University of Bahrain
College of Information Technology

ITNE352
Corse Project:

Multithreaded Flight arrival Client/Server Information System

Made By:
Abdulrahman Jalal Zaid
Abdulqadeer Abdulhamid

Table Of Contents

Introduction:

The client-server model, a foundational architecture for distributed systems, has traditionally addressed client requests sequentially, limiting its efficiency in handling simultaneous requests. This limitation becomes particularly noticeable in situations where real-time data, such as aviation information, is critical. Implementing multi-threading transforms this model, enabling servers to concurrently engage with multiple clients in real-time. In the scope of aviation, where immediate access to flight data is as critical as the flights themselves, a system capable of handling multiple client requests concurrently becomes a necessity.

The aviationstack.com API serves as a gateway to real-time flight data, presenting an opportunity to enhance client-server interaction [1]. The API implementation becomes integral to the server-side functionality. The server, upon receiving client requests, pulls the API to fetch the real-time flight information. The use of multi-threading is critical in optimizing this process, allowing the server to simultaneously handle multiple client requests without sacrificing responsiveness [2].

In this project, we created a client-server model that exchanges some flight information via API from aviationstack.com with the proper parameters. The server implements multi-threading where it accepts connections from multiple clients (up to three clients) and handles their requests. Moreover, a simple graphics interface (GUI) was added on the client side for more user-friendly interaction for the user.

Design:

The Multithreaded Flight Arrival Client/Server Information System model is implemented through two distinct Python scripts, one dedicated to server functionality and the other to client interactions, via a shared JSON file. The server script initiates by establishing a TCP socket and connecting to an external API to fetch flight information based on user-specified parameters(arr_icao) and stored in a JSON file. All data acquired from the API are dumped into the JSON file with indentation of 2. Afterward, when the file is opened in read-only mode, we load the file and use some arguments (if statements / for loops) to extract the required data. Then the server turns its socket from active to passive and can accept three connections at the same time.

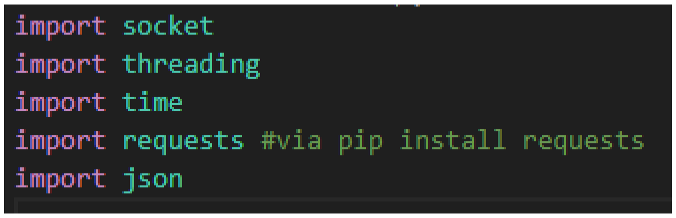
On the client side, the script begins by creating a TCP socket, connecting to the server, and transmitting the client's name to facilitate thread identification. Upon successful connection, the server creates a new thread, providing real-time updates on the connected client's name and ongoing requests. Simultaneously, the client is presented with a menu offering a range of services, enabling them to request specific flight information. The server efficiently handles these requests by retrieving the required data from the pre-saved JSON file and transmitting it back to the client. This iterative process continues until the client decides to exit and terminate the connection.

Importantly, all connected clients share a common arrival airport, minimizing redundant API connections and file overwrites. Both scripts have exception-error handling to deal with client/server errors.

Implementation & main Elements:

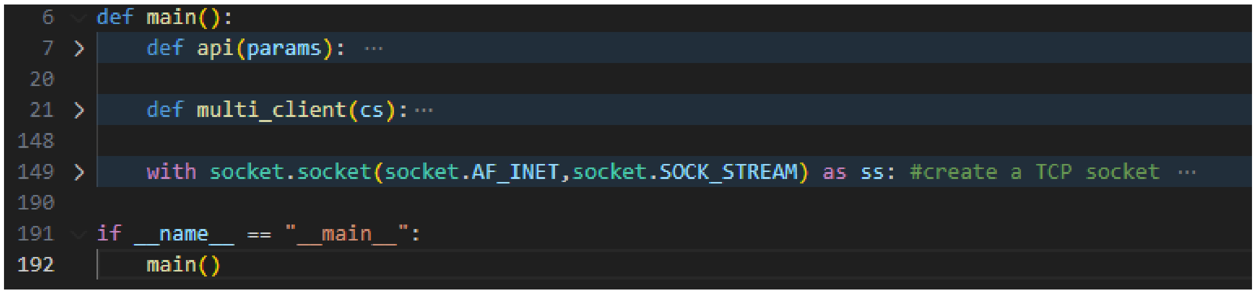
1. Server script

Firstly, we used some modules to enable the server to function as required:



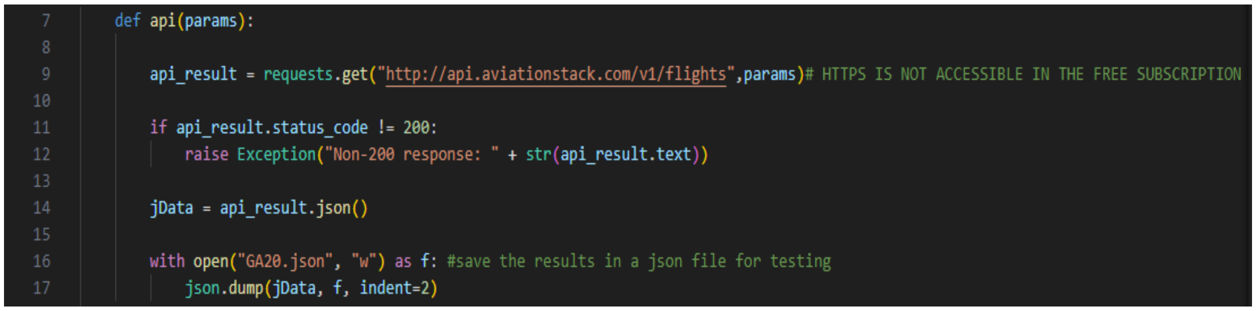
```python
import socket
import threading
import time
import requests #via pip install requests
import json
```

Then we designed the server script to be in three blocks. All the blocks are in the main () function to ensure that certain code in a Python script is executed only when the script is run directly, not when it is imported as a module into another script [3].



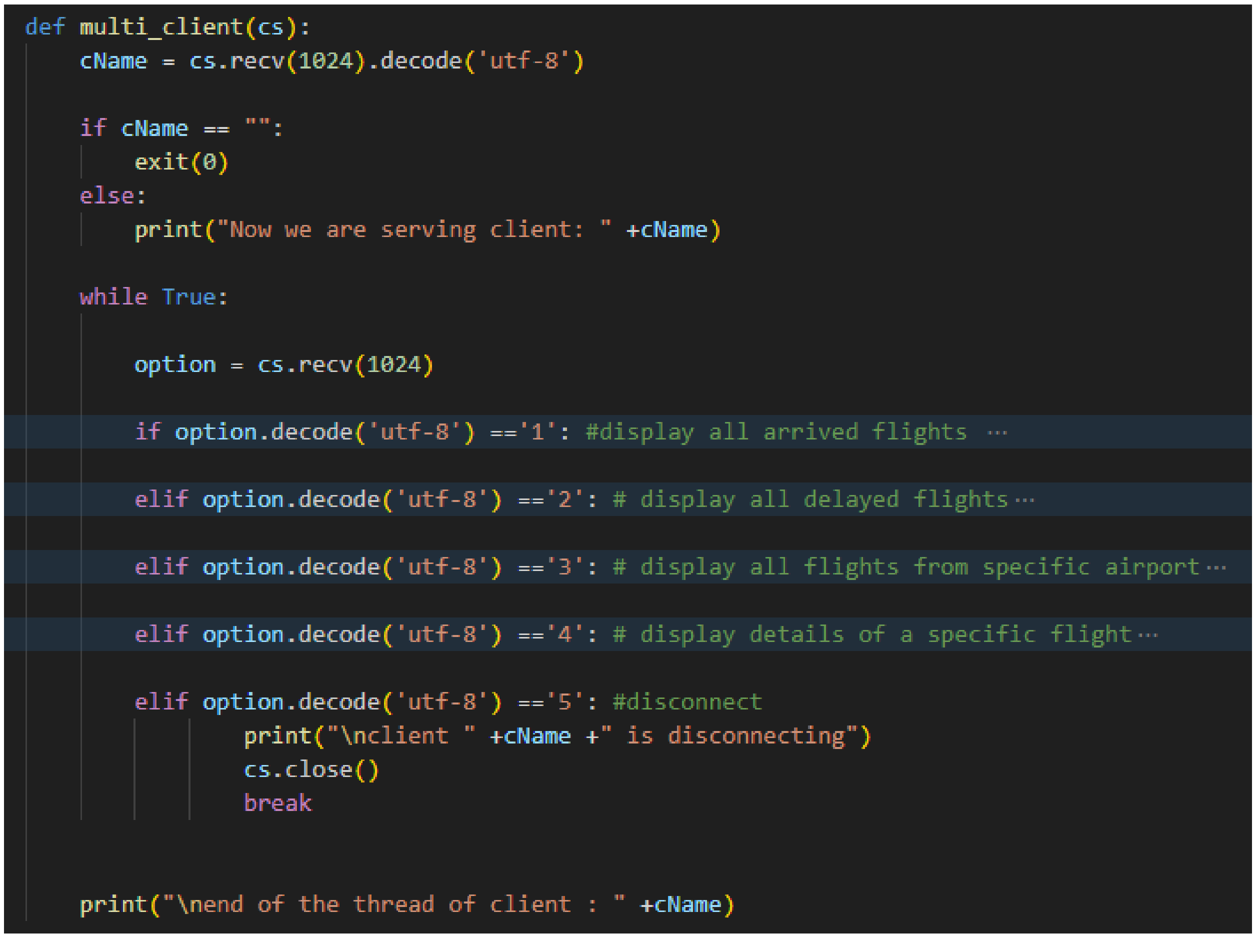
```python
  6  def main():
  7 >    def api(params): ···
 20
 21 >    def multi_client(cs):···
148
149 >    with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as ss: #create a TCP socket ···
190
191  if __name__ == "__main__":
192      main()
```

The first block (line 7) is a function that contains the API setup to connect the server to aviationstack.com with a parameter that the user passes for the required information which will be passed in the 3rd block. We raised an exception when the API connection fails to connect and prompts the user of the error message. Afterwards, we took the API results and parse is from HTTP response via “. json ()” (line 14). Then finally we open a new file named “GA20.json” with writing permission “w” and can be overwritten for more efficient storage performance and dump the data inside it in a JSON format with indentation of 2 via “. dump ()”.



```python
 7      def api(params):
 8
 9          api_result = requests.get("http://api.aviationstack.com/v1/flights",params)# HTTPS IS NOT ACCESSIBLE IN THE FREE SUBSCRIPTION
10
11          if api_result.status_code != 200:
12              raise Exception("Non-200 response: " + str(api_result.text))
13
14          jData = api_result.json()
15
16          with open("GA20.json", "w") as f: #save the results in a json file for testing
17              json.dump(jData, f, indent=2)
```

The second block (line 21) is the main thread body where it allows the server to handle multiple clients concurrently. For each incoming connection, a new thread is spawned, ensuring that the server can respond to client requests independently. This concurrency improves the overall responsiveness of the system. It takes the client socket as a parameter for sending and receiving the required data. It starts with receiving the client's name and display for the user. Then it starts a while loop to ensure that the server remains active and continuously listens for incoming messages from clients. This design choice enables the server to handle a continuous stream of client requests without terminating prematurely.

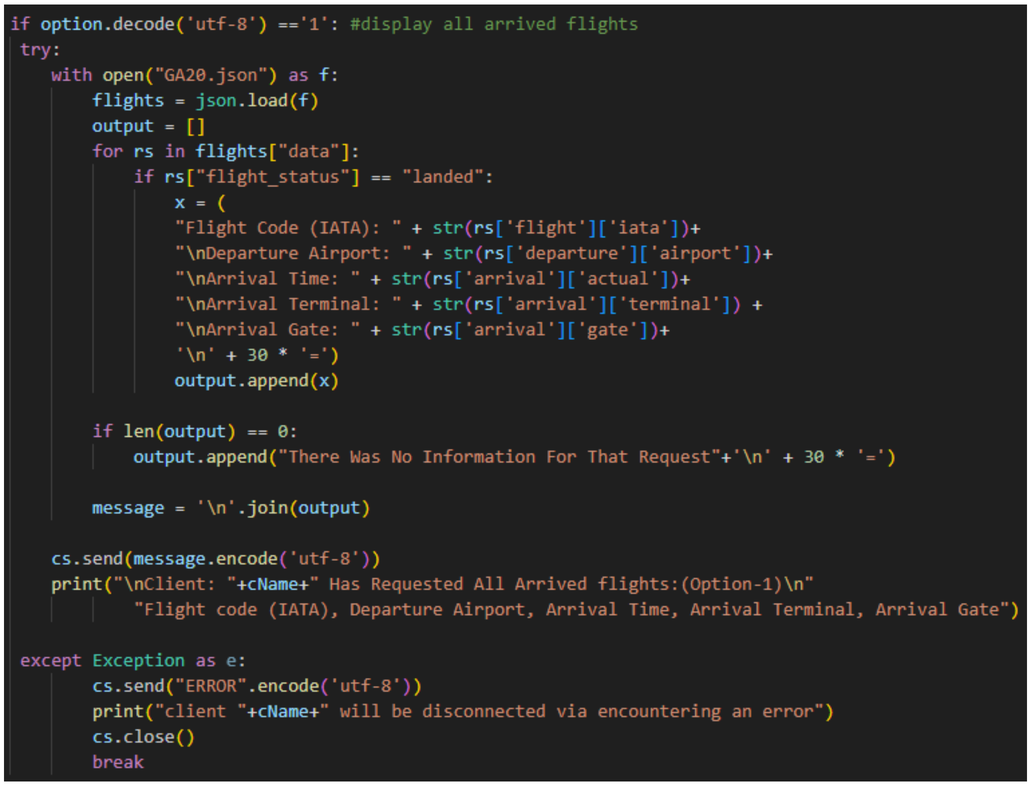


```python
def multi_client(cs):
    cName = cs.recv(1024).decode('utf-8')

    if cName == "":
        exit(0)
    else:
        print("Now we are serving client: " +cName)

    while True:

        option = cs.recv(1024)

        if option.decode('utf-8') =='1': #display all arrived flights ...

        elif option.decode('utf-8') =='2': # display all delayed flights...

        elif option.decode('utf-8') =='3': # display all flights from specific airport...

        elif option.decode('utf-8') =='4': # display details of a specific flight...

        elif option.decode('utf-8') =='5': #disconnect
                print("\nclient " +cName +" is disconnecting")
                cs.close()
                break


    print("\nend of the thread of client : " +cName)
```

The server has four services besides disconnection, where client disconnects by closing the socket connection “**cs.close()**” when a client sends the termination signal **'5'**. The corresponding thread for that client is terminated, allowing the server to maintain responsiveness to other connected clients. All option verification is from the client side.

