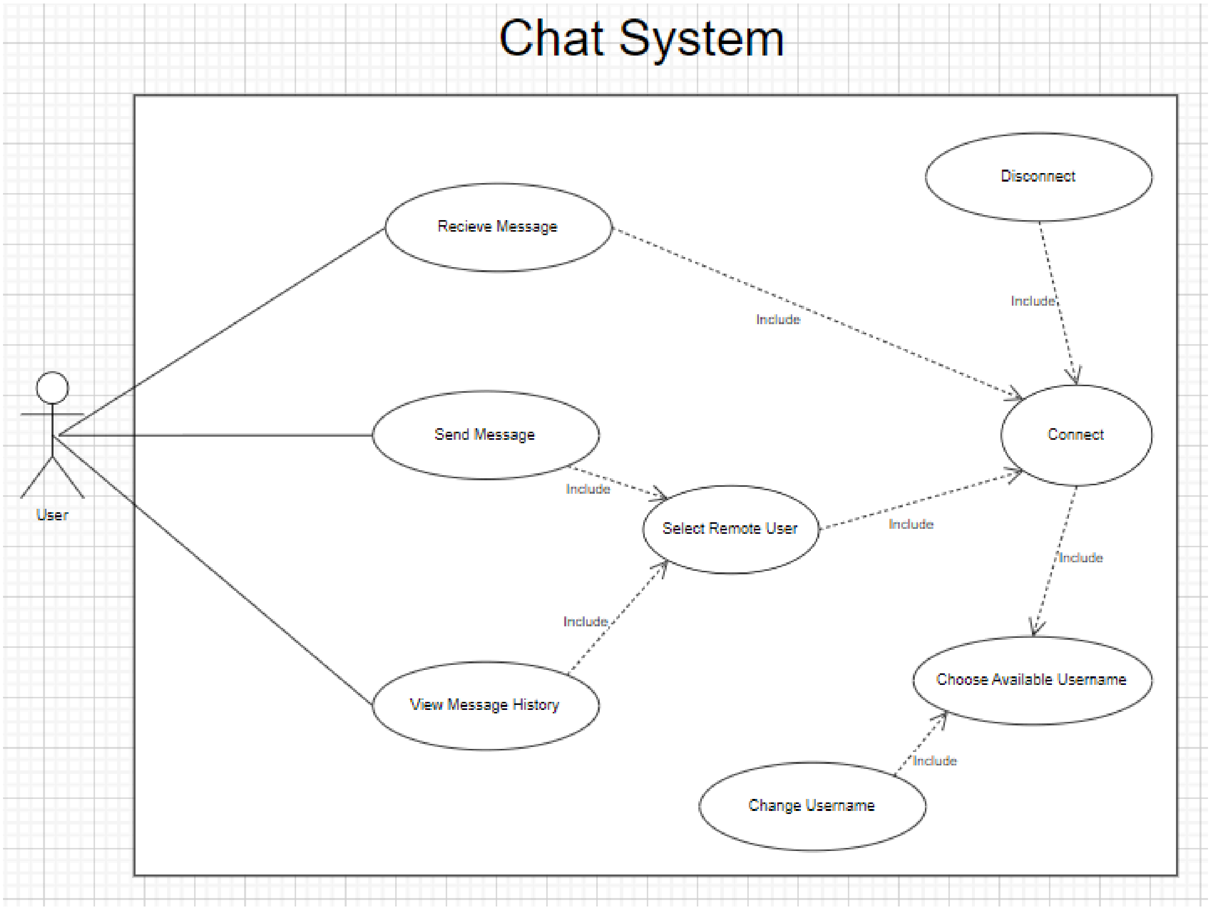
GREAKER Emilie, AUNAN Daniel 23/01/2024

UML Report

Table of contents :

Use case diagram :

