PA1_Team 14 Project Report

Anirudh Atrey, Mudit Arya, Prateek Abbi, Syed Faizan Ali

(59919994,32141614, 89387132, 26592828,)

aniruddh.atrey@ufl.edu ,arya.mudit@ufl.edu, prateekabbi@ufl.edu, syed.ali1@ufl.edu

-To compile and run the code, you have to run one command only: “dotnet run”.

-After running this command, the terminal will ask you to enter a port number. You can enter any port number (preferred is 9000, 3000, 8000 etc). After entering the port number, the system will ask you to run client or server. If server is not running, then you have to enter server, else the program will give error. If your server is running, then you can enter client.
After running client and server, you can start executing the commands.

The valid command is: <operator> <operand_1> <operand_2>.

-For the project, only 3 operators are valid: add, subtract and multiply. Your command should have atleast 2 operands, and you can add upto 4 operands.

Our project contains 3 files: Program.fs, Client.fs, Server.fs

1) **Program.fs**: This is the entry point for the project. First this file is executed and from this file only client or server functions are called.
2) **Client.fs:** This is the file where client is making connection with server. The default IP address is 127.0.0.1 and port is the port which you will enter in the terminal. There are 2 special conditions where user inputs either bye or terminate command. As per the problem statement, if the user input is “bye”, then server won’t shut down but that user will leave and for the “terminate”, server and client, both will be stopped.
   For every other command, we have 4 special cases:
   a) If the operands are less than 2, then server will return -2 and since we can not show the same to user, so using the if condition, we have printed that number of inputs are less than 2.
   b) If the operands are more than 4, then server will return -3 and client will get the response that number of inputs are more than 4.
   c) If the operator is other than “add”, “subtract”, “multiply”, then server will return with -1 code and user will get the output that operation entered is invalid.

3) **Server.fs:**
   The code starts by defining the "Server" module and importing necessary .NET namespaces, including System, System.Net, System.Net.Sockets, and System.Text
   We keep track of clients by implementing mutable variable **clients.** The function **operation** takes a command (e.g., "add," "subtract," "multiply") and a list of numbers as input and performing these operations. We also applied error handling, by returning specific codes{-1,-2,-3,-4} on unknown and invalid commands. The **serverMain** functions takes port number as a parameter which sets up TCP listener on the specified port and enters an infinite loop to accept incoming client connections.

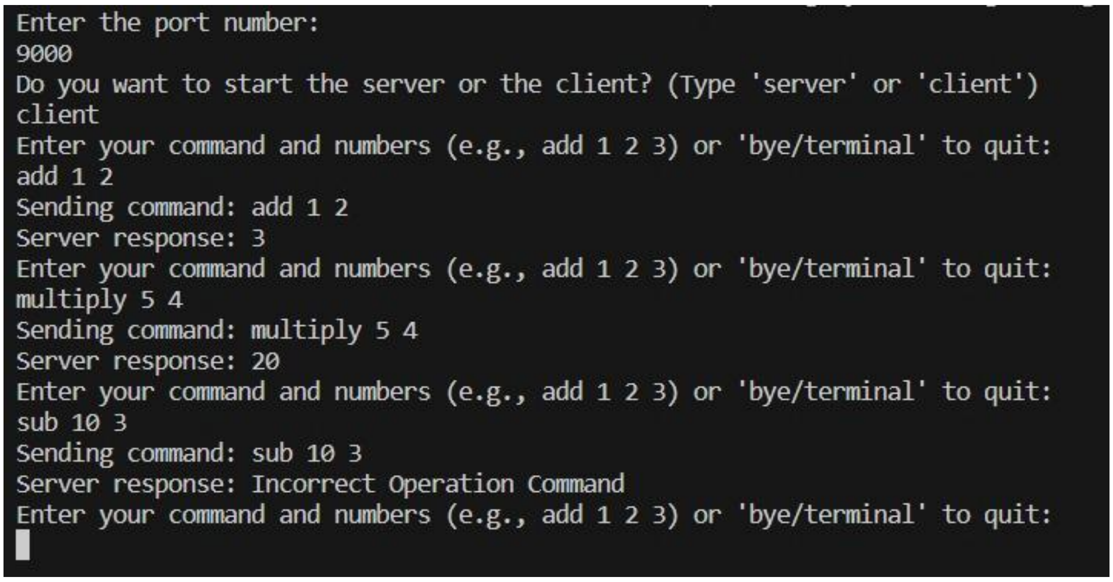
   We talked about critical component of server.fs. Now talking about the multiple steps of Client Communication Process-
   -The server takes a client connection inside the loop, reads the client's message, and separates it into a command and a list of integers.

