**CHAPTER 1 : ABSTRACT**

Online Quiz System (OQS) is a web-based examination system where quiz is taken online i.e. through the internet or intranet using computer system. The purpose of OQS is to take Semester Quizzes in an efficient manner and no time wasting for checking the paper. The main objective of OQS is to efficiently evaluate the candidate through a fully automated system that not only saves lot of time but also gives fast results. Teachers can administer quizzes using the OQS .The system will show result after the examination is finished. A teacher has control in the question bank and is supposed to make schedule for the quiz. The system carries out the examination and auto-grading for multiple choice questions which is fed into the system. Administrative control of the whole system is provided.

The main objective of the project on MCQ Quiz Applications is to manage the details of Students, Examinations, Marks, Courses, Papers. It manages all the information about Students, Results, Papers,Sudents. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for Marks, Couses, Papers.

**TABLE OF CONTENTS**

CHAPTERS CONTENTS PAGE NO.

1. ABSTRACT 4

2. INTRODUCTION 5-6

3. REQUIREMENT ANALYSIS 6-8

4. ARCHITECTURE & DESIGN 8-12

5. IMPLEMENTATION 12-13

6. EXPERIMENT RESULTS & ANALYSIS 13-26

7. REFERENCES 27

**CHAPTER 2 : INTRODUCTION**

It is the online quiz where two users can take the quiz and select the answers to the given questions.After completing the quiz,the score will be displayed and tells,which user has won the quiz and it displays how much time the user has taken to complete the quiz.Here first we have to run the server,and then the two clients which mean symbolically two users and the quiz questions will be in Ui format.

**CHAPTER 3 : REQUIREMENT ANALYSIS**

**2.1 Hardware Requirements**

|  |  |
| --- | --- |
| **Number** | **Description** |
| 1 | PC with 250 GB or more Hard disk. |
| 2 | PC with 2 GB RAM. |
| 3 | PC with Pentium 1 and Above. |

**2.2 Software Requirements**

|  |  |  |
| --- | --- | --- |
| **Number** | **Description** | **Type** |
| 1 | Operating System | Windows XP / Windows |
| 2 | Language | Python |
| 3 | Database | MySQL |
| 4 | IDE | Jupyter |
| 5 | Browser | Microsoft Edge |

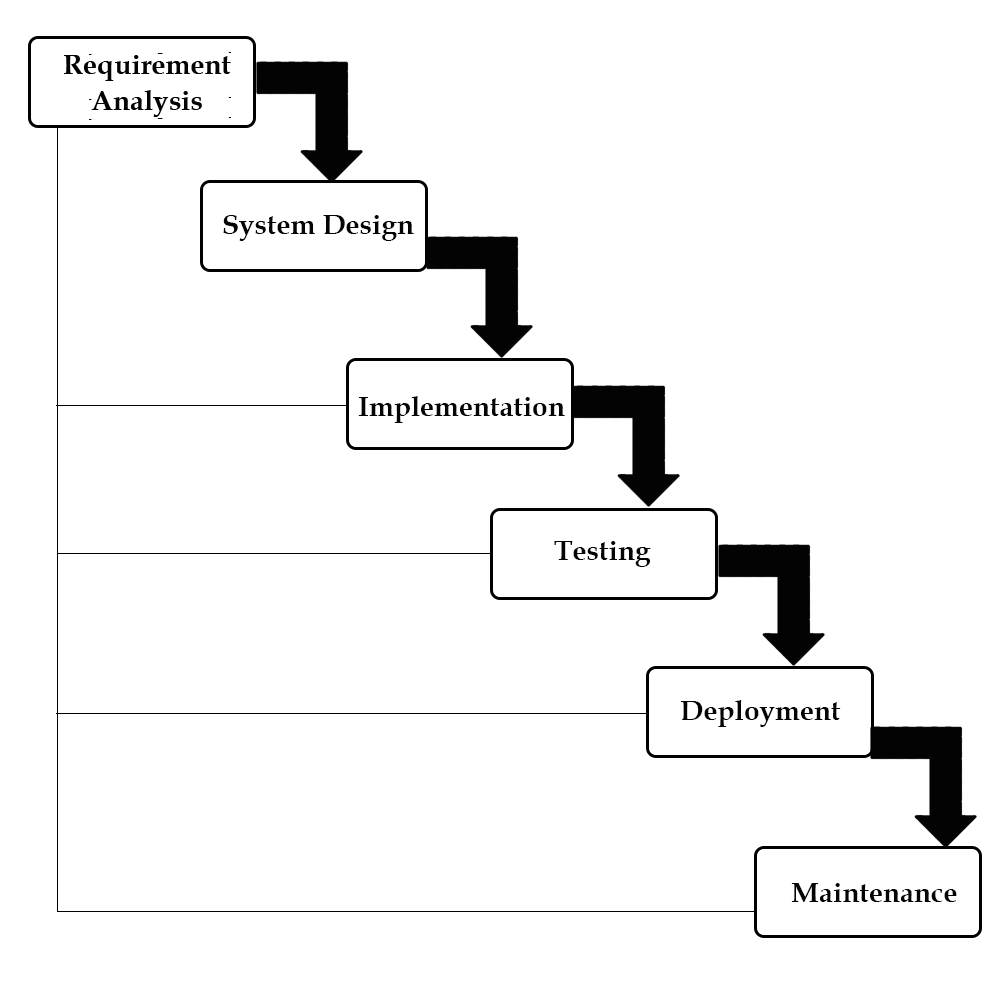
**CHAPTER 4 : ARCHITECTURE & DESIGN**

**3.1 Software Development Life Cycle Model**

**3.1.1 WATERFALL MODEL**

The waterfall model was selected as the SDLC model due to the following reasons:

* Requirements were very well documented, clear and fixed.
* Technology was adequately understood.
* Simple and easy to understand and use.
* There were no ambiguous requirements.
* Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
* Clearly defined stages.
* Well understood milestones.
* Easy to arrange tasks.



### Assumptions:-

* The client can connect to the server IP.
* The client has the server address.
* The server has a parameter NUM\_PLAYERS restricting the number of players.
* The quiz will begin only when all players have joined.
* The folder structure of the project remains unchanged.
* The client tries to connect only when the server is up.
* There are no connection problems between the server and client in the duration of the quiz.

