A short description including:

o A short design description of the chosen design.

O A section stating who did what

o A short description of how you have implemented the state behavior

In our application design we tried to utilize as many OOP principles as we could.Not only did we use getters and setters, but we tried to make the system have high cohesion by creating classes that would contain the methods associated only with that particular class. That is why we have separate classes for GUI and the Chat client. It would not make sense to put these together since the GUI is only supposed to display the information. It does not have to know the connection to the server. We kept the coupling as low as we could but of course we have to have classes that know about others. Good example of this is the GUI class which knows the client but the client has no knowledge about the GUI. To make the GUI display the messages we used the Observer pattern which allowed us to create an illusion that the client knew the GUI, when in fact it only knows the method from the interface which is implemented in the GUI class. This solved the blocking issue on the client side if the GUI would be waiting for a message from the server and it would not be arriving. The client does not communicate directly with the server, it is handled by a ClientHandler class which is stored in a HashMap in the server. We chose hashmap because it is very efficient and the keys are the usernames, which will always be nonrepeating. Each time a new client connects a new client handler is created and a new thread is started by the client handler class. This way the client would not block the server. We made the chat server a singleton but we have not used double locking this time since we are sure that only the main method in the Webserver class will try to instantiate this object so no other threads have access to it. That of course means that we decided to have the Webserver as the top class that starts everything therefor it must know about the ChatServer class.

Smara was responsible for the Webserver class, she made the website and made it functional.

Cristi developed the GUI and he handled deployment of the program to the server.

Marek did the Chat server with the client and the testing.

In our design we had to think about how to differentiate between the states of the clients and the server itself because if we hadn’t done that we would be stuck with clients and the chat server that would exist forever. Therefor we decided that the server will run on a loop while the Boolean keepRunning will be true. This will allow us to turn the chat server off. To know active clients and know when to stop their threads we created a Boolean isActive which would tell us the state of the client. In case the client disconnected the state of the Boolean would be changed and the thread that run that client would be finished.