and register

In this session tasks of each person are scheduled. We decided in each day every person had a responsibility of writing report and project leader will be another one in each session according to our portfolio.

Server module is created, initializing and two condition of server module related to register and login are implemented. Registering new user based on name, password and privilege of each client is implemented in the last session. To prevent repeating registered usernames we need to have a file to check new users' name with previous registered usernames. In these session we have had an overview to how we can save registered information to compare the items in the future. We found Jason file can help to reach the goal.

Directory structure:

Root

|---Admin

|---admin1

|---info.txt

|---admin2

|---info.txt

|---User

|---user1

|---info.txt

|---user2

|---info.txt

|---server

|---signed-info.json

|---registered-info.json

Forth day:

Date: 10/19/2019

- Overviewing on functionalizing the program
- Login part is completed
- Using Jason file to save register and login data
- Invisible password → Fail

In this session functionalizing of program has been reviewed.

One part of server module is completed which is related to log in. There are some weaknesses like accepting all type of name even question mark or slash we hope to handle such problem in the next sessions.

We consider Jason file to save registered user via their ”name, password, privilege” such file considered for logged in user there are differences between “registered Jason file” and “signed Jason file” the former is a dictionary including three list” name, password, privilege” and the latter is a list including name.

Signed Out of logged in user is considered.

Invisible password was a part which is failed.After asking the problem we consider it as a not obligatory object.

Fifth day

Date: 10/20/2019

- Functionalizing →→→ FAILED
- Using Service functions in Server module
- Working on read_file command

We try to functionalize the Server module last session and also tried to use functions of Service which are in user and admin class from Server module. However, we faced a problem for using Os. build in functions: write and read. They cannot be used in other functions. An error appeared when we use them in other functions.

The other problem was we were not able to use Service functions from Server module. We try to use them while the class User was imported in Service module.

After reading about accessibility of functions and about how we can use a function from another file, we found that we have to make an object from a class and then we can call the class functions by that object. So we make an object from class User and named it client:

client = User(name, password, privilege)

And then we call function, like function list (which list current directory files and folders):

client.list (name, privilege)

But we still have problem for using write and read (build-in function of os) in our functions.

Also functionalizing faced failure in this step

Sixth day:

Date: 10/21/2019

- Working on create_ folder
- Error handling of read function and create folder

In the session we focused on Error handing for read function and create folder.

In the session Error handling for create_folder the Os.error is reviewed and used.

Another try was related to find the bug when different folder is implemented for read command.

Read file command based on structure is completed but it has not been functionalized.

Seventh day:

Date: 10/22/2019

- Create folder for all sub directories
- Implementing cd .. commands
- Making list command with considering the size and date of creation
- Date part in list → Failed

There were a problem in which create_folder did not work correctly for sub directory the reason was related to commands that used and when we changed command to “os.makedirs(path)” the problem is tackled.

The command cd .. should only be available if the user is not currently standing in Root. It did not worked correctly and went back to other folders before root. The issue has been tackled with a condition and error handling is considered. Based on structure current working directory for the current user to the specified folder residing in the current folder is possible. And if the <name> does not point to a folder in the current working directory, the request is to be denied with a proper message highlighting the error for the user.

In list command, all features like the name of file and folders and the size are considered but date and time of creation in writer.write command was missed after searching and using ctime command the structure of list command work correctly so now, all files and folders in the current working directory for the user issuing the request will be printed. This command is expected to give information about the name, size, date and time of creation, in an easy-to-read manner. Information regarding content in sub-directories will not printed.

Os module to how we can write create folder simpler is searched and studied.

Eighth day:

Date: 10/23/2019

- Modifying folder date print
- List function →→→→ FAIL
- Write_file function
- Read_file function

In list function, there was a problem in folders date printing format, which was because of missing below command after using get date command:

time.ctime(date)

There is also another problem in list function. The folders info is not printed in ordered columns(the printed list was like below list, all files and just first folder are printed correctly and in order). And we could not handle it in this session.

File2 time of creation date

File2 time of creation date

Folder1 time of creation date

Folder2 time of creation date

Folder3 time of creation date

Challenging part of Write_file and read_file function was: the path of creating file when it was not created in advance and printing next 100 character for next read command.

Ninth day:

Date: 10/24/2019

- Delete function is programmed this function is just Admin ability not User
- Focused in all options of read_file
- Create_folder with user has problem which refer to path → Failed
- Create_folder
- Clean signed Json file after server reset to prevent problem relating re-log in the same user

In the session delete the user conforming with <username> from the server is considered. This service is only available to users with a privilege level of admin, and <password> is to be the password of the admin currently logged in. Error handling is programmed, If the request is done by a user that does not have admin privileges, <password> does not match or if the <username> does not exist, an error message should be returned to the request for the client to present to the user.
Create_folder with user has problem the problem refers to the last edited in object.

The server must be able to handle two different kinds of users—user and admin. Some service requests must be denied if done by a user and not an admin. This option is done.

In read_file all aspects has been considered like:
1- Read data from the file <name> in the current working directory for the user issuing the request and return the first hundred characters in it.
2- Each subsequent call by the same client is to return the next hundred characters in the file, up until all characters are read.
3-If a file with the specified <name> does not exist in the current working directory for the user, the request is to be denied with a proper message highlighting the error for the user.
4-A service request without a <name> variable should close the currently opened file from reading.
5-Subsequent calls with this file as <name> should start reading from the beginning of the file

The last but not the least was cleaning signed json file after server reset

Tenth day:

Date: 10/25/2019

- log in from different session
- service requests done by different users should not interfere
- Error: socket.send() → Failed
- Overviewing on client module

In login function in service module log in from different session should possible. If a user is already logged in, further requests to log in with that username should be denied. It was failed in this session but the problem handled in session 13.