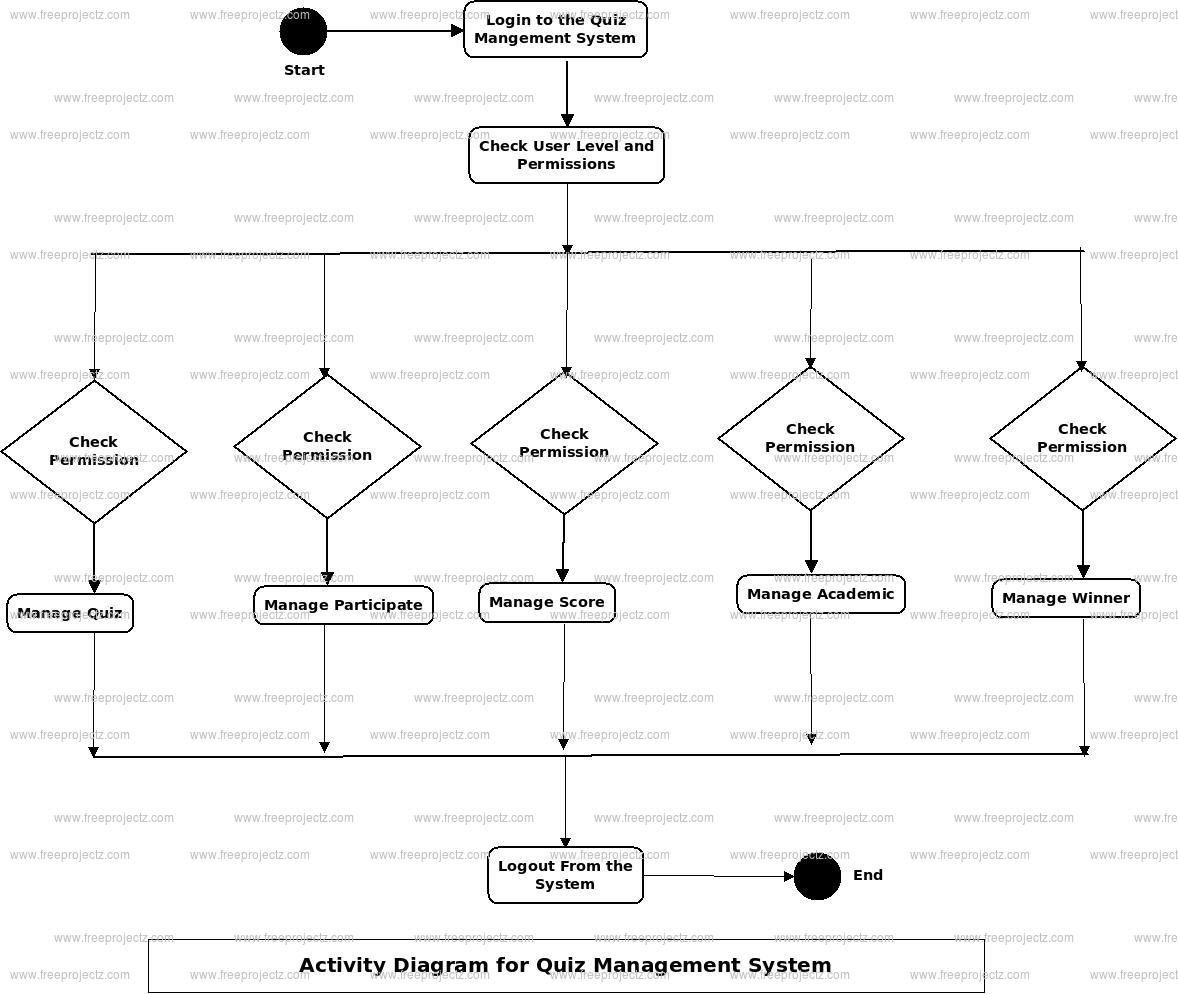
### Execution Instructions:-

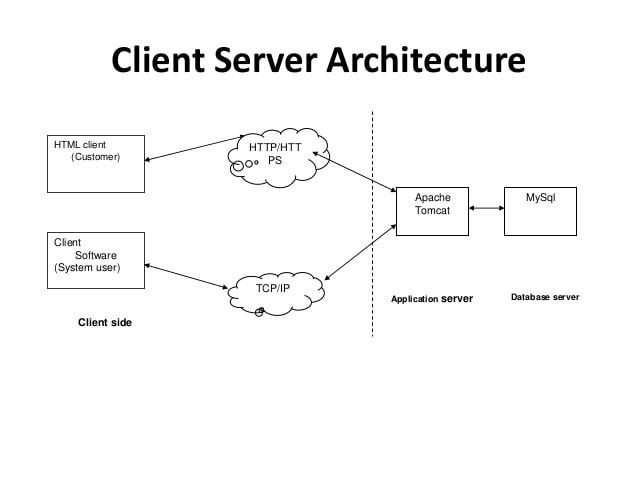
* Set the NUM\_PLAYERS.
* Pass the server IP address to the client.
* Start the server.
* Enter the quiz using the client file with updated server details.
* After the quiz ends exit the server.

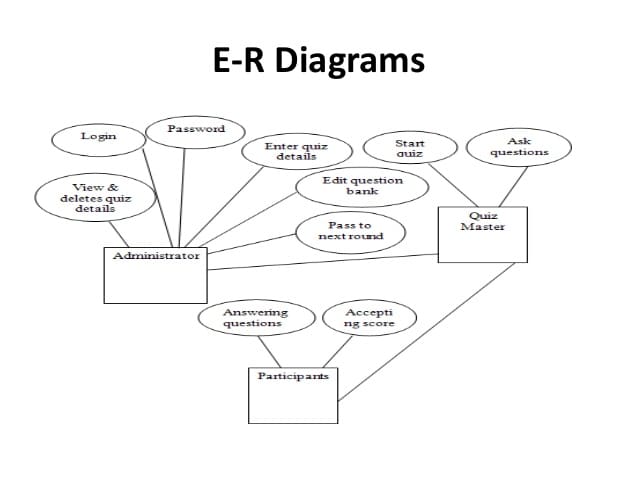
### Rules and Regulations:-

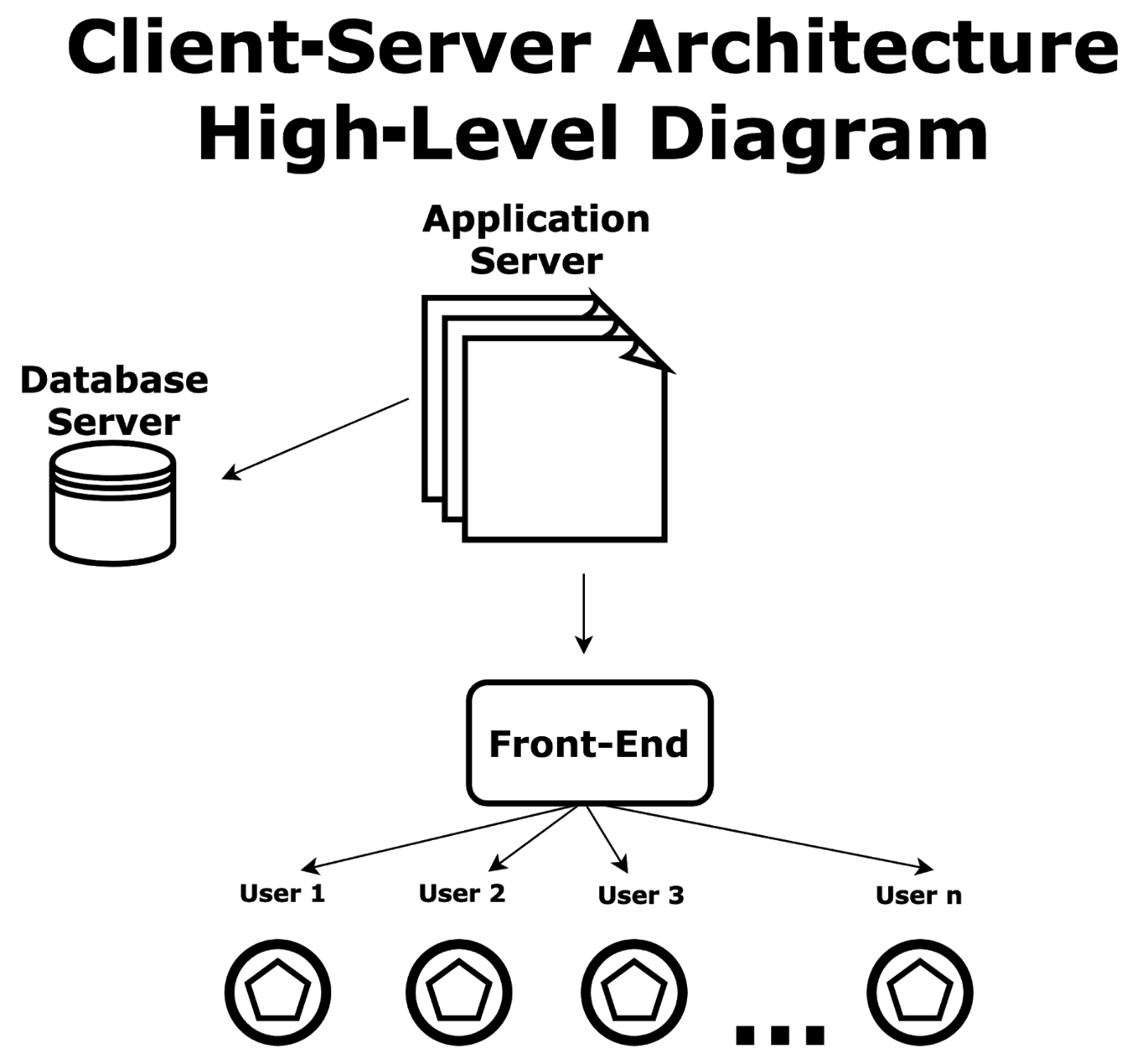
* Each participant will be given questions and options.
* The participant has to select one of the four options.
* There is a time limit for each question.
* Each question is equally weighted.
* If none of the questions is marked at the end of the timer, a default wrong response gets recorded.
* If the user had selected an option but failed to submit the option by the end of the quiz, the selected option gets recorded.
* The scores of the participants are given at the end of the quiz.
* The tie-breaker in case of equal scores will be the time taken to choose the option.