-When the command is "terminate," the server answers with a -5, cuts off the client connection, and ceases to operate.
-If the client sends the command "bye," the server simply cuts off the connection.
-For other commands, it uses the operations function to calculate the outcome before sending the ASCII-encoded message back to the client.

Output Screenshots-

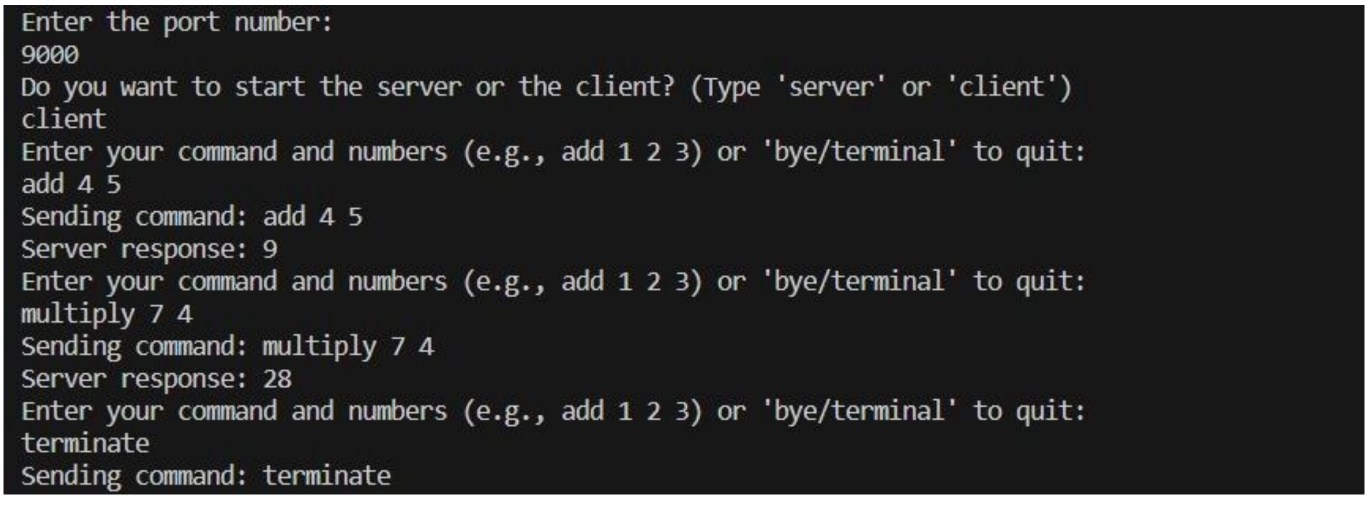
1) The first screenshot is from client side where we input port number as 9000.

Then we sent **add 1 2** command and as you can see server responded with 3. We gave another command and it was executed correctly. The next statement we intentionally typed in **sub 10 3**, but in the code we defined **subtract** to be a valid command. Since it is not a acceptable command ,server throws an exception. So this shows that ,we handled errors flawlessly.