A. Option 1 – display all arrival flights:

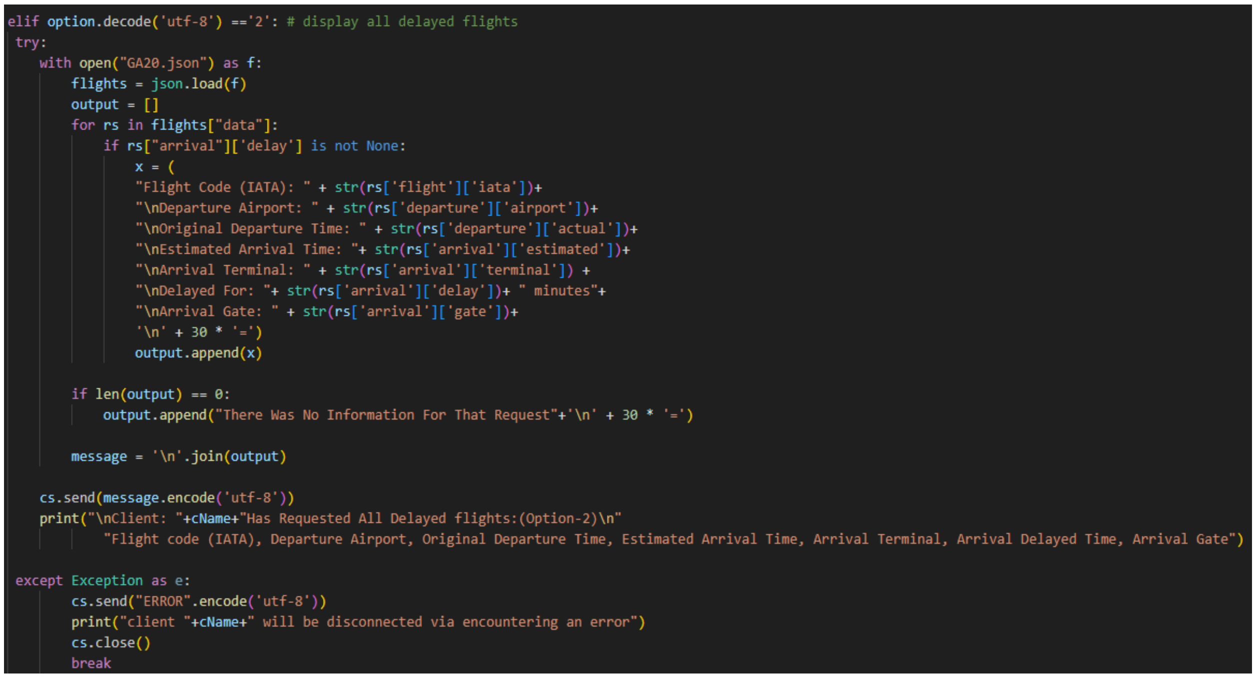
The process begins by opening the "GA20.json" file and using the “json.load(f)” method to deserialize its contents into a Python data structure. Subsequently, a loop is initiated to iterate through each flight's data. Within the loop, an if statement checks whether the flight status is equal to "landed." If the condition is met, relevant information such as the flight code (IATA), departure airport, arrival time, arrival terminal, and arrival gate is formatted and appended to an output list. If there are no matching flights, a default message indicating the absence of information for the request is appended. Finally, this string is sent to the client via the socket connection, and a corresponding message is printed on the server side to notify that the client, identified as “cName”, has requested details about all arrived flights under Option 1.



```python
if option.decode('utf-8') =='1': #display all arrived flights
 try:
    with open("GA20.json") as f:
        flights = json.load(f)
        output = []
        for rs in flights["data"]:
            if rs["flight_status"] == "landed":
                x = (
                "Flight Code (IATA): " + str(rs['flight']['iata'])+
                "\nDeparture Airport: " + str(rs['departure']['airport'])+
                "\nArrival Time: " + str(rs['arrival']['actual'])+
                "\nArrival Terminal: " + str(rs['arrival']['terminal']) +
                "\nArrival Gate: " + str(rs['arrival']['gate'])+
                '\n' + 30 * '=')
                output.append(x)

        if len(output) == 0:
            output.append("There Was No Information For That Request"+'\n' + 30 * '=')

        message = '\n'.join(output)

    cs.send(message.encode('utf-8'))
    print("\nClient: "+cName+" Has Requested All Arrived flights:(Option-1)\n"
            "Flight code (IATA), Departure Airport, Arrival Time, Arrival Terminal, Arrival Gate")

 except Exception as e:
        cs.send("ERROR".encode('utf-8'))
        print("client "+cName+" will be disconnected via encountering an error")
        cs.close()
        break
```

B. Option 2 – display all delayed flights:

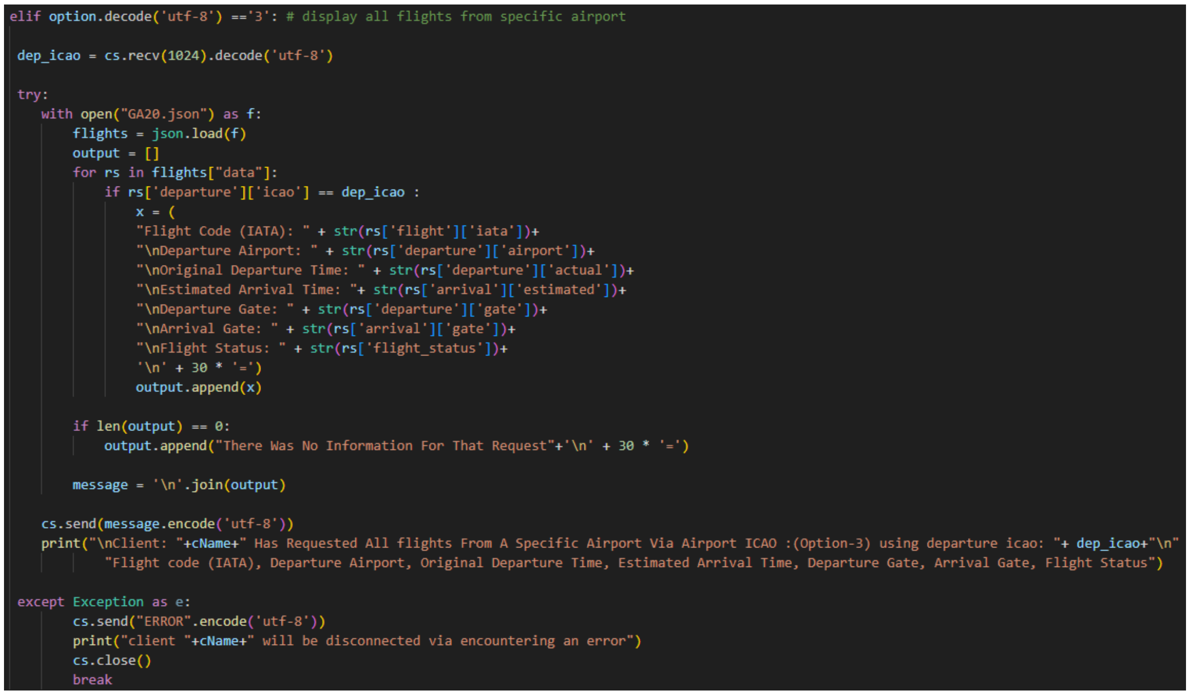
It starts with the same process as option 1, but the if statement checks if the arrival delay is not None. If this condition is met, relevant information, including the flight code (IATA), departure airport, original departure time, estimated arrival time, arrival terminal, delay duration, and arrival gate, is formatted and appended to an output list and continue with the same process.



```
elif option.decode('utf-8') =='2': # display all delayed flights
 try:
    with open("GA20.json") as f:
        flights = json.load(f)
        output = []
        for rs in flights["data"]:
            if rs["arrival"]['delay'] is not None:
                x = (
                "Flight Code (IATA): " + str(rs['flight']['iata'])+
                "\nDeparture Airport: " + str(rs['departure']['airport'])+
                "\nOriginal Departure Time: " + str(rs['departure']['actual'])+
                "\nEstimated Arrival Time: "+ str(rs['arrival']['estimated'])+
                "\nArrival Terminal: " + str(rs['arrival']['terminal']) +
                "\nDelayed For: "+ str(rs['arrival']['delay'])+ " minutes"+
                "\nArrival Gate: " + str(rs['arrival']['gate'])+
                '\n' + 30 * '=')
                output.append(x)

        if len(output) == 0:
            output.append("There Was No Information For That Request"+'\n' + 30 * '=')

        message = '\n'.join(output)

    cs.send(message.encode('utf-8'))
    print("\nClient: "+cName+"Has Requested All Delayed flights:(Option-2)\n"
            "Flight code (IATA), Departure Airport, Original Departure Time, Estimated Arrival Time, Arrival Terminal, Arrival Delayed Time, Arrival Gate")

 except Exception as e:
        cs.send("ERROR".encode('utf-8'))
        print("client "+cName+" will be disconnected via encountering an error")
        cs.close()
        break
```