**UML Diagram**









**CHAPTER 5 : IMPLEMENTATION**

### User Interface:-

We have also added a user interface for the quiz, so instead of using the terminal, the participants in the quiz can select one radio button from a set of 4. We also have a submit button and a timer as part of the UI. At the start of each question for a participant, the timer is set at 10 seconds (can be modified according to the constant parameter WAIT\_TIME), and if the participant fails to select and submit any of the options in that time, the question will be marked wrong and the next question will be loaded. In case a button is selected but the submit button is not pressed within WAIT\_TIME seconds, the selected button will be taken as the answer given by the participant.

### Question Bank:-

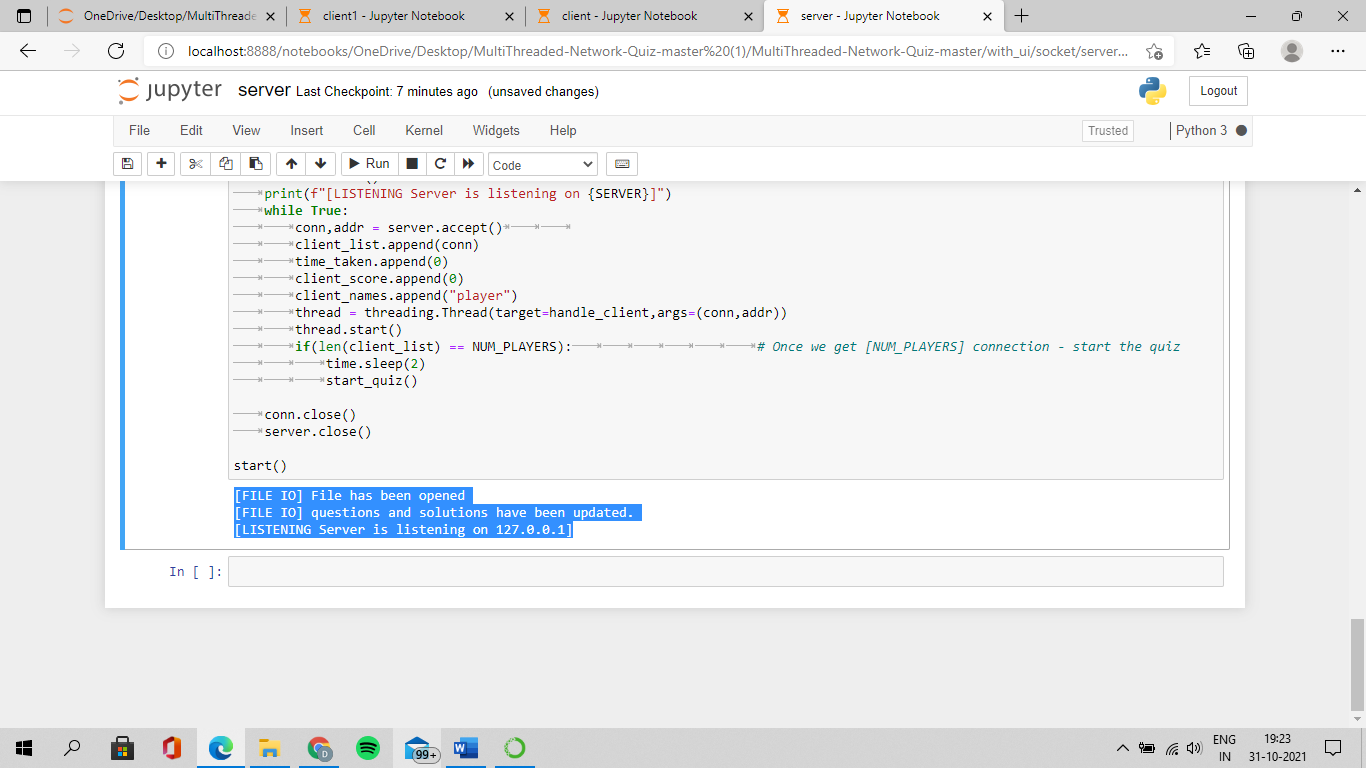
We also maintain a questionBank.txt file where all the questions to be asked are stored. The getQuestions.py contains a function which reads the file and updates the list of questions and solutions to be used by the server. The server upon starting calls this function to update the question and solution list. The questionBank.txt is a self-explanatory file and any number of questions can be added to it.