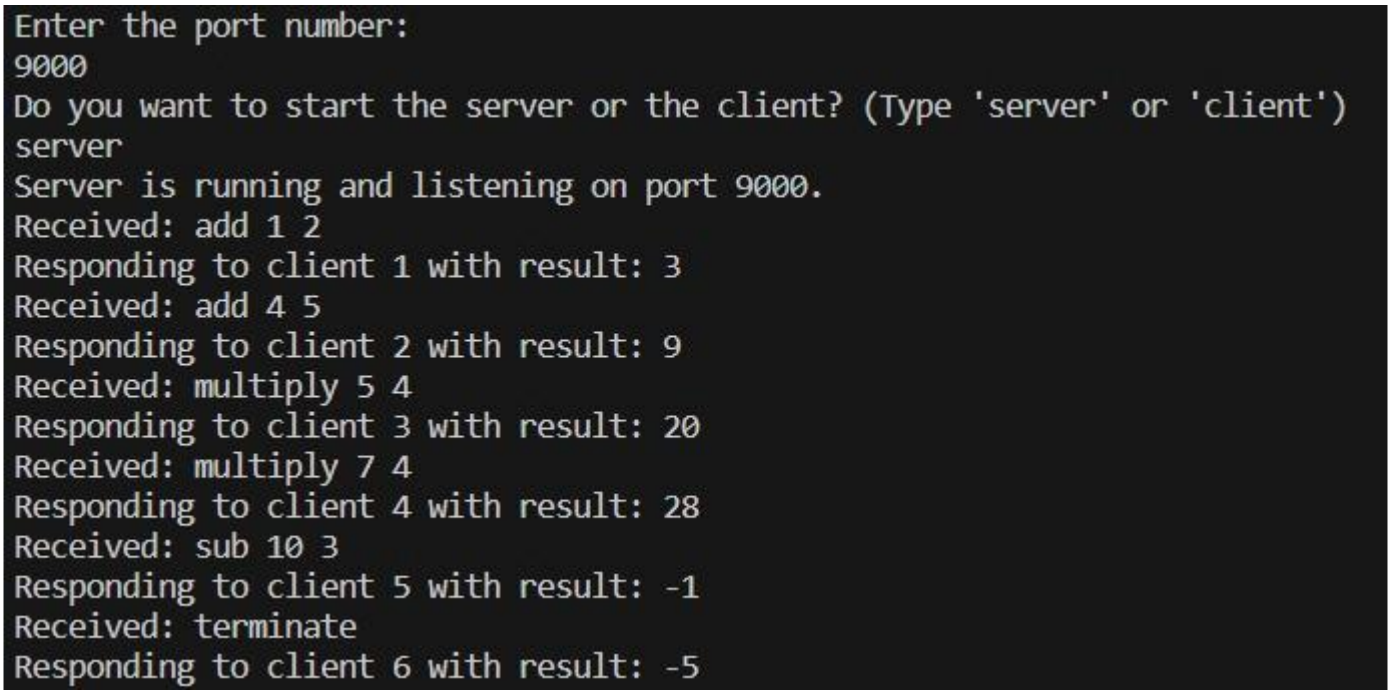
```
Enter the port number:
9000
Do you want to start the server or the client? (Type 'server' or 'client')
client
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
add 1 2
Sending command: add 1 2
Server response: 3
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
multiply 5 4
Sending command: multiply 5 4
Server response: 20
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
sub 10 3
Sending command: sub 10 3
Server response: Incorrect Operation Command
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
```

2)The second screenshot shows the second client we created, and again we passed commands and server responded correctly with proper error handling. As we can see at the last , the command **terminate** is used , which breaks all the connections between all clients and server. Server responds to the client with a result of -5. It closes the client connection using client.Close() and stops listening for new connections using listener.Stop()



```
Enter the port number:
9000
Do you want to start the server or the client? (Type 'server' or 'client')
client
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
add 4 5
Sending command: add 4 5
Server response: 9
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
multiply 7 4
Sending command: multiply 7 4
Server response: 28
Enter your command and numbers (e.g., add 1 2 3) or 'bye/terminal' to quit:
terminate
Sending command: terminate
```

3) The third screenshot shows from server side terminal, where we chose it by inputting server. It responds with the status of port number and port number. It displays the received communication form multiple client at the same time and simultaneously revert back with the accurate output. At the end we can see the **Received:terminate** which gave the server signal to end and break all the connections with all the clients and hence the -**5 .**



```
Enter the port number:
9000
Do you want to start the server or the client? (Type 'server' or 'client')
server
Server is running and listening on port 9000.
Received: add 1 2
Responding to client 1 with result: 3
Received: add 4 5
Responding to client 2 with result: 9
Received: multiply 5 4
Responding to client 3 with result: 20
Received: multiply 7 4
Responding to client 4 with result: 28
Received: sub 10 3
Responding to client 5 with result: -1
Received: terminate
Responding to client 6 with result: -5
```

Bugs and Limitations

1.**Delays in response**

We were getting delayed response from server side to multiple clients. The delay happened occasionally , but the lag was noticeable after client requested and server took some to respond. We thought of some solutions and optimizing the server code made it better to a particular extent.

2. **Automatic Port Detection**

We occasionally ran into a problem where the code couldn't figure out what port it needed to run on and wouldn't automatically allocate one. We discovered that we had to manually specify the port number because the code wasn't automatically detecting an open port for communication. This was a challenging problem as we couldn’t identify the core issue and where to find it. There were some port conflicts and unassigned ports.

3. We ran into a problem with connection management in our networked application. In particular, we anticipated that when we started the termination of a connection between the server and one client, this action would cascade and end all client-server relationships. But what we saw was that while the server and the terminated client would successfully disconnect, the other clients would stay connected and continue to operate whilst the server being “off”.

Work Distribution-

1)Prateek Abbi

Worked on server file, he mostly coded on the server program as he is much comfortable in F# language

2) Anirudh Attrey

Also co-worked on server file with Prateek. Mainly he made algorithms for server code.

3)Mudit Arya

Worked on Client source code and handeled errors and helped in making report

4) Syed Faizan Ali

Worked on client with mudit with exception handling and made majority of report.

For debugging ,we all sat together and assisted each other in finding out errors which took some significant time.