C. Option 3 – display all arrival flights from specific airport:

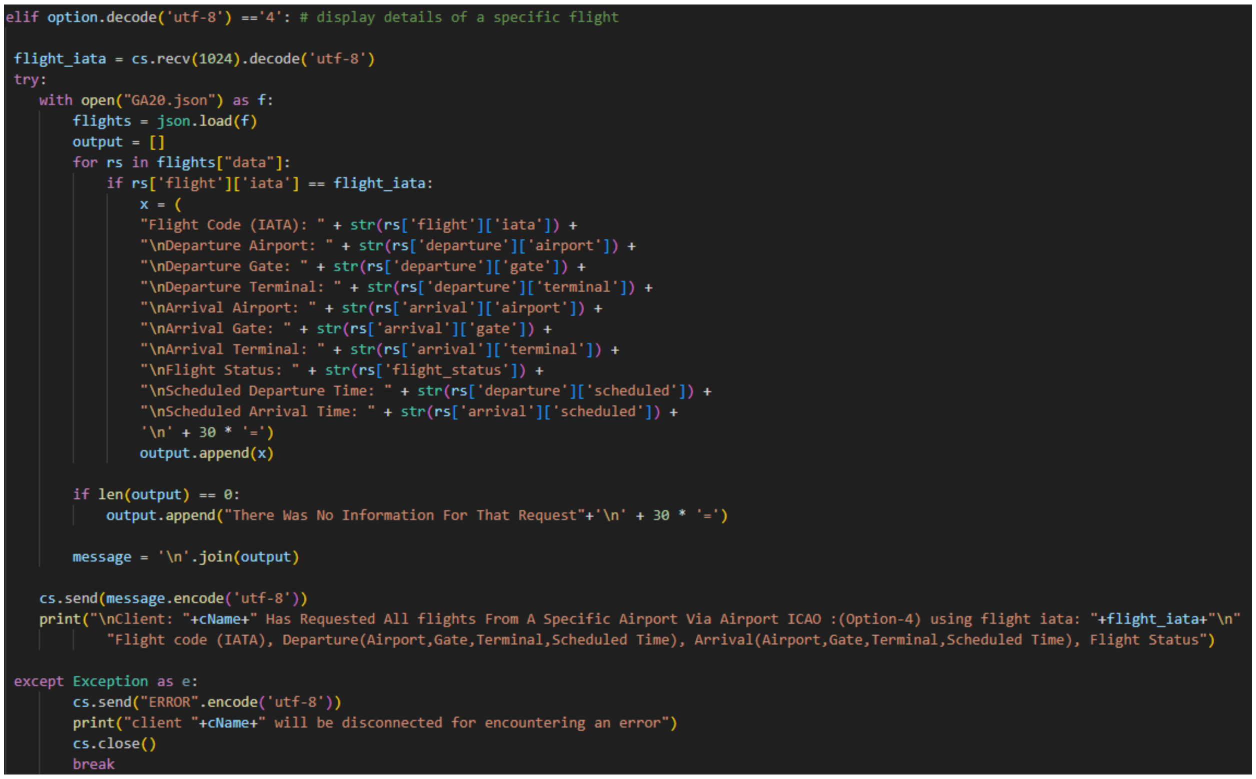
The process by receiving the specific airport's ICAO code through the socket connection using “cs.recv(1024).decode('utf-8')”, then goes with the same process as the others. In the if statement it checks if the departure ICAO matches the received code. If there is a match, relevant information such as the flight code (IATA), departure airport, original departure time, estimated arrival time, departure gate, arrival gate, and flight status is formatted and appended to an output list and continue with the same process with printing the departure ICAO along with in the server side.



```
elif option.decode('utf-8') =='3': # display all flights from specific airport

 dep_icao = cs.recv(1024).decode('utf-8')

 try:
    with open("GA20.json") as f:
        flights = json.load(f)
        output = []
        for rs in flights["data"]:
            if rs['departure']['icao'] == dep_icao :
                x = (
                "Flight Code (IATA): " + str(rs['flight']['iata'])+
                "\nDeparture Airport: " + str(rs['departure']['airport'])+
                "\nOriginal Departure Time: " + str(rs['departure']['actual'])+
                "\nEstimated Arrival Time: "+ str(rs['arrival']['estimated'])+
                "\nDeparture Gate: " + str(rs['departure']['gate'])+
                "\nArrival Gate: " + str(rs['arrival']['gate'])+
                "\nFlight Status: " + str(rs['flight_status'])+
                '\n' + 30 * '=')
                output.append(x)

        if len(output) == 0:
            output.append("There Was No Information For That Request"+'\n' + 30 * '=')

        message = '\n'.join(output)

    cs.send(message.encode('utf-8'))
    print("\nClient: "+cName+" Has Requested All flights From A Specific Airport Via Airport ICAO :(Option-3) using departure icao: "+ dep_icao+"\n"
            "Flight code (IATA), Departure Airport, Original Departure Time, Estimated Arrival Time, Departure Gate, Arrival Gate, Flight Status")

 except Exception as e:
        cs.send("ERROR".encode('utf-8'))
        print("client "+cName+" will be disconnected via encountering an error")
        cs.close()
        break
```

D. Option 4 – display all details of a specific flight:

The process initiates by receiving the flight's IATA code through the socket connection using “cs.recv(1024).decode('utf-8')”, then goes with the same process as the others. In the if statement it checks if the flight's IATA code matches the received code. If there is a match, comprehensive details such as the flight code (IATA), departure airport, departure gate, departure terminal, arrival airport, arrival gate, arrival terminal, flight status, scheduled departure time, and scheduled arrival time are formatted and appended to an output list and continue with the same process with printing the flight IATA along with in the server side.



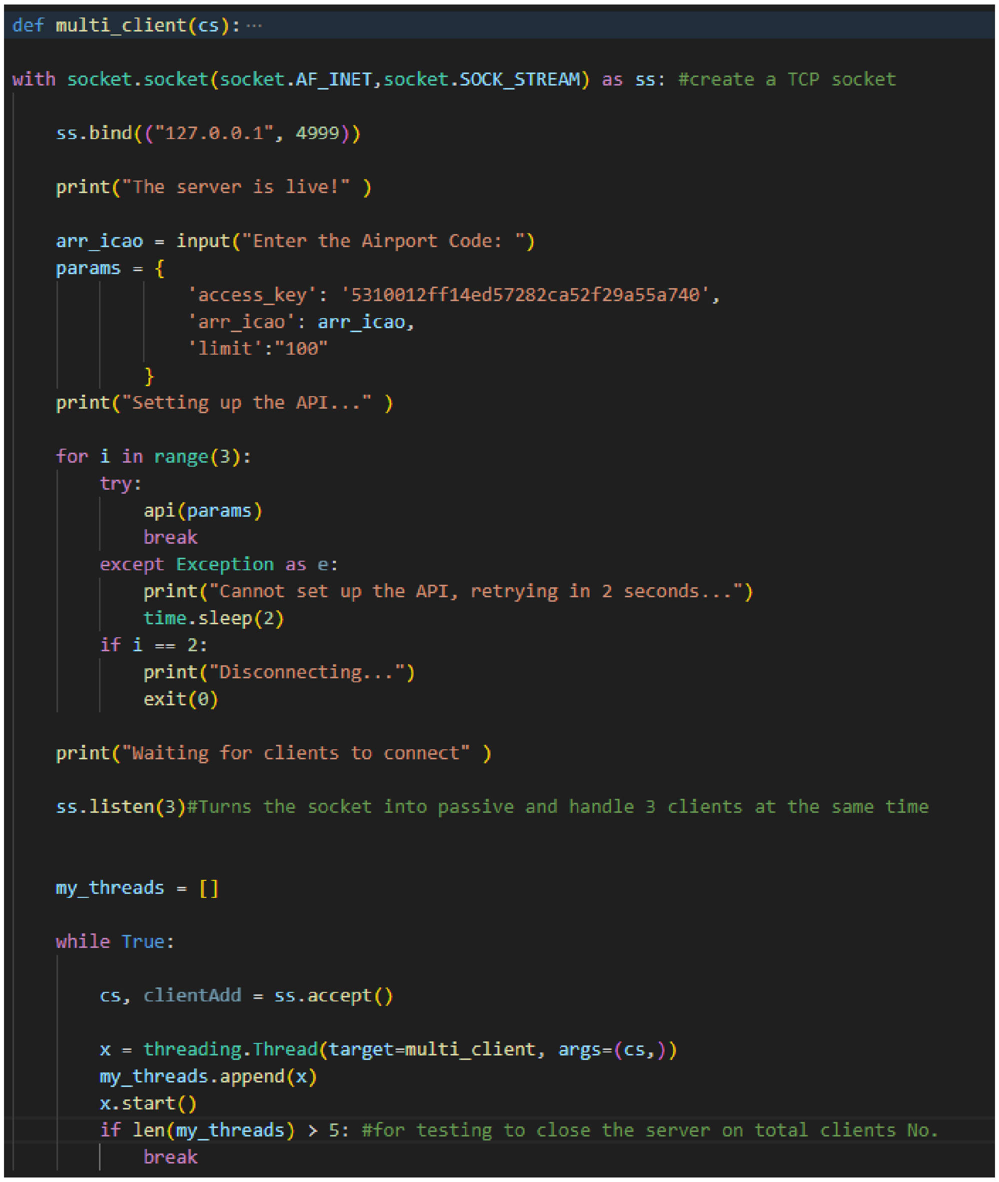
```
elif option.decode('utf-8') =='4': # display details of a specific flight

 flight_iata = cs.recv(1024).decode('utf-8')
 try:
    with open("GA20.json") as f:
        flights = json.load(f)
        output = []
        for rs in flights["data"]:
            if rs['flight']['iata'] == flight_iata:
                x = (
                "Flight Code (IATA): " + str(rs['flight']['iata']) +
                "\nDeparture Airport: " + str(rs['departure']['airport']) +
                "\nDeparture Gate: " + str(rs['departure']['gate']) +
                "\nDeparture Terminal: " + str(rs['departure']['terminal']) +
                "\nArrival Airport: " + str(rs['arrival']['airport']) +
                "\nArrival Gate: " + str(rs['arrival']['gate']) +
                "\nArrival Terminal: " + str(rs['arrival']['terminal']) +
                "\nFlight Status: " + str(rs['flight_status']) +
                "\nScheduled Departure Time: " + str(rs['departure']['scheduled']) +
                "\nScheduled Arrival Time: " + str(rs['arrival']['scheduled']) +
                '\n' + 30 * '=')
                output.append(x)

        if len(output) == 0:
            output.append("There Was No Information For That Request"+'\n' + 30 * '=')

        message = '\n'.join(output)

    cs.send(message.encode('utf-8'))
    print("\nClient: "+cName+" Has Requested All flights From A Specific Airport Via Airport ICAO :(Option-4) using flight iata: "+flight_iata+"\n"
        "Flight code (IATA), Departure(Airport,Gate,Terminal,Scheduled Time), Arrival(Airport,Gate,Terminal,Scheduled Time), Flight Status")

 except Exception as e:
        cs.send("ERROR".encode('utf-8'))
        print("client "+cName+" will be disconnected for encountering an error")
        cs.close()
        break
```

Finally, the last block (line149) is responsible for initializing the server and handling client connections. It begins by creating a TCP socket using **socket.socket(socket.AF_INET, socket.SOCK_STREAM)”** and binding it to the loopback address with “**ss.bind(("127.0.0.1", 4999))**”. The server status message "The server is live!" is then printed.

The user is prompted to enter the airport code “**arr_icao**” for setting up the API. The API parameters, including the access key, airport code, and limit, are defined in the **params** dictionary. The API setup is attempted in a loop with a maximum of three retries. If unsuccessful, the code waits for 2 seconds before retrying. If setup fails after three attempts, the server disconnects.

Once the API is set up successfully, the server enters a waiting state for clients to connect using “**ss.listen(3)”**. The server is passive until a client connection is established.

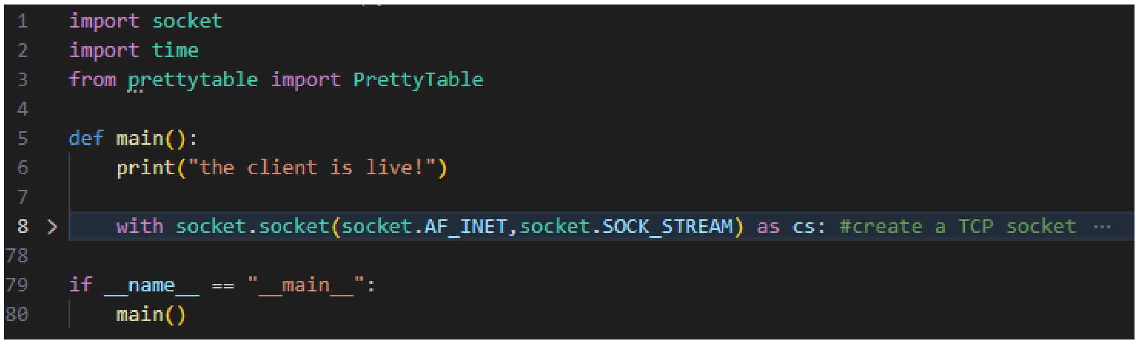
A list (**my_threads**) is created to manage multiple client threads. In an infinite loop, the server waits for incoming client connections with **ss.accept()**. For each connection, a new thread is started using the **multi_client** function as the target. The thread is then appended to the list. For testing purposes, the server can be closed after handling more than five clients, indicated by the condition **if len(my_threads) > 5: break**.



```
def multi_client(cs):···

with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as ss: #create a TCP socket

    ss.bind(("127.0.0.1", 4999))

    print("The server is live!" )

    arr_icao = input("Enter the Airport Code: ")
    params = {
                'access_key': '5310012ff14ed57282ca52f29a55a740',
                'arr_icao': arr_icao,
                'limit':"100"
            }
    print("Setting up the API..." )

    for i in range(3):
        try:
            api(params)
            break
        except Exception as e:
            print("Cannot set up the API, retrying in 2 seconds...")
            time.sleep(2)
        if i == 2:
            print("Disconnecting...")
            exit(0)

    print("Waiting for clients to connect" )

    ss.listen(3)#Turns the socket into passive and handle 3 clients at the same time


    my_threads = []

    while True:

        cs, clientAdd = ss.accept()

        x = threading.Thread(target=multi_client, args=(cs,))
        my_threads.append(x)
        x.start()
        if len(my_threads) > 5: #for testing to close the server on total clients No.
            break
```

2. **Client script: (no GUI – appendix A)**

Firstly, we used some modules to enable the client to function as required:



```python
import socket
import time
from prettytable import PrettyTable

def main():
    print("the client is live!")

    with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as cs: #create a TCP socket ···

if __name__ == "__main__":
    main()
```

This segment of the code manages the client-side operations and user interaction. The **main** function initializes the client, attempting to connect to the server with a TCP socket. It allows three retries with a 5-second delay between attempts. If unsuccessful after three attempts, the client exit.

Upon successful connection, the client sends its name to the server. The client then enters a loop to display a menu using the PrettyTable library, offering options for different queries. The user can choose options ranging from displaying arrived and delayed flights to obtaining details about specific flights or quitting the program.

Depending on the user's choice, the client sends the corresponding option to the server using **cs.send(option.encode('utf-8'))**. For options 3 and 4, additional inputs (departure ICAO or flight IATA) are obtained and sent to the server.

The client receives the server's response using **cs.recv(4096)**. If the server encounters an error, the client prints an error message and exits. Otherwise, it prints the received message.

If the user chooses option 5 to quit, the client sends the option to the server and displays a farewell message before exiting.