### Code Parameters

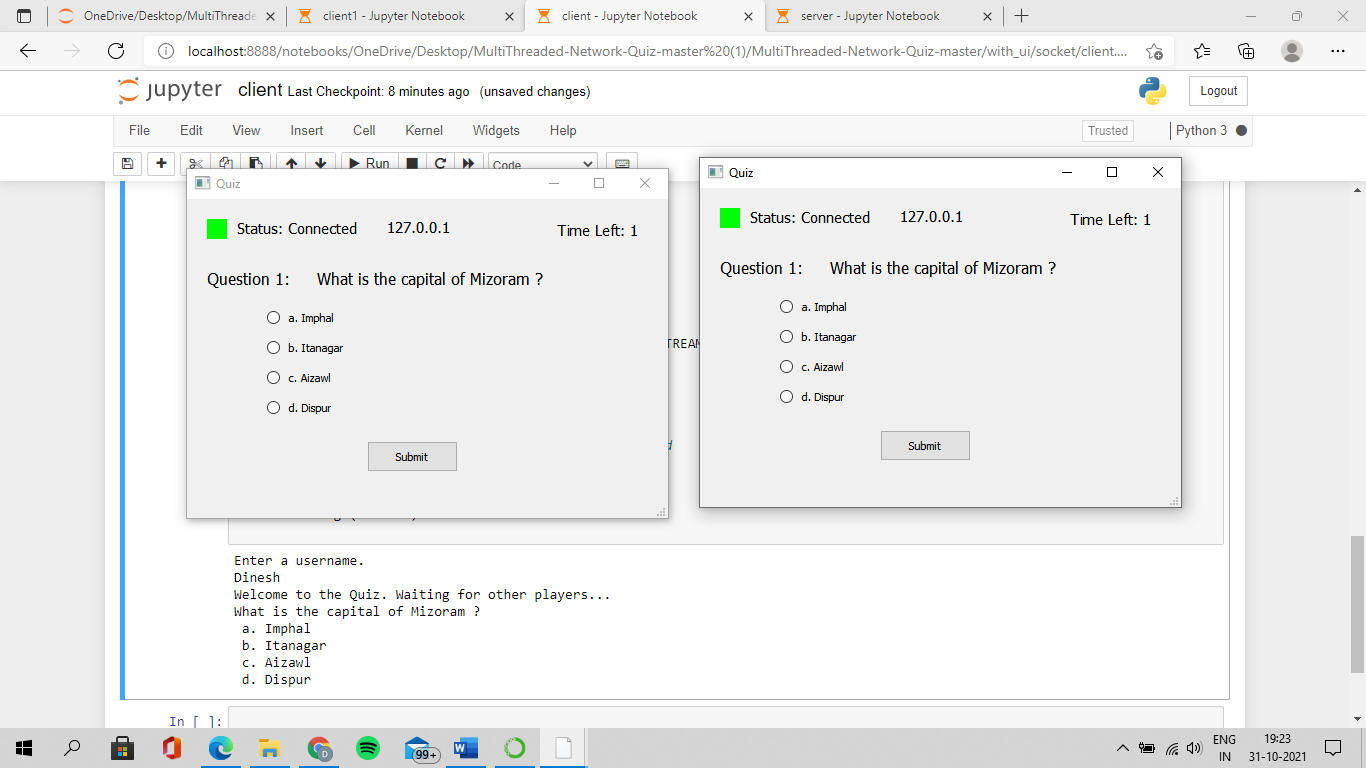
* WAIT\_TIME : Specifies the amt of time for each question. Has to be changed in client.py.
* NUM\_PLAYERS : Specifies the number of client connections required for the quiz. Given in server.py.
* SERVER : Sepcifies server address. Present in both client and server. Should be updated properly.
* PORT : Sepcifies port address. Present in both client and server. Should be updated properly.
* FORMAT : Specifies format of the packet that is sent or received over the network
* DISCONNECT\_MSG : Specifies the message which when received causes the client to close. Indicates Game Over state.

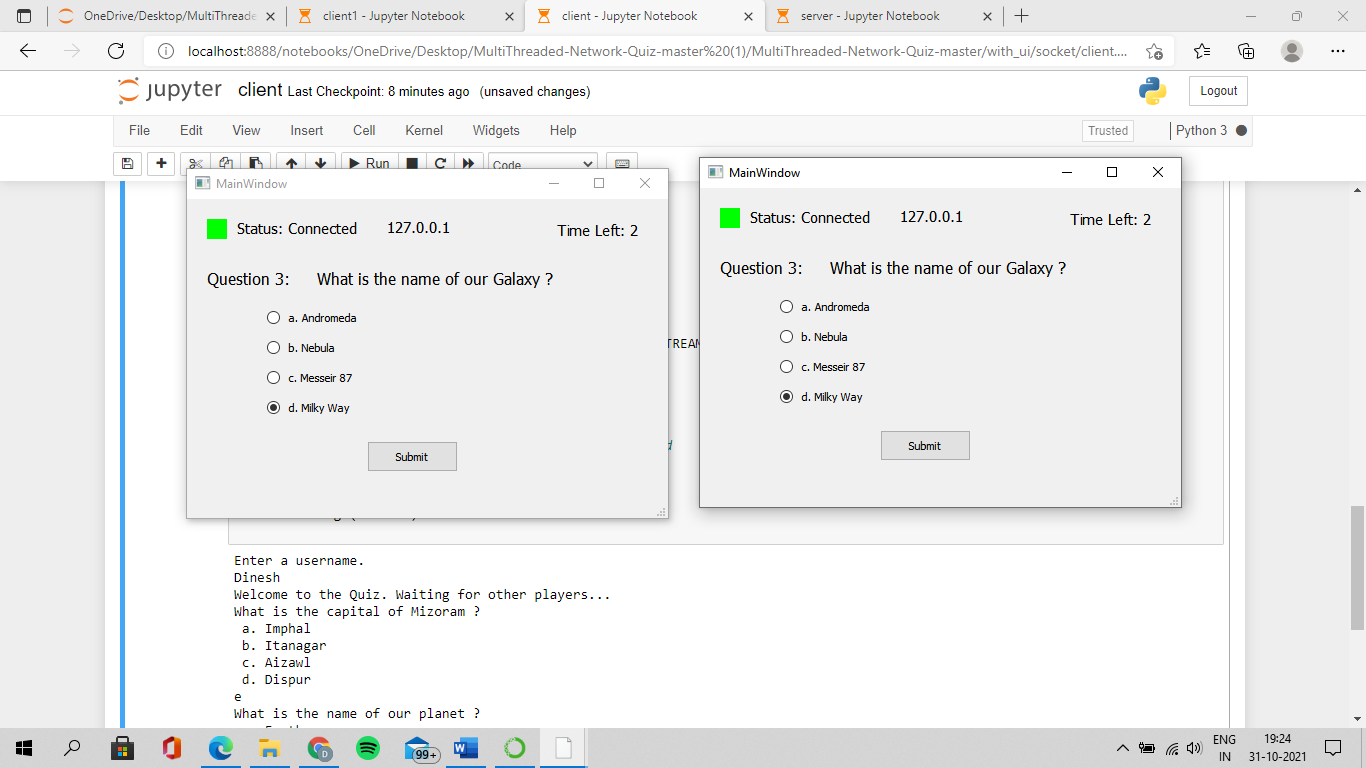
**CHAPTER 6 : EXPERIMENT RESULTS AND ANALYSIS**

First,we have to run the server,then the file with questions and answers will get updated and server will be connecting to IP address.