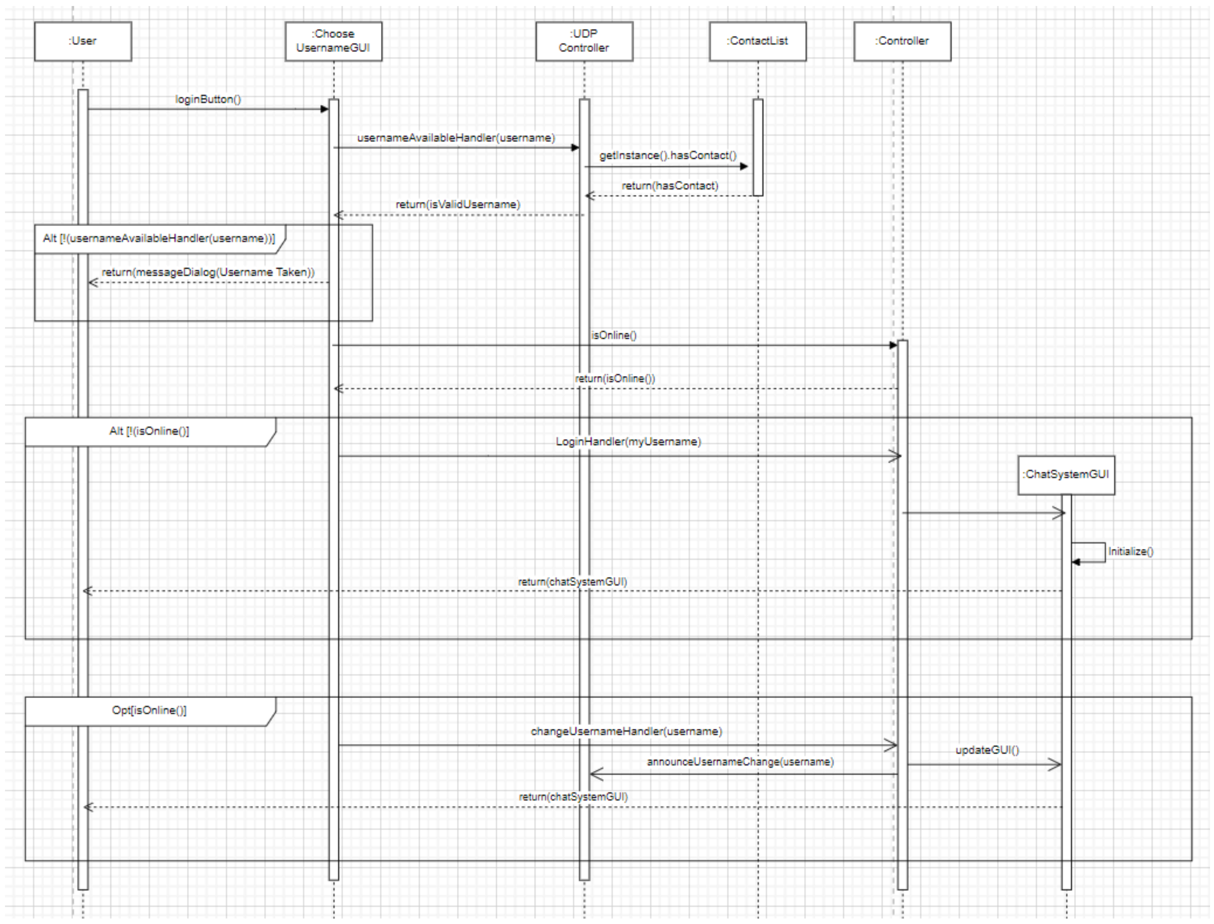
We constructed the use case diagram for the entire chat system as shown in the picture. We concluded that the user should be able to view the chat history or send a message to a remote user only after having selected a remote user from the contact list. From within the chat system, it would also be possible to change username and disconnect from the system.