Exception handling is implemented to capture errors related to server disconnection while the client is still active. If such an error occurs, the client prints a disconnection message and exits the program.

3. **Additional concepts:**
   - **Handling Concurrent Connection Requests:**

     The server uses threading to handle multiple clients simultaneously.

     Each client connection is assigned to a separate thread.

   - **Dealing with Client Failures:**

     If an error occurs while processing a client's request, the server sends an "ERROR" message to that client and disconnects.

     The client script has exception handling for potential errors, and it exits gracefully if the server encounters a problem.

   - **Dealing with Server Failures:**

     The server attempts to set up an API connection and exits if unsuccessful after three attempts.

     The server continues to serve other clients even if one encounters an error.

Limitations:

Security Concerns:

- The code doesn't implement secure communication between the server and clients (no encryption/authentication).
- The server accepts connections from any IP address, which might be a security risk.

No Persistent Storage:

The server reads flight information from an API and saves it to a file (GA20.json) for testing. However, there's no persistent storage or database.

Conclusion:

Python stands out in ease and resources but considers limitations for performance-critical scenarios.

Pros:

1. **Ease of Development:** Python's simplicity and readability expedite development.
2. **Rich Ecosystem:** Abundance of libraries/frameworks like **socket** and **Celery** aids development.[4]

Cons:

1. **Global Interpreter Lock (GIL):** Limits parallel execution, impacting CPU-bound tasks.[2]
2. **Performance:** Not the fastest, especially for high-performance computing.

An extension to Python that addresses the GIL limitation by providing better support for concurrency and parallelism would be valuable. While efforts like asynchronous programming with **asyncio** and the **concurrent.futures** module exist, a more comprehensive solution that allows for efficient parallel execution without the GIL bottleneck would be beneficial for developing high-performance distributed systems.

References:

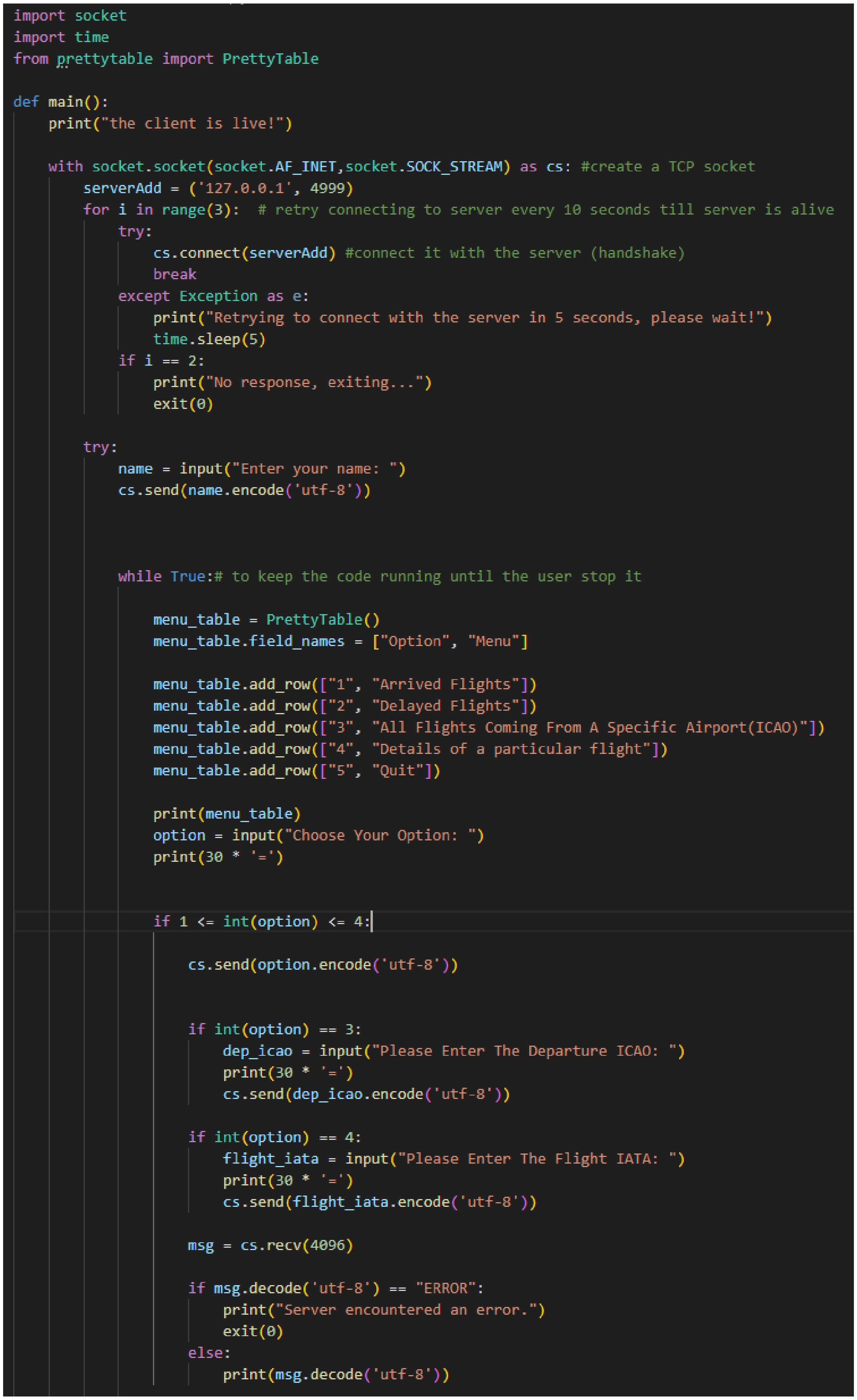
[1] “API Documentation - aviationstack,” *aviationstack.com*. https://aviationstack.com/documentation

[2] Quân Nguyễn, *Mastering concurrency in python : create faster programs using concurrency, asynchronous, multithreading, and parallel programming*. Packt Uuuu-Uuuu.
[3]. A. Almeer, "Network Programming-course slides," Bahrain, 2023”,” pp.14”

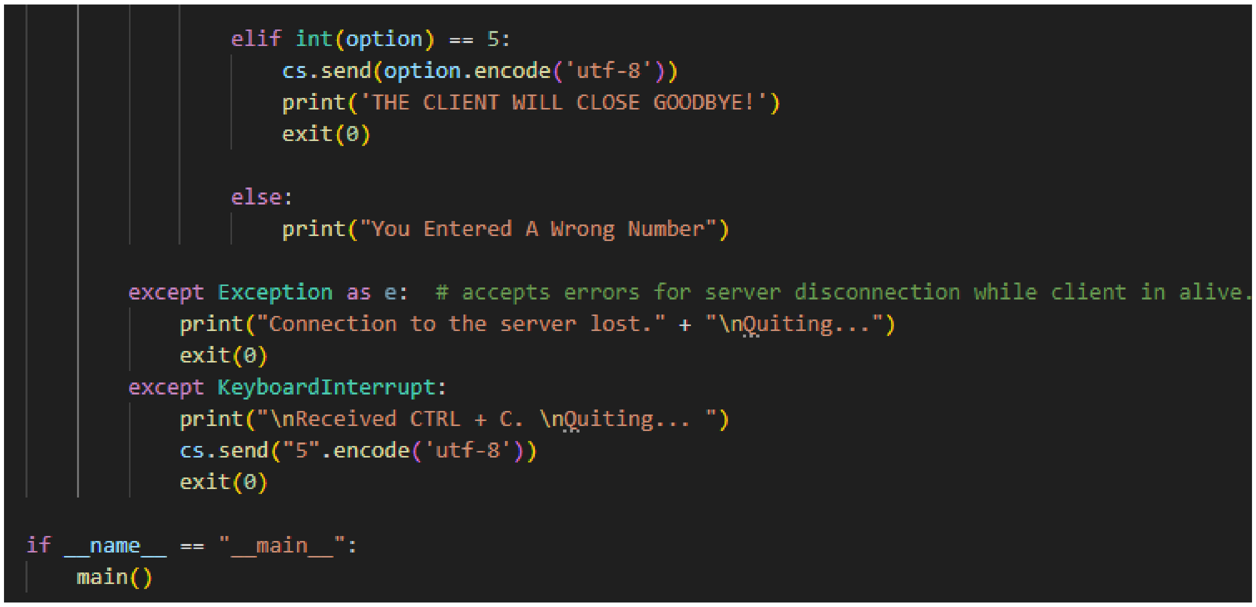
[4]. Li Jun and Li Ling, "Comparative research on Python speed optimization strategies," 2010 International Conference on Intelligent Computing and Integrated Systems, Guilin, 2010, pp. 57-59,

Appendix:

client menu code (Part A):

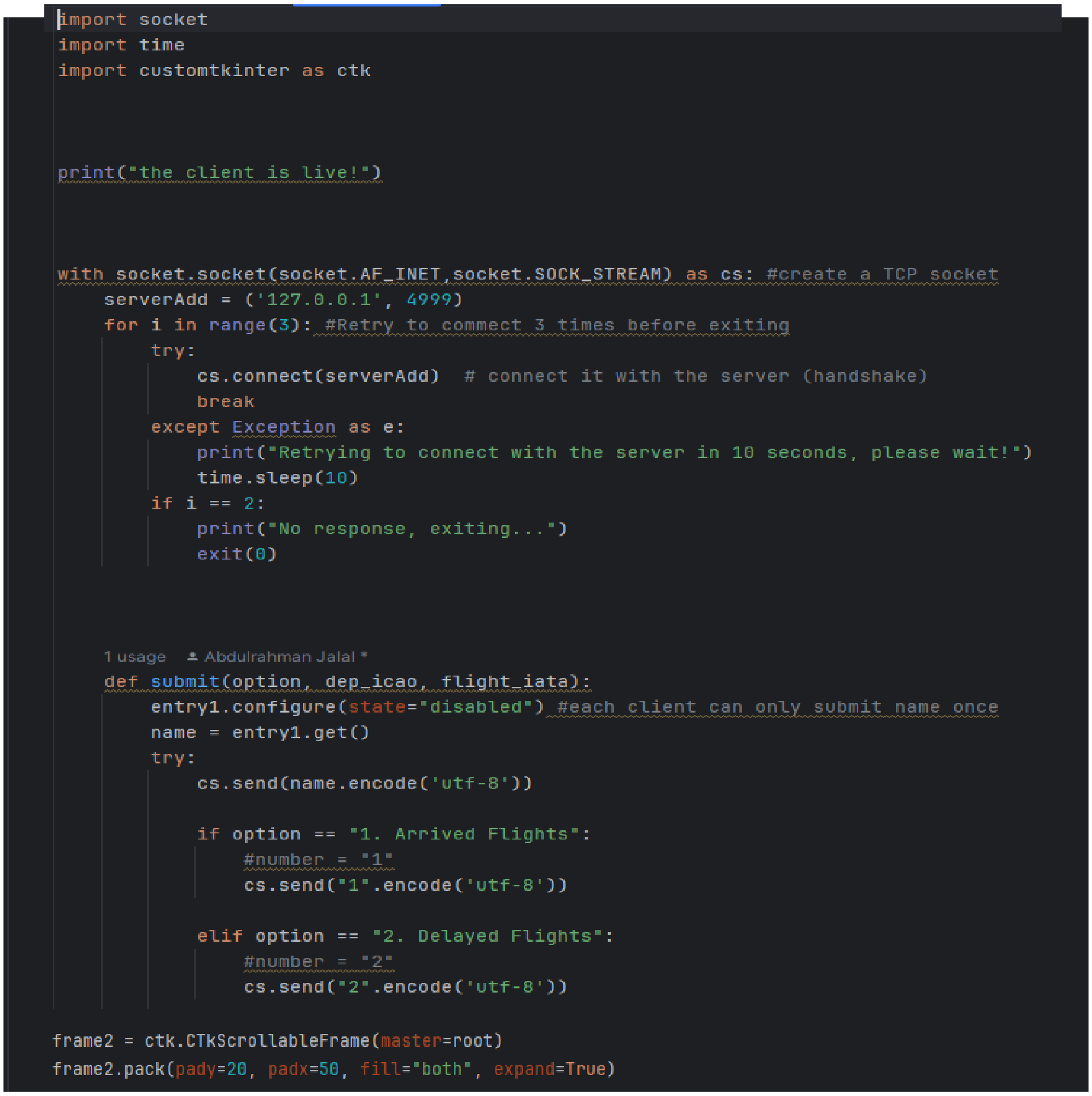


```python
import socket
import time
from prettytable import PrettyTable

def main():
    print("the client is live!")

    with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as cs: #create a TCP socket
        serverAdd = ('127.0.0.1', 4999)
        for i in range(3):  # retry connecting to server every 10 seconds till server is alive
            try:
                cs.connect(serverAdd) #connect it with the server (handshake)
                break
            except Exception as e:
                print("Retrying to connect with the server in 5 seconds, please wait!")
                time.sleep(5)
            if i == 2:
                print("No response, exiting...")
                exit(0)

        try:
            name = input("Enter your name: ")
            cs.send(name.encode('utf-8'))



            while True:# to keep the code running until the user stop it

                menu_table = PrettyTable()
                menu_table.field_names = ["Option", "Menu"]

                menu_table.add_row(["1", "Arrived Flights"])
                menu_table.add_row(["2", "Delayed Flights"])
                menu_table.add_row(["3", "All Flights Coming From A Specific Airport(ICAO)"])
                menu_table.add_row(["4", "Details of a particular flight"])
                menu_table.add_row(["5", "Quit"])

                print(menu_table)
                option = input("Choose Your Option: ")
                print(30 * '=')


                if 1 <= int(option) <= 4:

                    cs.send(option.encode('utf-8'))


                    if int(option) == 3:
                        dep_icao = input("Please Enter The Departure ICAO: ")
                        print(30 * '=')
                        cs.send(dep_icao.encode('utf-8'))

                    if int(option) == 4:
                        flight_iata = input("Please Enter The Flight IATA: ")
                        print(30 * '=')
                        cs.send(flight_iata.encode('utf-8'))

                    msg = cs.recv(4096)

                    if msg.decode('utf-8') == "ERROR":
                        print("Server encountered an error.")
                        exit(0)
                    else:
                        print(msg.decode('utf-8'))
```