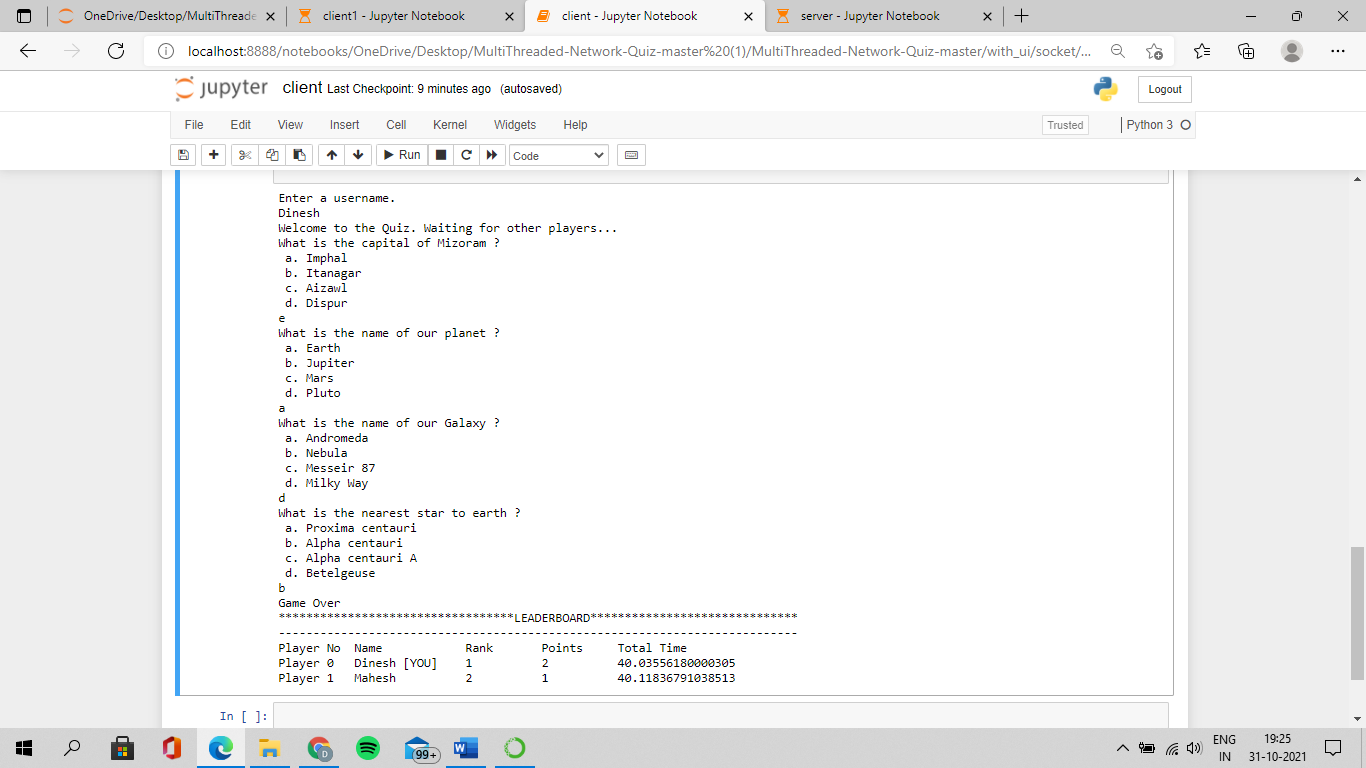


Then we have to run the client which is user1 who will be going to take the quiz and we have to run one more client which is user2 who will be going to take the quiz ,then the quiz starts with timelimit of 10 seconds to each question.