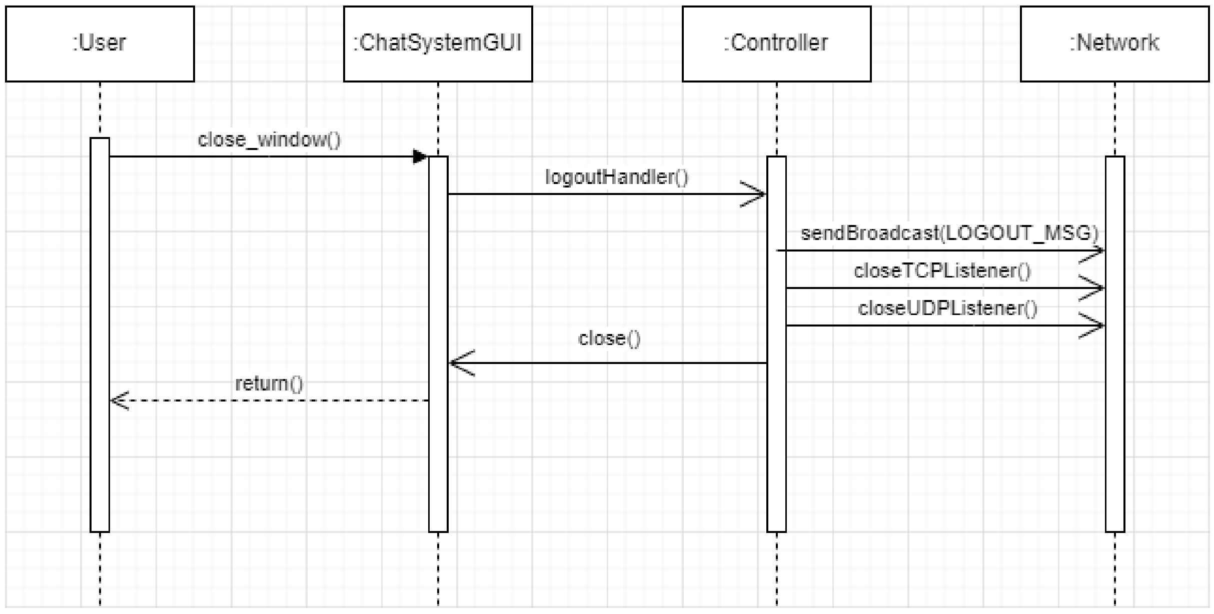
Sequence Diagrams :

For the sequence diagrams, we decided to create separate sequence diagrams for each use case in the use case diagram.

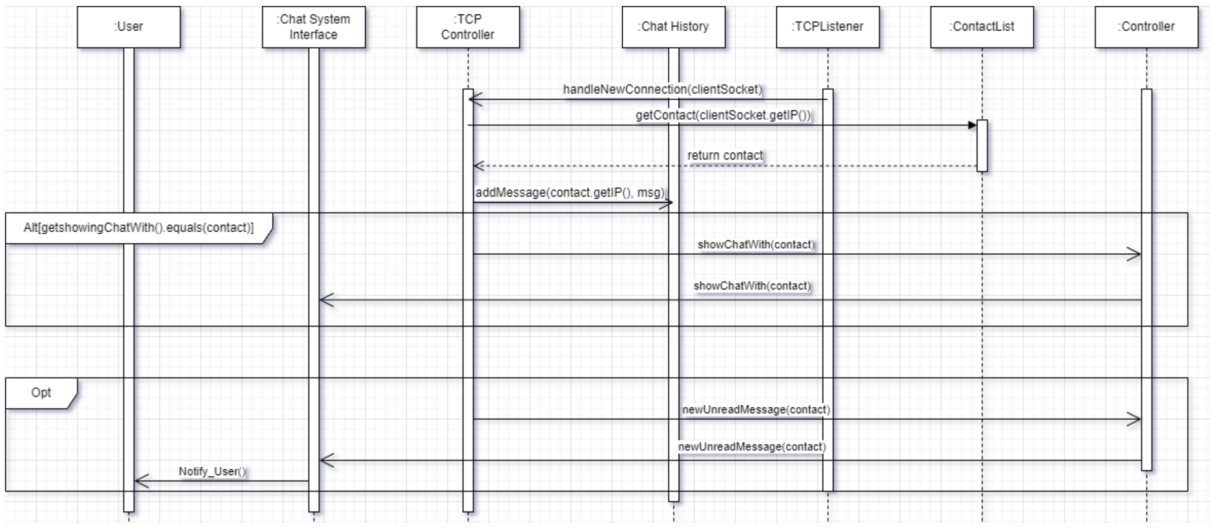
Sequence Diagram - Login to Chat System (Connect) :