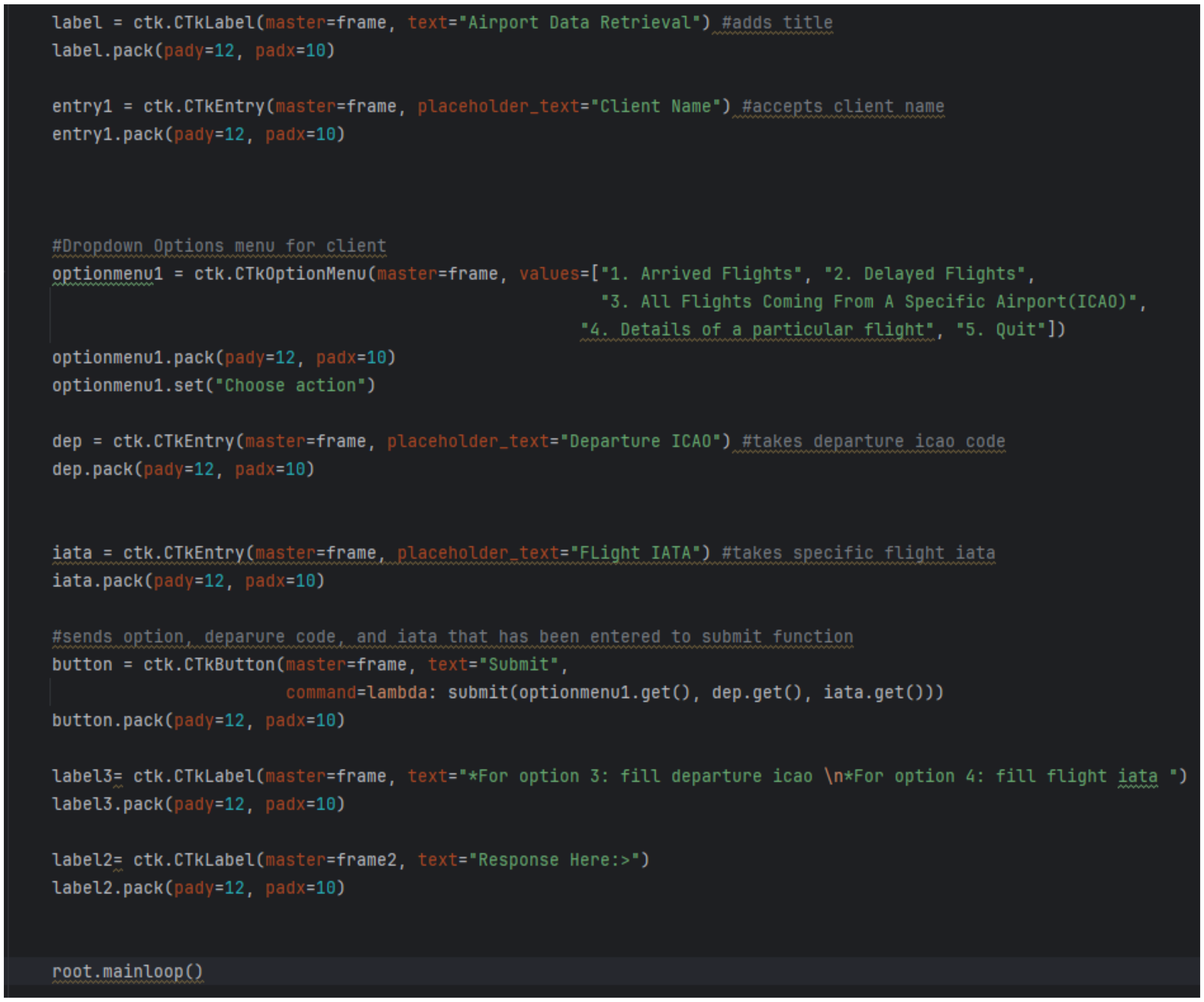


```
                elif int(option) == 5:
                    cs.send(option.encode('utf-8'))
                    print('THE CLIENT WILL CLOSE GOODBYE!')
                    exit(0)

                else:
                    print("You Entered A Wrong Number")

        except Exception as e:  # accepts errors for server disconnection while client in alive.
            print("Connection to the server lost." + "\nQuiting...")
            exit(0)
        except KeyboardInterrupt:
            print("\nReceived CTRL + C. \nQuiting... ")
            cs.send("5".encode('utf-8'))
            exit(0)

if __name__ == "__main__":
    main()
```

Gui code and interface (Part B):

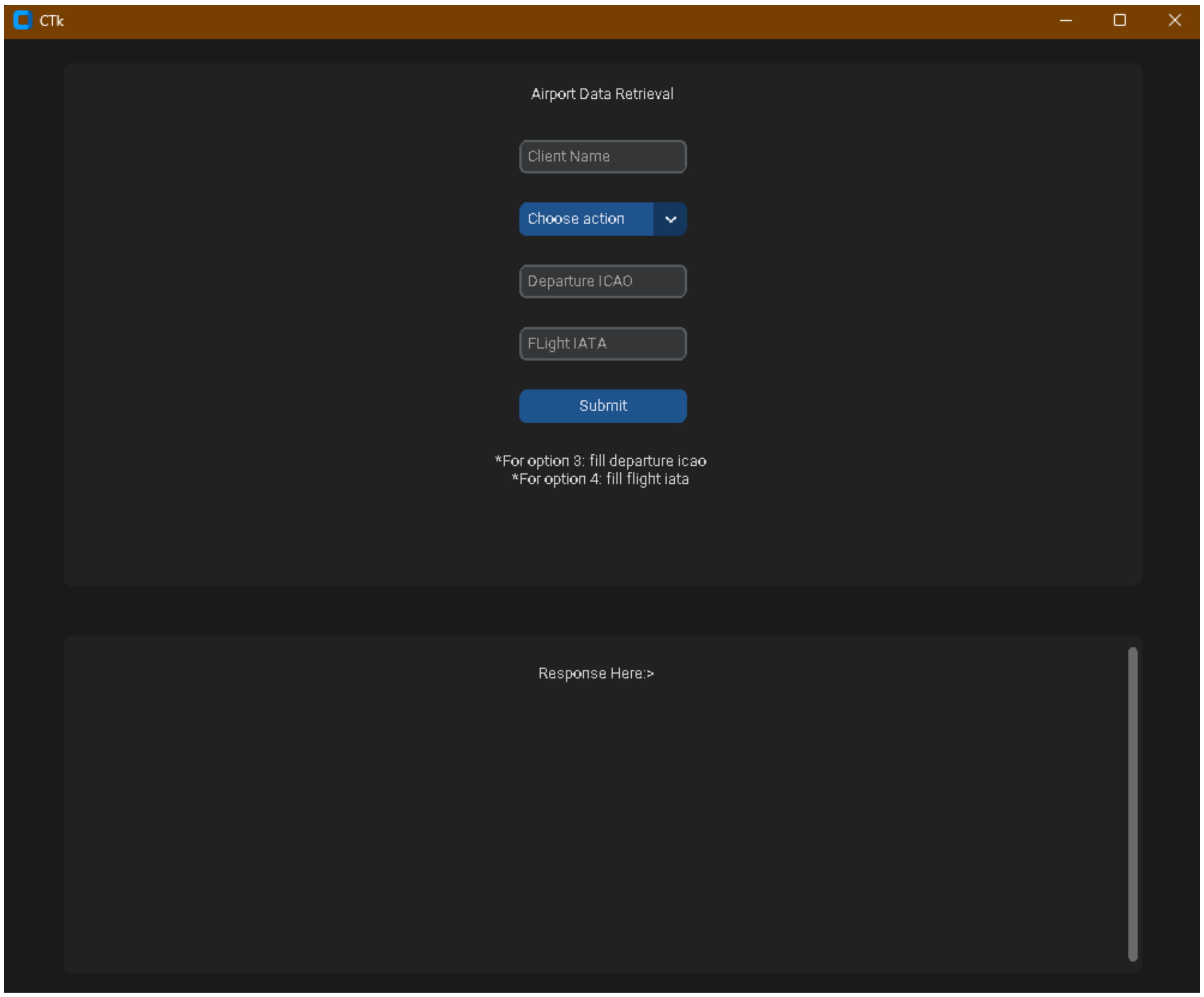


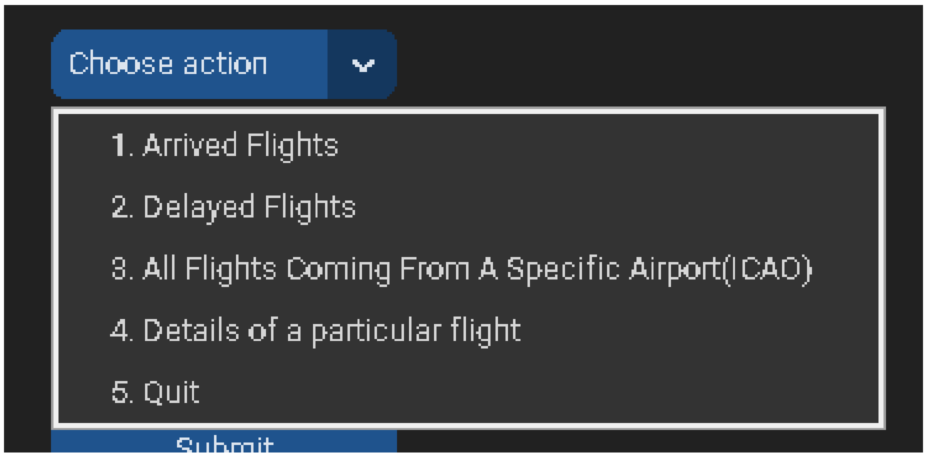
```
import socket
import time
import customtkinter as ctk



print("the client is live!")



with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as cs: #create a TCP socket
    serverAdd = ('127.0.0.1', 4999)
    for i in range(3): #Retry to commect 3 times before exiting
        try:
            cs.connect(serverAdd)  # connect it with the server (handshake)
            break
        except Exception as e:
            print("Retrying to connect with the server in 10 seconds, please wait!")
            time.sleep(10)
        if i == 2:
            print("No response, exiting...")
            exit(0)



    1 usage   Abdulrahman Jalal *
    def submit(option, dep_icao, flight_iata):
        entry1.configure(state="disabled") #each client can only submit name once
        name = entry1.get()
        try:
            cs.send(name.encode('utf-8'))

            if option == "1. Arrived Flights":
                #number = "1"
                cs.send("1".encode('utf-8'))

            elif option == "2. Delayed Flights":
                #number = "2"
                cs.send("2".encode('utf-8'))

frame2 = ctk.CTkScrollableFrame(master=root)
frame2.pack(pady=20, padx=50, fill="both", expand=True)
```