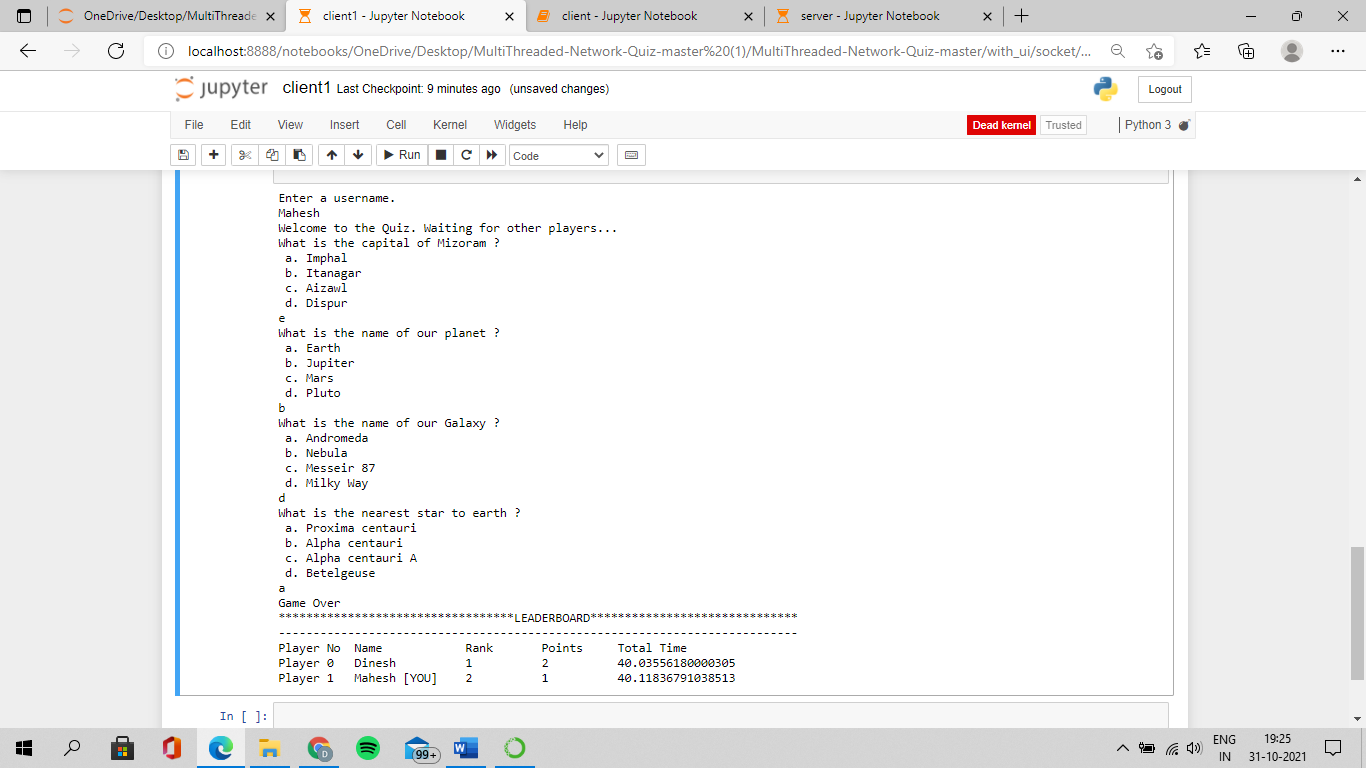




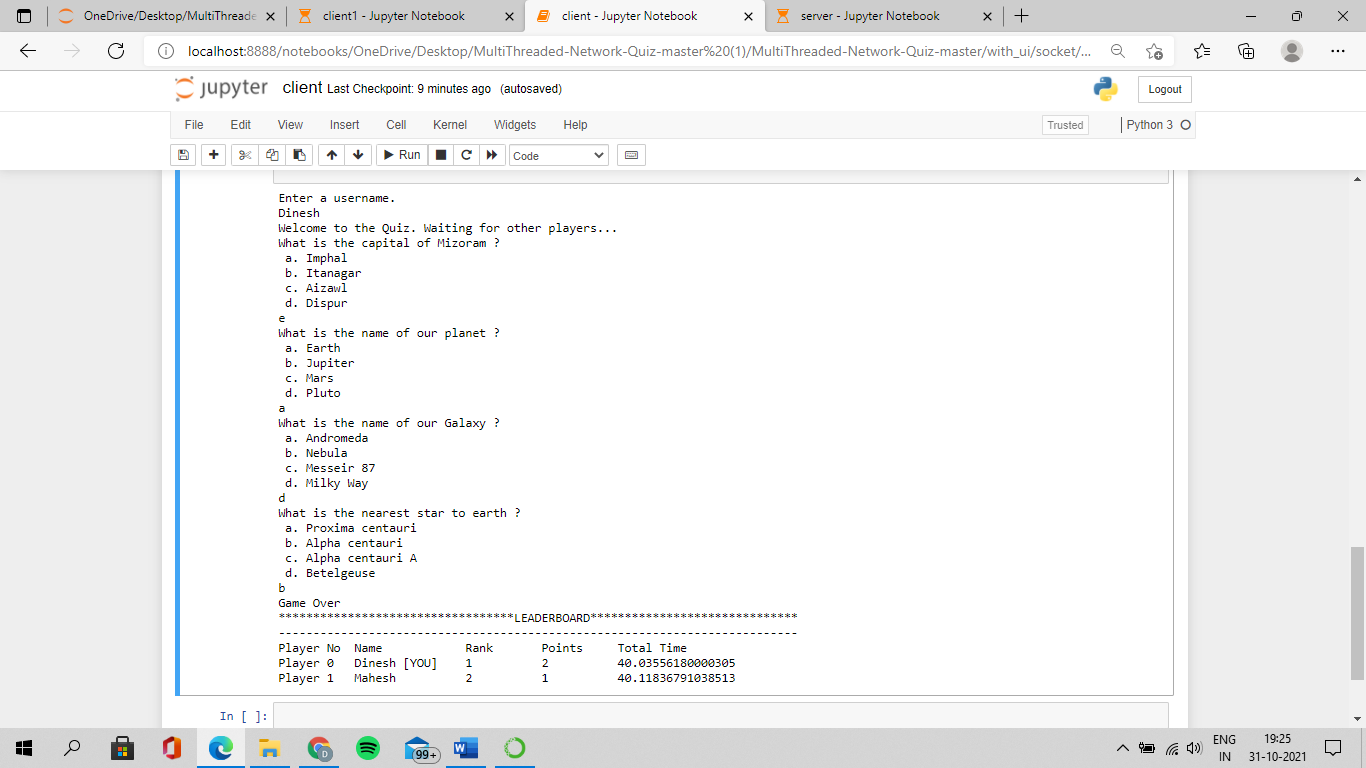
Whatever u will be updating answers in UI,you will be getting the status in client.If u haven’t choosen any option in given timelimit,then the status will be saved as e.

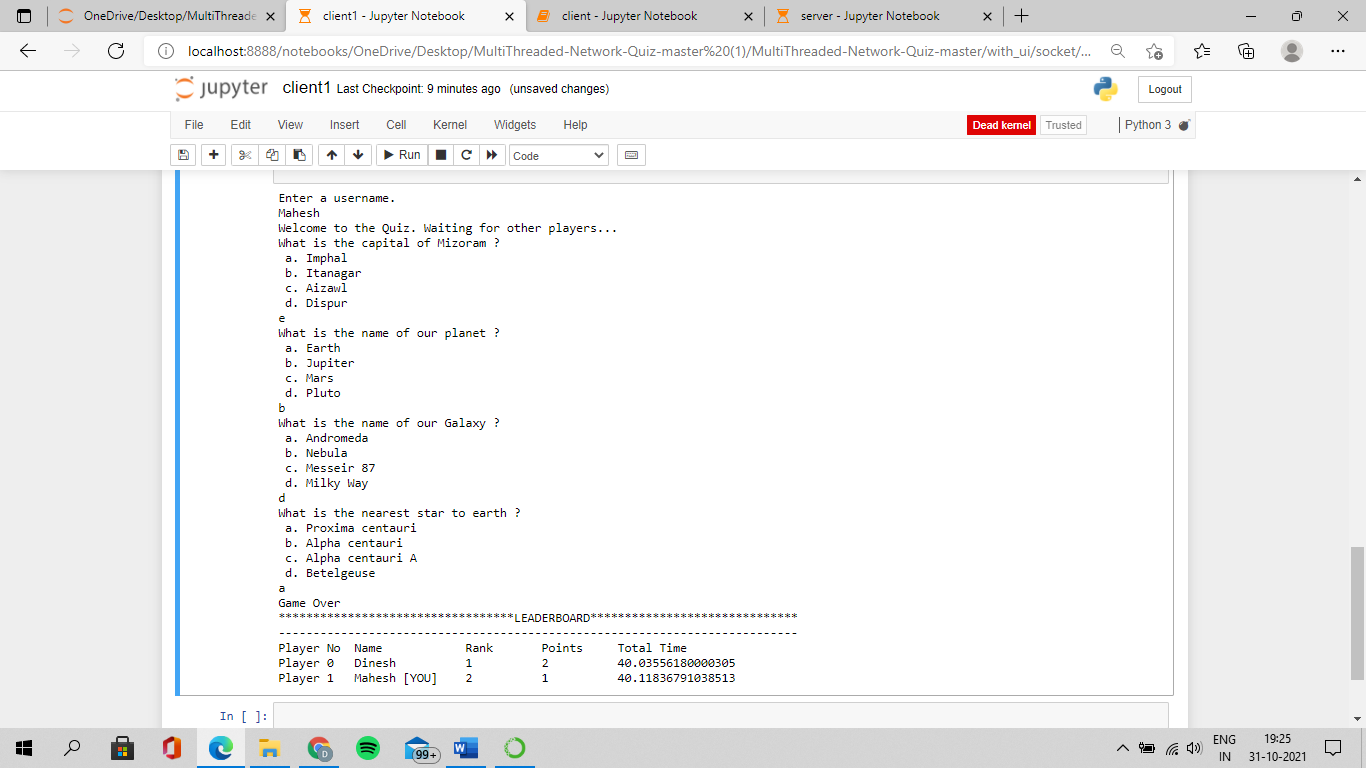


The same status of answers which was answered by user2 will also get updated.

