**DOCUMENTATION: GROUP MEMBERS VALENTINA, MADALINA AND PRANIT.**

**A short design description of the chosen design:**

For our chat server we decided to design the project so that it will be thread safe and by using the observer pattern to make the connection easily.

For example, the clientHandler class extends the thread class that actually runs every single time we have an instance of the clientHandler.

Each client has a clientHandler object, which allows him to connect to the server, this way getting the information needed from there.

The observer pattern is mainly used to implement distributed event handling systems. We have the ChatClient class as a subject, the ChatListener as an interface and the GUI as observer (which implements the interface).

The ChatListener notifies the clients when they receive a message and when someone else connects or disconnects.

**Implementing of State behavior:**

* the server

The server has at least 3 states of behavior.

One of the states is when the server gets a new client. In this state we connect the client with the server. The client waits for the server to accept the connection and then it returns the client with a socket.

State 2 is EDIT. In this state, the server splits a message into the protocol and the content, thereby figuring out what kind of input it has received and what to do with it.

The third state 3 is CLOSE. In this state, we disconnect the server. At this time, if we have any client connected to the server, it will disconnect automatically.

In our implementation, state 1 is implemented where we put the accept method in a loop, which exits only when the accept method is finished executing. Thereby the state is maintained and the system is prevented from crashing.

We implements the state 2 when we check whether the message got by the server is in the protocol. If it is not, the server has to know that it is not legal and thereby the system has to exit.

State 3 is implemented by the server receiving a message comprising of the Protocol CLOSE. State is maintained as nothing else closes the connection to the server except for this particular message. Therefore, the system is also prevented from unduly crashing.