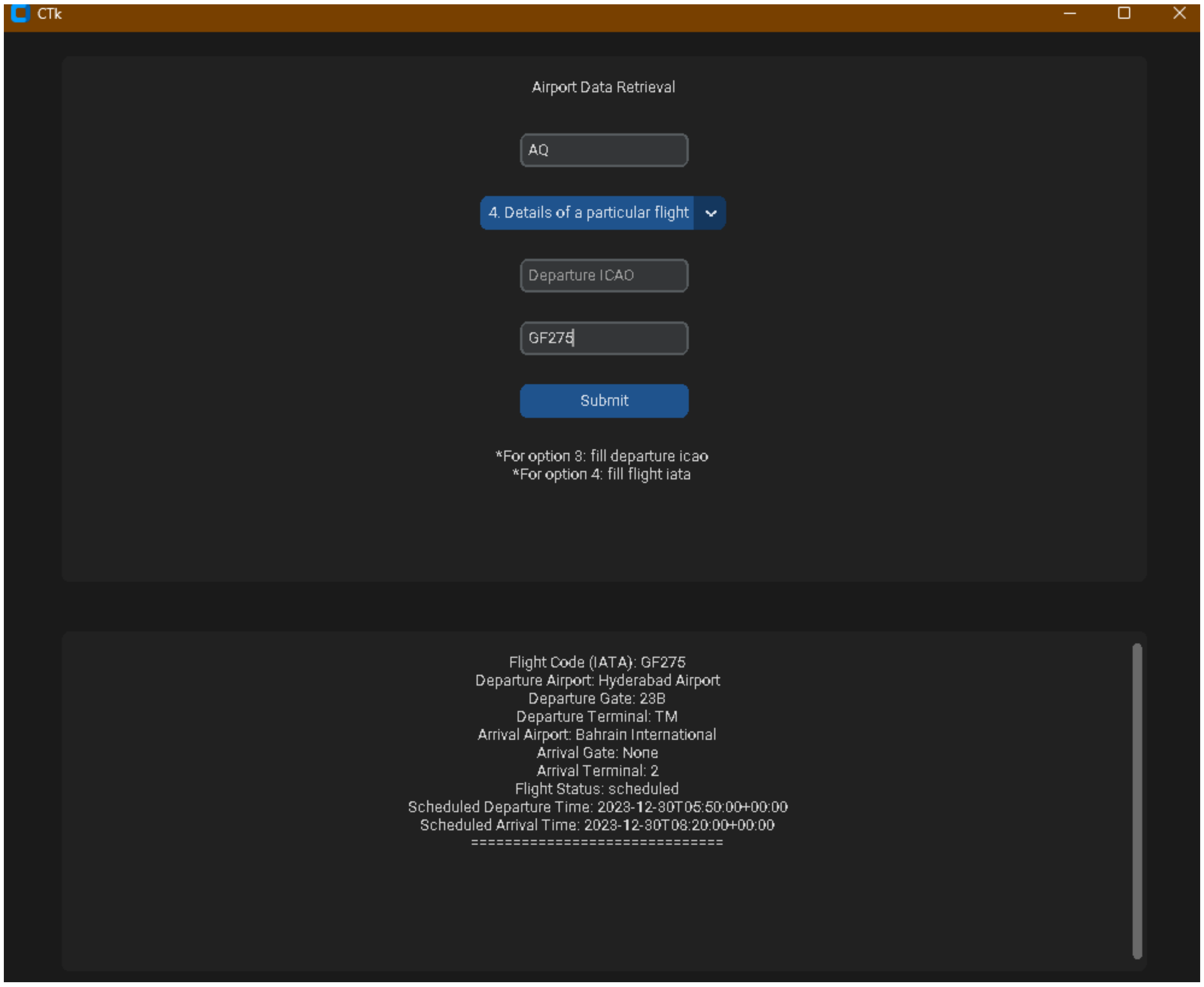
```python
    label = ctk.CTkLabel(master=frame, text="Airport Data Retrieval") #adds title
    label.pack(pady=12, padx=10)

    entry1 = ctk.CTkEntry(master=frame, placeholder_text="Client Name") #accepts client name
    entry1.pack(pady=12, padx=10)



    #Dropdown Options menu for client
    optionmenu1 = ctk.CTkOptionMenu(master=frame, values=["1. Arrived Flights", "2. Delayed Flights",
                                                           "3. All Flights Coming From A Specific Airport(ICAO)",
                                                         "4. Details of a particular flight", "5. Quit"])
    optionmenu1.pack(pady=12, padx=10)
    optionmenu1.set("Choose action")

    dep = ctk.CTkEntry(master=frame, placeholder_text="Departure ICAO") #takes departure icao code
    dep.pack(pady=12, padx=10)


    iata = ctk.CTkEntry(master=frame, placeholder_text="FLight IATA") #takes specific flight iata
    iata.pack(pady=12, padx=10)

    #sends option, deparure code, and iata that has been entered to submit function
    button = ctk.CTkButton(master=frame, text="Submit",
                           command=lambda: submit(optionmenu1.get(), dep.get(), iata.get()))
    button.pack(pady=12, padx=10)

    label3= ctk.CTkLabel(master=frame, text="*For option 3: fill departure icao \n*For option 4: fill flight iata ")
    label3.pack(pady=12, padx=10)

    label2= ctk.CTkLabel(master=frame2, text="Response Here:>")
    label2.pack(pady=12, padx=10)


    root.mainloop()
```

Gui interface:



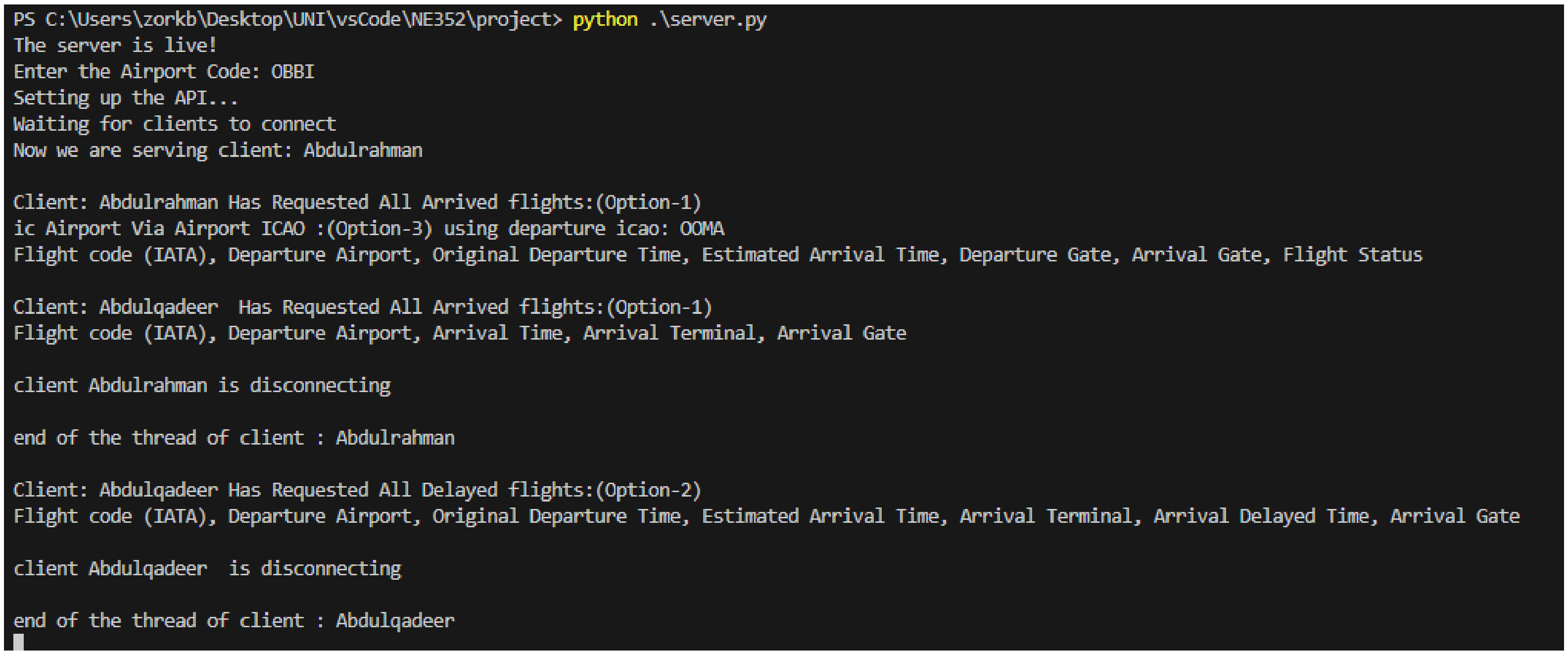

Choose action
1. Arrived Flights
2. Delayed Flights
3. All Flights Coming From A Specific Airport(ICAO)
4. Details of a particular flight
5. Quit
Submit

Run example:


CTk
Airport Data Retrieval
AQ
4. Details of a particular flight
Departure ICAO
GF275
Submit
*For option 3: fill departure icao
*For option 4: fill flight iata

```
Flight Code (IATA): GF275
Departure Airport: Hyderabad Airport
Departure Gate: 23B
Departure Terminal: TM
Arrival Airport: Bahrain International
Arrival Gate: None
Arrival Terminal: 2
Flight Status: scheduled
Scheduled Departure Time: 2023-12-30T05:50:00+00:00
Scheduled Arrival Time: 2023-12-30T08:20:00+00:00
==============================
```

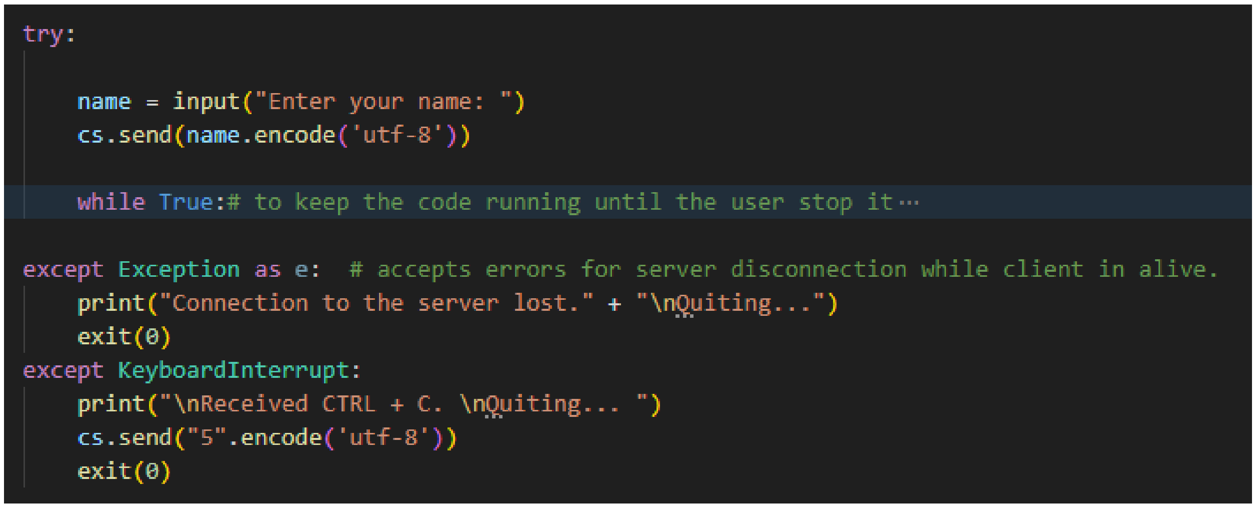
Server outputs (Part B-2):



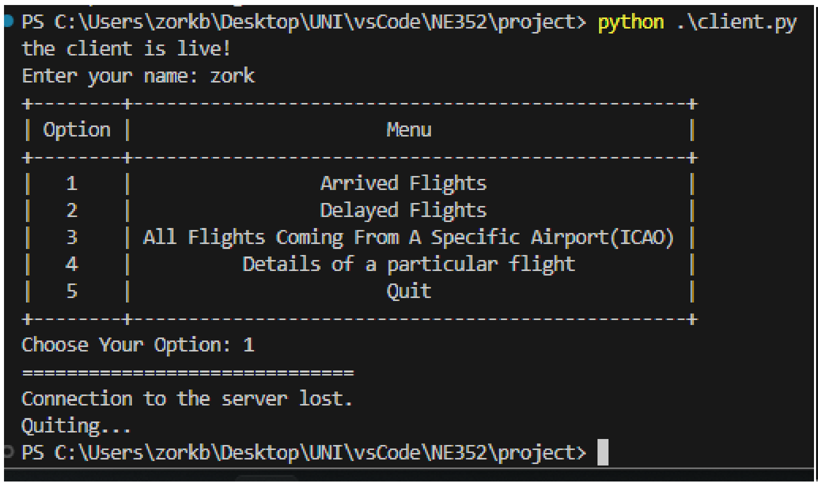
```
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project> python .\server.py
The server is live!
Enter the Airport Code: OBBI
Setting up the API...
Waiting for clients to connect
Now we are serving client: Abdulrahman

Client: Abdulrahman Has Requested All Arrived flights:(Option-1)
ic Airport Via Airport ICAO :(Option-3) using departure icao: OOMA
Flight code (IATA), Departure Airport, Original Departure Time, Estimated Arrival Time, Departure Gate, Arrival Gate, Flight Status

Client: Abdulqadeer  Has Requested All Arrived flights:(Option-1)
Flight code (IATA), Departure Airport, Arrival Time, Arrival Terminal, Arrival Gate

client Abdulrahman is disconnecting

end of the thread of client : Abdulrahman

Client: Abdulqadeer Has Requested All Delayed flights:(Option-2)
Flight code (IATA), Departure Airport, Original Departure Time, Estimated Arrival Time, Arrival Terminal, Arrival Delayed Time, Arrival Gate

client Abdulqadeer  is disconnecting

end of the thread of client : Abdulqadeer
```

try and catch exceptions (Part C):