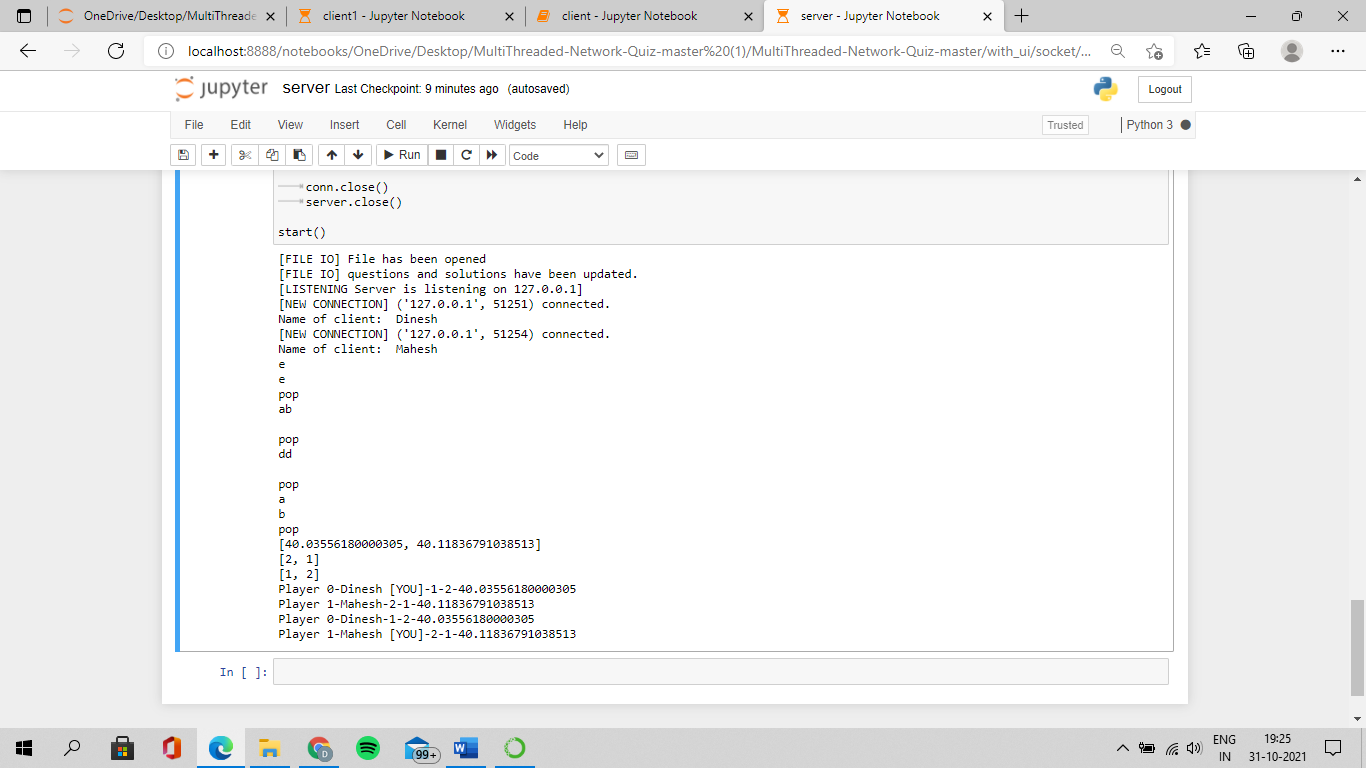


After completion of all the questions,ur answers will be checked and u can see who is winner of the quiz and how much time he has taken to attend the quiz will also be displayed.





At last,you will get all information like what are the options chosen by both users,how much time they have taken to answer etc.,



Code for Server:-

|  |
| --- |
| import socket |
|  | import threading |
|  | import time |
|  | import sys |
|  | import numpy as np |
|  | from getQuestions import \* |
|  |  |
|  | # Define const. parameters |
|  | MAX\_LEN = 64 |
|  | PORT = 9001 |
|  | SERVER = "127.0.0.1" |
|  | ADDR = (SERVER,PORT) |
|  | FORMAT = "utf-8" |
|  | DISCONNECT\_MSG = "Game Over" |
|  | WAIT\_TIME = 30 |
|  | NUM\_PLAYERS = 2 |
|  |  |
|  | # Global variables |
|  | client\_list = [] |
|  | client\_names = [] |
|  | time\_taken = [] |
|  | client\_score = [] |
|  | locked\_client = [] |
|  | thread\_lock = threading.Lock() |
|  |  |
|  | # Create server socket |
|  | server = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) |
|  | server.bind(ADDR) |
|  |  |
|  | # Define the question set to be used |
|  | questions = [] |
|  | solutions = [] |
|  |  |
|  | # Function to handle client responses concurrently |
|  | def handle\_client(conn,addr): |
|  | print(f"[NEW CONNECTION] {addr} connected.") |
|  | # Welcome Message |
|  | msg = "Welcome to the Quiz. Waiting for other players..." |
|  | msg = msg.encode(FORMAT) |
|  | conn.send(msg) |
|  |  |
|  | # receive the username of client |
|  | username = conn.recv(1024).decode(FORMAT) |
|  | indx = client\_list.index(conn) |
|  | client\_names[indx] = username |
|  | print("Name of client: ",username) |
|  |  |
|  | connected = True |
|  | while connected: |
|  | # receive response from the client |
|  | response = conn.recv(1024).decode(FORMAT) |
|  | print(response) |
|  | conn\_indx = client\_list.index(conn) |
|  | time.sleep(0.1) |
|  | time\_duration = conn.recv(1024).decode(FORMAT) |
|  | # print(time\_duration) |
|  | time\_duration\_float = float(time\_duration) |
|  | # print(time\_duration\_float) |
|  | time\_taken[conn\_indx] = time\_taken[conn\_indx] + time\_duration\_float |
|  |  |
|  | # Update score |
|  | if(response == solutions[0]): |
|  | client\_score[conn\_indx] = client\_score[conn\_indx] + 1 |
|  |  |
|  | locked\_client.append(conn) |
|  |  |
|  | # Wait for the minute to get over |
|  | while(len(locked\_client)%NUM\_PLAYERS !=0): |
|  | pass |
|  |  |
|  | thread\_lock.acquire() |
|  | if(locked\_client[-1] == conn): |
|  | print("pop") |
|  | questions.pop(0) |
|  | solutions.pop(0) |
|  |  |
|  | # Pop already asked question and ask another question |
|  | if len(questions) == 0: |
|  | end\_quiz() |
|  | break |
|  | else: |
|  | start\_quiz() |
|  | thread\_lock.release() |
|  | server.close() |
|  |  |
|  | def start\_quiz(): |
|  | if(len(questions)!=0): |
|  | for connection in client\_list: |
|  | connection.send(questions[0].encode(FORMAT)) |
|  |  |
|  | def end\_quiz(): |
|  | broadcast("Game Over") |
|  | print(time\_taken) |
|  | print(client\_score) |
|  |  |
|  | time.sleep(1) |
|  |  |
|  | # calculate rank |
|  | rank = [0]\*NUM\_PLAYERS |
|  | # based on the key - lowest key value gets highest rank |
|  | indices = [i for i in range(NUM\_PLAYERS)] |
|  | neg\_time = [((-1) \* i) for i in time\_taken] |
|  | c = list(zip(client\_score,neg\_time,indices)) |
|  | c = sorted(c) |
|  | for pos in range(len(c)): |
|  | rank[c[pos][2]] = NUM\_PLAYERS - pos |
|  | print(rank) |
|  |  |
|  | # Now broadcast the number of players |
|  | broadcast(str(NUM\_PLAYERS)) |
|  | time.sleep(1) |
|  |  |
|  | for client in client\_list: |
|  | for player in client\_list: |
|  | indx = client\_list.index(player) |
|  | rank\_msg = "Player "+ str(indx) + "-" + client\_names[indx] |
|  | if(indx == client\_list.index(client)): |
|  | rank\_msg = rank\_msg + " [YOU]" |
|  | rank\_msg = rank\_msg + "-" + str(rank[indx]) + "-" + str(client\_score[indx]) + "-" + str(time\_taken[indx]) |
|  | print(rank\_msg) |
|  | client.send(rank\_msg.encode(FORMAT)) |
|  | time.sleep(0.1) |
|  | time.sleep(0.3) |
|  | final\_msg = "Sorry. You came in " + str(rank[client\_list.index(client)]) + ". Better Luck next time." |
|  | if(rank[client\_list.index(client)] == 1): |
|  | final\_msg = "Congrats! You have won the quiz." |
|  | client.send(final\_msg.encode(FORMAT)) |
|  |  |
|  | # Close all connections |
|  | for clients in client\_list: |
|  | clients.close() |
|  | sys.exit() |
|  |  |
|  | def broadcast(message): |
|  | for clients in client\_list: |
|  | clients.send(message.encode(FORMAT)) |
|  |  |
|  |  |
|  | # server start listening |
|  | def start(): |
|  | # Let us update the questions and solutions |
|  | getQuestions(questions,solutions) |
|  |  |
|  | server.listen() |
|  | print(f"[LISTENING Server is listening on {SERVER}]") |
|  | while True: |
|  | conn,addr = server.accept() |
|  | client\_list.append(conn) |
|  | time\_taken.append(0) |
|  | client\_score.append(0) |
|  | client\_names.append("player") |
|  | thread = threading.Thread(target=handle\_client,args=(conn,addr)) |
|  | thread.start() |
|  | if(len(client\_list) == NUM\_PLAYERS): # Once we get [NUM\_PLAYERS] connection - start the quiz |
|  | time.sleep(2) |
|  | start\_quiz() |
|  |  |
|  | conn.close() |
|  | server.close() |
|  |  |
|  | start() |