According to the structure, the server is only required to be able to handle one service request at a time and service requests done by different users should not interfere with one another unless necessary. In this part necessary condition is vague but we consider the point as admin who can read, write and delete each folder. For instance, if two users use the command "read_file" but with different files, then subsequent calls done by these users are to handle the specific user's file and not any other file. The structure implemented in read file function of service module in User class.

At the first we assume that there was a problem in the Server showed Error: socket.send() and Error exception and try for removing error related to disconnecting putty should be considered but this part remained failed because after session 15 when client module was created such problem had not has any effects on our program since client module is more intelligent and convenient compares to putty.

Eleventh day:

Date: 10/28/2019

- Modifying Json file address
- Delete folder of the client who was deleted by admin
- Deleting a client username, password and privilege from Json files when the client was deleted by admin
- Controlling “cd ..” to *only* be available if the user is not currently standing in Root
- Limiting each client to work in a directory of their own (a user can just enter to his/her directory and make changes)

The first problem one json files because we made it in root/server and when CWD get changed in a function (for example in change folder directory), we could not read or write json file there. And we have an error “the json file is not in this address”. So we try to save the CWD at the first of service module and we named it init_cwd = str(os. getcwd()) and its equal the place of your project in your own computer, so when we use our client-server application in different computers, we would not face any problem. Then we use “ init_cwd/root/server/registered.json “ in address part when we want to read or write in json file.

Deleted client: while we were testing our application, we found that the deleted client could not registered again. Because we did not delete it in our json file and also we did not delete the directory of its own. So we modify our delete function and complete it to solve this problem. Fortunately, it works correctly and deleted client could register again, because they are unknown for server.

Controlling users when they are moving through folders was the last part we did in this session. We have to control they walk back to previous folder until root. So we put restrictions in going back function, we asked the current folder, if the client was in root we return an error “Error: You are in root directory”.

Next restriction was where they want to enter a folder, user can just enter user folder and then just his/her own directory, so we set a restriction, asked for the privilege and current place:

1) if it was user and stood in root, he/she just can enter user folder. If it was admin, he/she can go anywhere.

2) If it was user and stood in user folder, he/she just can enter his/her own folder (which name is the same as user name). If it was admin, he/she can go anywhere.

3) If it was not 2 last, it means the user is in his/her own directory and he can go anywhere he want. And he can do anything(changing folder, making folder, …). If the folder name was not exist, the exception occur so we handle it with try exception, and return and error “Error: The folder does not exist.”

Twelfth day:

Date: 10/29/2019

- Starting to Using Client.py and synchronizing with Server module

Thirteenth day:

Date: 10/30/2019

- Using split command to exactly follow by commands structure
- Read_file command reviewed and tried to handle errors in this file
- In this session we have some questions
- Some Questions
- Test via Unittest
- Restriction.punctuation in register and logged in

Split command: A problem was in entering commands in which commands imported separately. The entering command has been changed to the format which has been pointed in structure Like: register <username> <password> <privileges> in order and not separately, to achieve the purpose "split commands" is used in all functions and now all commands in server and client is exactly as same as structure of assignment 3 Table 1.

Reviewing on read_file command and related restriction. There are four conditions in this method. The first condition is related to when read command implemented without any file name.

The second condition is related to when in the current working Directory for the user issuing the request and return the first hundred characters in it.

The third if condition is related to when two users use the command "read_file" but with different files, then subsequent calls done by these users are to handle the specific user's file and not any other file

The forth if condition is related to each subsequent call by the same client is to return the next hundred characters in the file, up until all characters are read.

One flag has been defined in server module which check file name this flag helps to the goal that service request without a <name> variable should close the currently opened file from reading. Subsequent calls with this file as <name> should start reading from the beginning of the file.

In read function error handling has been implemented for when the file with the pointed name does not exist or user issues a command of read file with standing up in a wrong path.

In this session we have some questions and asked professor about them which were related to client part and saving commands between sessions is necessary or not?

Test: We defined all type of tests which should be implemented in Test part of program all of which pointed to problems that will occur during working with the application. All of tests will be implemented, described in Test report.

There was a problem in login and registration in which user had not any restriction when wanted make a name. Therefor restriction.punctuation has been implemented to user be able to make a name in meaningful way. The error handling of this part has been implemented too. Also in creating folder such restriction for choosing name has been used and Error handling was done.

Fourteenth day:

Date: 11/02/2019

- Modifying Client.py
- Modifying function
- Start writing test via Unittest for create_folder
- Completing all return function in service module

We used putty first, because it accelerates our work. However, we have to use client.py instead of putty. As they are differences between putty and client.py, we faced some issues. One of the problems was something like delay in our application: it means when we entered the first command we did not get any answer, but when we entered the second command the answer of first command appeared in terminal and go on. After reading about async, reader and writer we found that our last command and it response was saving somewhere and in next step in appears, it was because we did not use “await writer.drain” after each writer.write.

Then we found that when a response is not in one writer.write we have this problem again. So change our server and service. We just call function of service in server by its object and got the answers from the functions (by using return … in service functions) and send the message from server to client.

Fifteenth day:

Date: 11/03/2019

- Concluding the report related to program
- Concluding the report of Test
- Finalizing the Test part of the program
- Uploading file