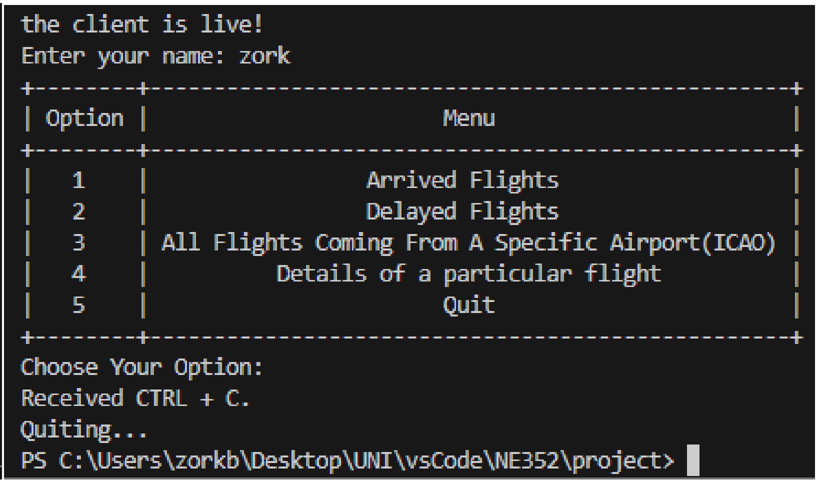
client-side:



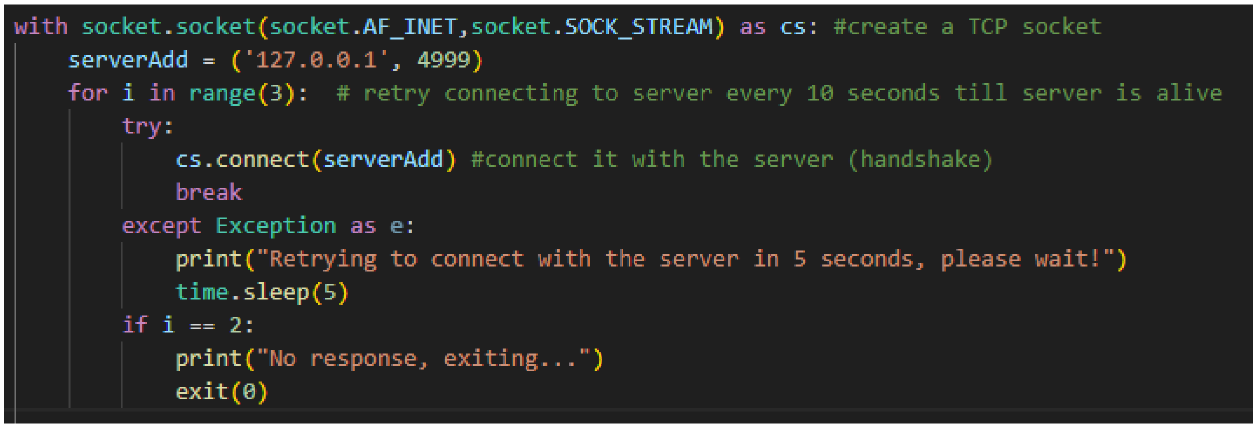
```python
try:

    name = input("Enter your name: ")
    cs.send(name.encode('utf-8'))

    while True:# to keep the code running until the user stop it···

except Exception as e:  # accepts errors for server disconnection while client in alive.
    print("Connection to the server lost." + "\nQuiting...")
    exit(0)
except KeyboardInterrupt:
    print("\nReceived CTRL + C. \nQuiting... ")
    cs.send("5".encode('utf-8'))
    exit(0)
```



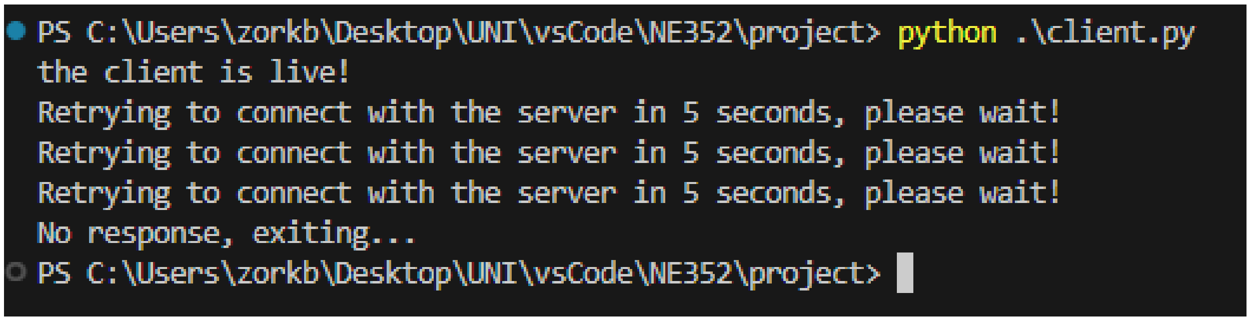
```
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project> python .\client.py
the client is live!
Enter your name: zork
+--------+---------------------------------------------------+
| Option |                        Menu                       |
+--------+---------------------------------------------------+
|   1    |                 Arrived Flights                   |
|   2    |                 Delayed Flights                   |
|   3    | All Flights Coming From A Specific Airport(ICAO)  |
|   4    |            Details of a particular flight         |
|   5    |                       Quit                        |
+--------+---------------------------------------------------+
Choose Your Option: 1
==============================
Connection to the server lost.
Quiting...
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project>
```



```
the client is live!
Enter your name: zork
+--------+---------------------------------------------------+
| Option |                        Menu                       |
+--------+---------------------------------------------------+
|   1    |                 Arrived Flights                   |
|   2    |                 Delayed Flights                   |
|   3    | All Flights Coming From A Specific Airport(ICAO)  |
|   4    |            Details of a particular flight         |
|   5    |                       Quit                        |
+--------+---------------------------------------------------+
Choose Your Option:
Received CTRL + C.
Quiting...
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project>
```

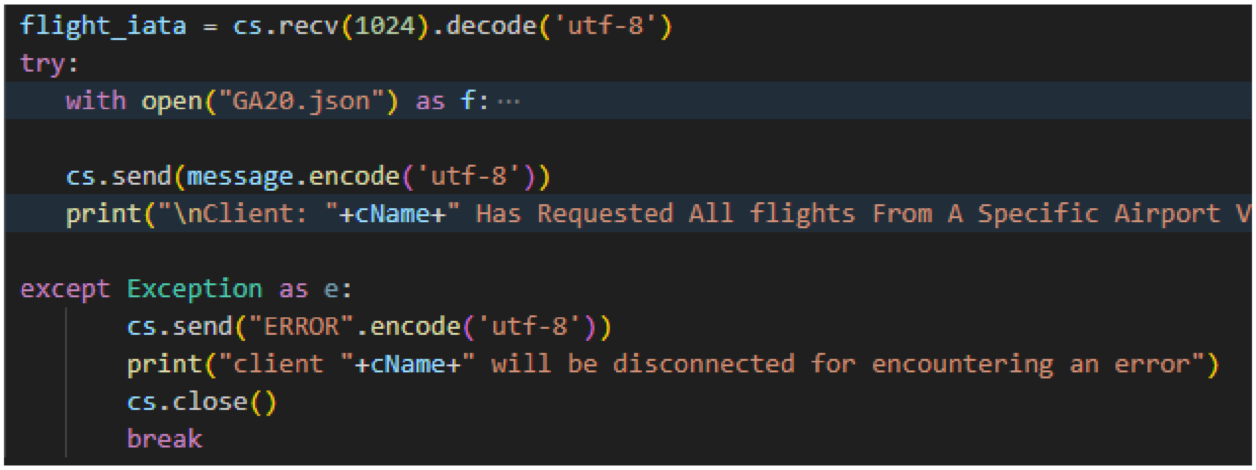


```python
with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as cs: #create a TCP socket
    serverAdd = ('127.0.0.1', 4999)
    for i in range(3):  # retry connecting to server every 10 seconds till server is alive
        try:
            cs.connect(serverAdd) #connect it with the server (handshake)
            break
        except Exception as e:
            print("Retrying to connect with the server in 5 seconds, please wait!")
            time.sleep(5)
        if i == 2:
            print("No response, exiting...")
            exit(0)
```

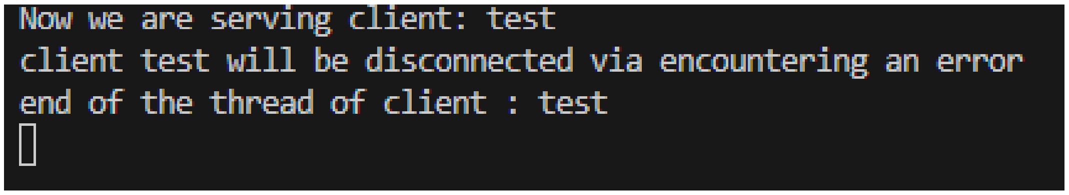


```
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project> python .\client.py
the client is live!
Retrying to connect with the server in 5 seconds, please wait!
Retrying to connect with the server in 5 seconds, please wait!
Retrying to connect with the server in 5 seconds, please wait!
No response, exiting...
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project>
```

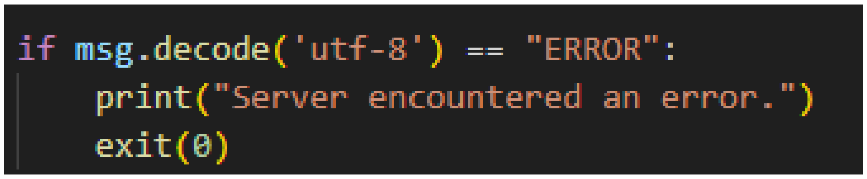
Server-side:



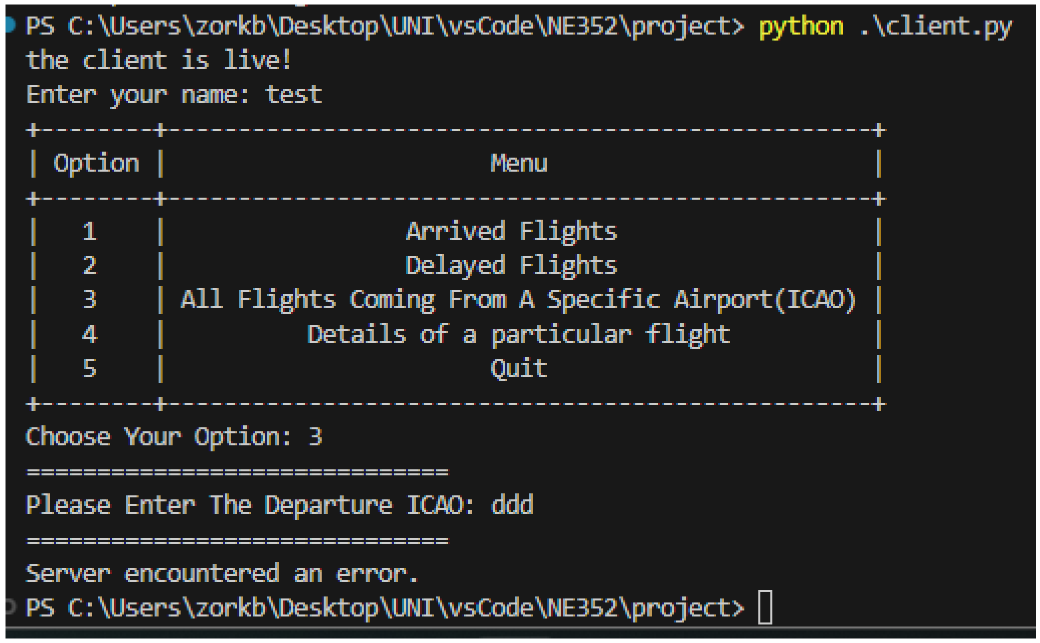
```
flight_iata = cs.recv(1024).decode('utf-8')
try:
    with open("GA20.json") as f:⋯

    cs.send(message.encode('utf-8'))
    print("\nClient: "+cName+" Has Requested All flights From A Specific Airport V

except Exception as e:
        cs.send("ERROR".encode('utf-8'))
        print("client "+cName+" will be disconnected for encountering an error")
        cs.close()
        break
```



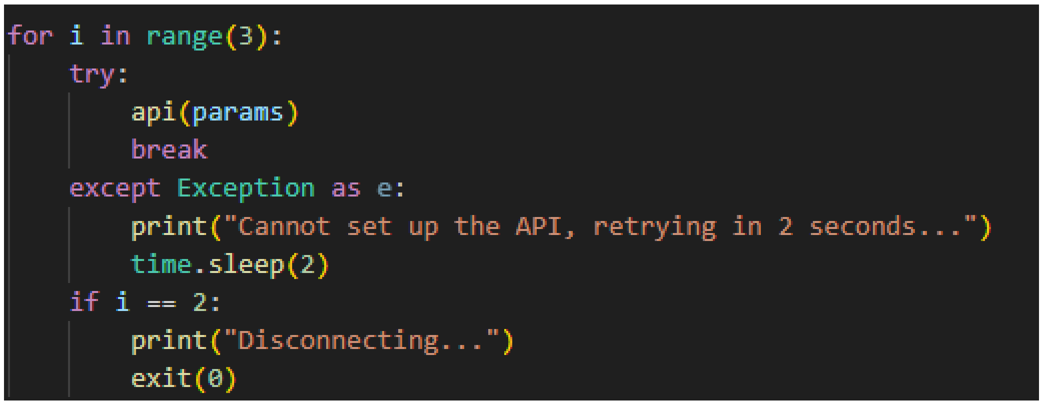
```
Now we are serving client: test
client test will be disconnected via encountering an error
end of the thread of client : test
```