Code for Client:-

|  |  |
| --- | --- |
| import socket | |
|  | import sys |
|  | import select |
|  | import os |
|  |  |
|  | sys.path.append(os.path.normpath("../ui/")) |
|  | from question import \* |
|  | from home import \* |
|  |  |
|  | HEADER = 64 |
|  | PORT = 9001 |
|  | SERVER = "127.0.0.1" |
|  | ADDR = (SERVER,PORT) |
|  | FORMAT = 'utf-8' |
|  | DISCONNECT\_MSG = "Game Over" |
|  |  |
|  |  |
|  | def closeConn(client): |
|  | # Now let us print the leaderboard before closing connection |
|  | print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LEADERBOARD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") |
|  | print("---------------------------------------------------------------------------") |
|  | number\_players = client.recv(4096).decode(FORMAT) |
|  | number\_players = int(number\_players) |
|  | print("%-10s %-15s %-10s %-10s %-10s" %("Player No","Name","Rank","Points","Total Time")) |
|  | for i in range(number\_players): |
|  | rank\_msg = client.recv(4096).decode(FORMAT).split("-") |
|  | print("%-10s %-15s %-10s %-10s %-10s" %(rank\_msg[0],rank\_msg[1],rank\_msg[2],rank\_msg[3],rank\_msg[4])) |
|  |  |
|  | final\_msg = client.recv(4096).decode(FORMAT) |
|  | print("---------------------------------------------------------------------------") |
|  | print(final\_msg) |
|  | print("---------------------------------------------------------------------------") |
|  | client.close() |
|  | sys.exit() |
|  |  |
|  | def recvMessage(client): |
|  | # Receive the question from the |
|  | q\_msg = client.recv(4096).decode(FORMAT) |
|  | print(q\_msg) |
|  |  |
|  | # If the received message is "Game Over" then exit |
|  | if(q\_msg == DISCONNECT\_MSG): |
|  | closeConn(client) |
|  |  |
|  | # Extract question and options from the message. |
|  | q\_msg = q\_msg.split("\n") |
|  | question = q\_msg[0] |
|  | options = q\_msg[1:] |
|  |  |
|  | # Create the Application |
|  | app = QtWidgets.QApplication(sys.argv) |
|  | MainWindow = QtWidgets.QMainWindow() |
|  | ui = Ui\_MainWindow1() |
|  |  |
|  | # set the UI according to the new question receieved. |
|  | ui.setupUi\_1(MainWindow,question,options,client,ui) |
|  | MainWindow.setWindowTitle("Quiz") |
|  | MainWindow.show() |
|  | sys.exit(app.exec\_()) |
|  |  |
|  | def recvMessageF(client,ui,MainWindow): |
|  | # Receive the question from the |
|  | q\_msg = client.recv(4096).decode(FORMAT) |
|  | print(q\_msg) |
|  |  |
|  | # If the received message is "Game Over" then exit |
|  | if(q\_msg == DISCONNECT\_MSG): |
|  | MainWindow.close() |
|  | closeConn(client) |
|  |  |
|  | # Extract question and options from the message. |
|  | q\_msg = q\_msg.split("\n") |
|  | question = q\_msg[0] |
|  | options = q\_msg[1:] |
|  |  |
|  | # set the UI according to the new question receieved. |
|  | ui.setupUi\_1(MainWindow,question,options,client,ui) |
|  | ui.updateQuestionIndex() |
|  |  |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  |  |
|  | # get username input |
|  | print("Enter a username.") |
|  | username = input().strip() |
|  |  |
|  | clientS = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) |
|  | clientS.connect(ADDR) |
|  | # receive the welcome and rules message |
|  | wlcm\_msg = clientS.recv(4096).decode(FORMAT) |
|  | print(wlcm\_msg) |
|  |  |
|  | # send the username to the server for the leaderboard |
|  | clientS.send(username.encode(FORMAT)) |
|  |  |
|  | # close this application |
|  | recvMessage(clientS) |

Question bank txt:-

|  |  |
| --- | --- |
|  |  |
|  | BEGIN |
|  | ------------------------------------------------------------------------------------------------------------------- |
|  | Question: |
|  | What is the capital of Mizoram ? |
|  | Options: |
|  | Imphal |
|  | Itanagar |
|  | Correct: Aizawl |
|  | Dispur |
|  |  |
|  | Question: |
|  | What is the name of our planet ? |
|  | Options: |
|  | Correct: Earth |
|  | Jupiter |
|  | Mars |
|  | Pluto |
|  |  |
|  | Question: |
|  | What is the name of our Galaxy ? |
|  | Options: |
|  | Andromeda |
|  | Nebula |
|  | Messeir 87 |
|  | Correct: Milky Way |
|  |  |
|  | Question: |
|  | What is the nearest star to earth ? |
|  | Options: |
|  | Proxima centauri |
|  | Alpha centauri |
|  | Correct: Alpha centauri A |
|  | Betelgeuse |
|  |  |
|  | END |
|  | --------------------------------------------------------------------------------------------------------------------- |

Project Implementation Schedule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr no. | Task Name | Duration | Start Date | End Date |
| 1 | Planning | 7 days | 20-10-2021 | 27-10-2021 |
| 2 | Design | 5 days | 27-10-2021 | 01-11-2021 |
| 3 | Implementation | 2 days | 01-11-2021 | 03-11-2021 |
| 4 | Testing | 1 day | 03-11-2021 | 04-11-2021 |

.

Test cases:-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case no.** | **Test Case** | **Expected Output** | **Actual Output** | **Result** |
| 1 | Verifying data | Entering the name of users and storing data | Successfully entered the name of users and stored the data | Passed |
| 2 | Time Limit | If the timelimit for ech question is completed,it will go to next question | It is going to next question once the timelimit completes | Passed |

Result:-The multithreaded online quiz system was implemented and executed successfully.

**CHAPTER 7 : REFERENCES**

<http://allproject4u.blogspot.com/2013/01/online-quiz-portal-project.html>

<http://blog.jitbit.com/2007/08/why-we-still-use-vb6-for-some-projects.html>