**Who did what:**

Logic code: Madalina, Valentina, Pranit

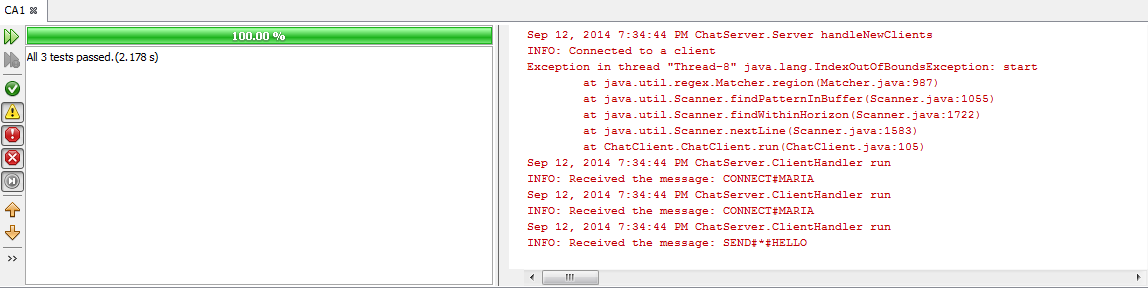
Tests: Valentina, Madalina

Web server: Pranit, Valentina, Madalina

Deploying on Azure: Valentina, Pranit

Description of the code: Madalina, Pranit, Valentina

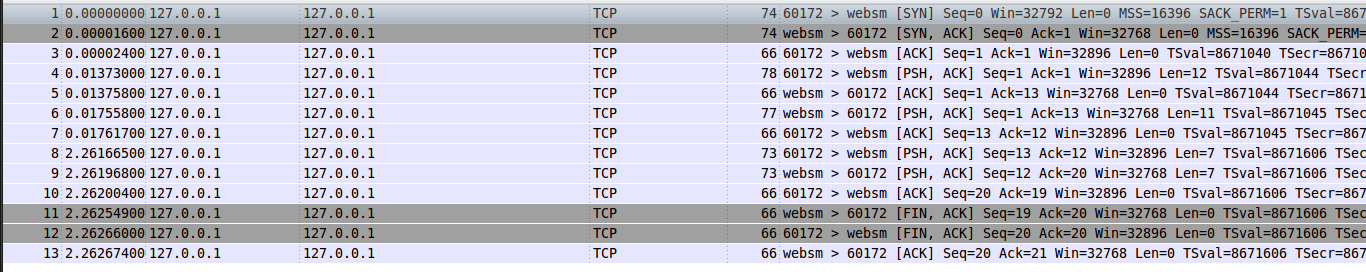
**Evidence of testing:**



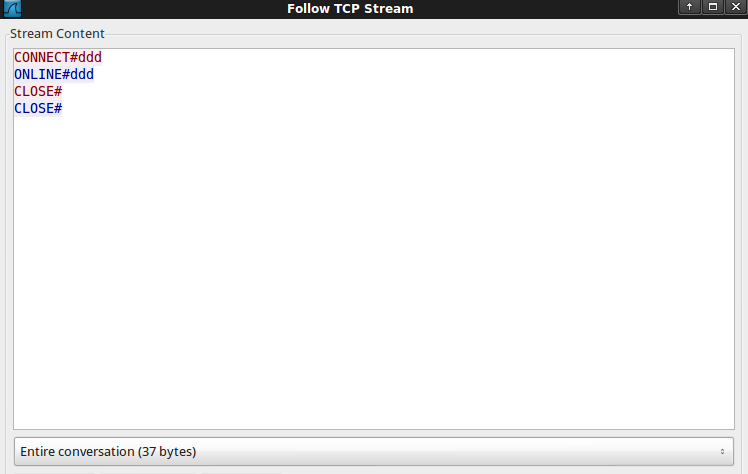
**Wireshark sample:**

We connect to the Localhost.

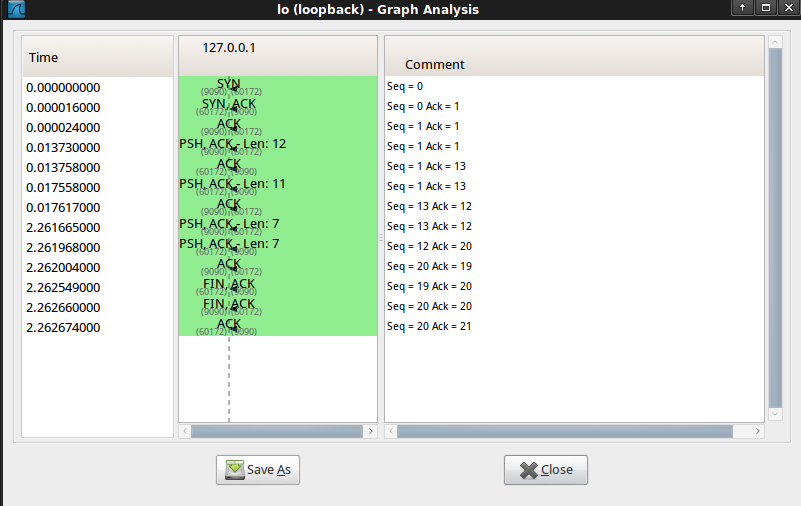
The packets sent and received can be seen below.

****

The following activities are logged:



The following FLOW graph is recorded:



The three way handshake is established in the first three movements.

The client synchronizes with the host (syn). The host synchronizes with the client and sends an acknowledgment to the client for his synchronization (syn/ack). Finally, the client acknowledges the hosts synchronization (ack).

The closing three way handshake is observed. The Client sends a finish flag to the host(fin/ack).

The ack flag along with it is to acknowledge the previous push from the host. The host sends a finish flag to the client and acknowledges the clients finish flag. The client acknowledges the hosts finish flag.

After the three way handshake there is a ‘psh/ack’, ‘ack’ and ‘psh/ack’ traffic. In the ‘psh/ack’ something is pushed along with an acknowledgement(total length:12). In the ‘ack’ the previous push is acknowledged. Lastly, in the ‘psh/ack’ another push with an acknowledgement occurs(total length 11).