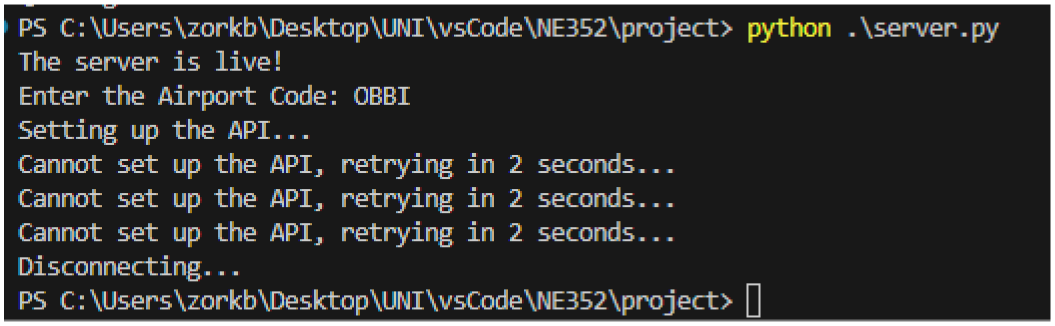
```
if msg.decode('utf-8') == "ERROR":
    print("Server encountered an error.")
    exit(0)
```



```
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project> python .\client.py
the client is live!
Enter your name: test
+--------+---------------------------------------------------+
| Option |                        Menu                       |
+--------+---------------------------------------------------+
|   1    |                  Arrived Flights                  |
|   2    |                  Delayed Flights                  |
|   3    | All Flights Coming From A Specific Airport(ICAO) |
|   4    |             Details of a particular flight        |
|   5    |                        Quit                       |
+--------+---------------------------------------------------+
Choose Your Option: 3
==============================
Please Enter The Departure ICAO: ddd
==============================
Server encountered an error.
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project>
```



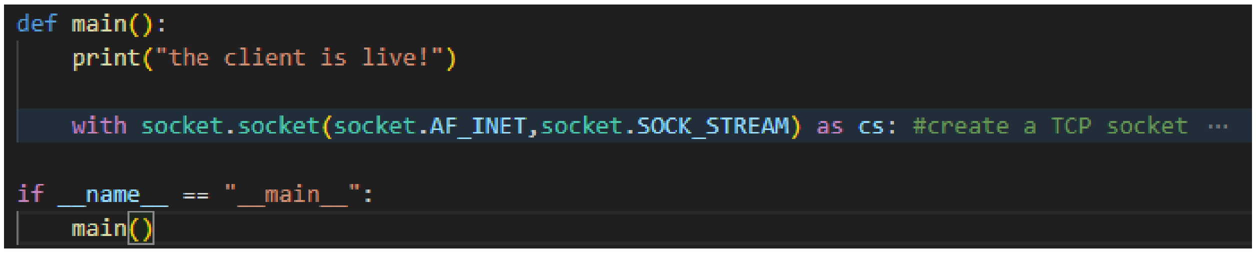
```python
for i in range(3):
    try:
        api(params)
        break
    except Exception as e:
        print("Cannot set up the API, retrying in 2 seconds...")
        time.sleep(2)
    if i == 2:
        print("Disconnecting...")
        exit(0)
```



```
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project> python .\server.py
The server is live!
Enter the Airport Code: OBBI
Setting up the API...
Cannot set up the API, retrying in 2 seconds...
Cannot set up the API, retrying in 2 seconds...
Cannot set up the API, retrying in 2 seconds...
Disconnecting...
PS C:\Users\zorkb\Desktop\UNI\vsCode\NE352\project>
```

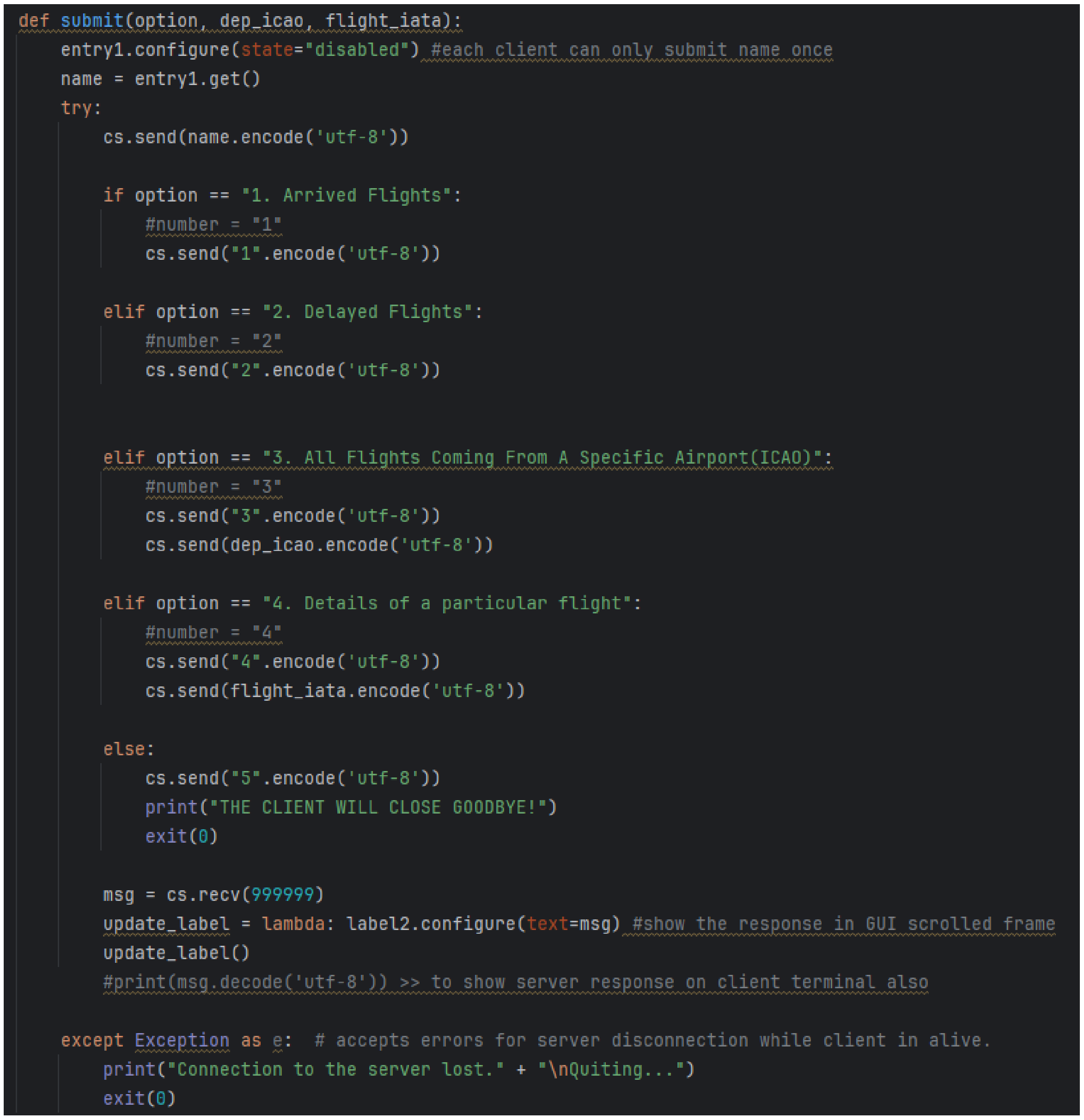
functions used on server/client side (Part D):

Client-side:



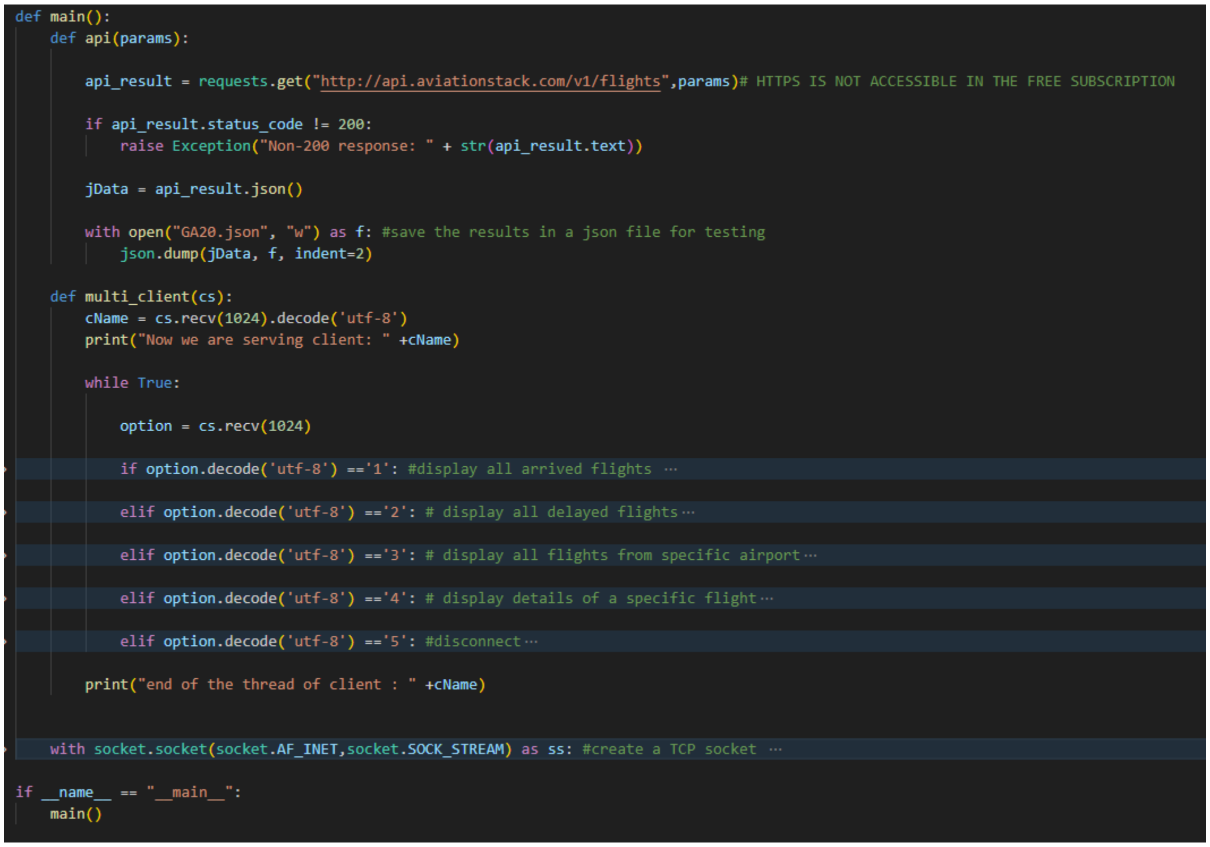
```python
def main():
    print("the client is live!")

    with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as cs: #create a TCP socket ...

if __name__ == "__main__":
    main()
```

Client-side(GUI):



```python
def submit(option, dep_icao, flight_iata):
    entry1.configure(state="disabled") #each client can only submit name once
    name = entry1.get()
    try:
        cs.send(name.encode('utf-8'))

        if option == "1. Arrived Flights":
            #number = "1"
            cs.send("1".encode('utf-8'))

        elif option == "2. Delayed Flights":
            #number = "2"
            cs.send("2".encode('utf-8'))


        elif option == "3. All Flights Coming From A Specific Airport(ICAO)":
            #number = "3"
            cs.send("3".encode('utf-8'))
            cs.send(dep_icao.encode('utf-8'))

        elif option == "4. Details of a particular flight":
            #number = "4"
            cs.send("4".encode('utf-8'))
            cs.send(flight_iata.encode('utf-8'))

        else:
            cs.send("5".encode('utf-8'))
            print("THE CLIENT WILL CLOSE GOODBYE!")
            exit(0)

        msg = cs.recv(999999)
        update_label = lambda: label2.configure(text=msg) #show the response in GUI scrolled frame
        update_label()
        #print(msg.decode('utf-8')) >> to show server response on client terminal also

    except Exception as e:  # accepts errors for server disconnection while client in alive.
        print("Connection to the server lost." + "\nQuiting...")
        exit(0)
```

Server-side:



```
def main():
    def api(params):

        api_result = requests.get("http://api.aviationstack.com/v1/flights",params)# HTTPS IS NOT ACCESSIBLE IN THE FREE SUBSCRIPTION

        if api_result.status_code != 200:
            raise Exception("Non-200 response: " + str(api_result.text))

        jData = api_result.json()

        with open("GA20.json", "w") as f: #save the results in a json file for testing
            json.dump(jData, f, indent=2)

    def multi_client(cs):
        cName = cs.recv(1024).decode('utf-8')
        print("Now we are serving client: " +cName)

        while True:

            option = cs.recv(1024)

            if option.decode('utf-8') =='1': #display all arrived flights ···

            elif option.decode('utf-8') =='2': # display all delayed flights···

            elif option.decode('utf-8') =='3': # display all flights from specific airport···

            elif option.decode('utf-8') =='4': # display details of a specific flight···

            elif option.decode('utf-8') =='5': #disconnect···

        print("end of the thread of client : " +cName)


    with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as ss: #create a TCP socket ···

if __name__ == "__main__":
    main()
```