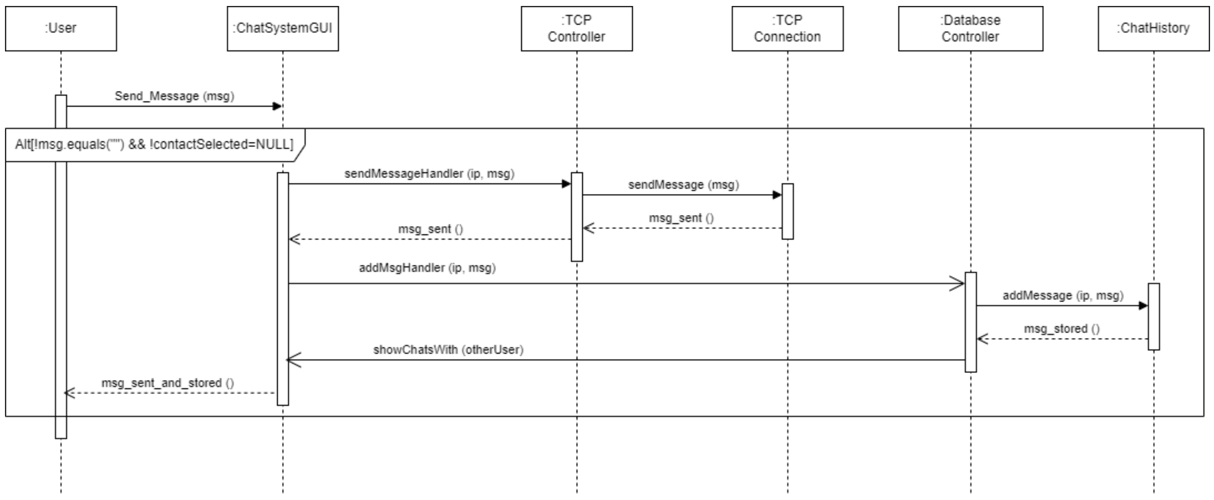
Sequence Diagram - Disconnect from Chat System :



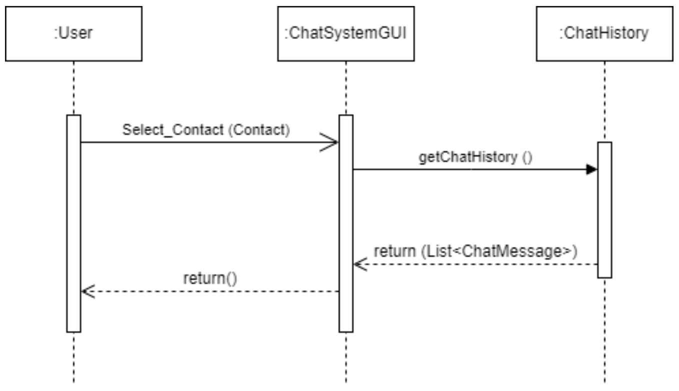
Sequence Diagram - Receive Message :


:User
:Chat System Interface
:TCP Controller
:Chat History
:TCPListener
:ContactList
:Controller
handleNewConnection(clientSocket)
getContact(clientSocket.getIP())
return contact
addMessage(contact.getIP(), msg)
Alt[getshowingChatWith().equals(contact)]
showChatWith(contact)
showChatWith(contact)
Opt
newUnreadMessage(contact)
newUnreadMessage(contact)
Notify_User()

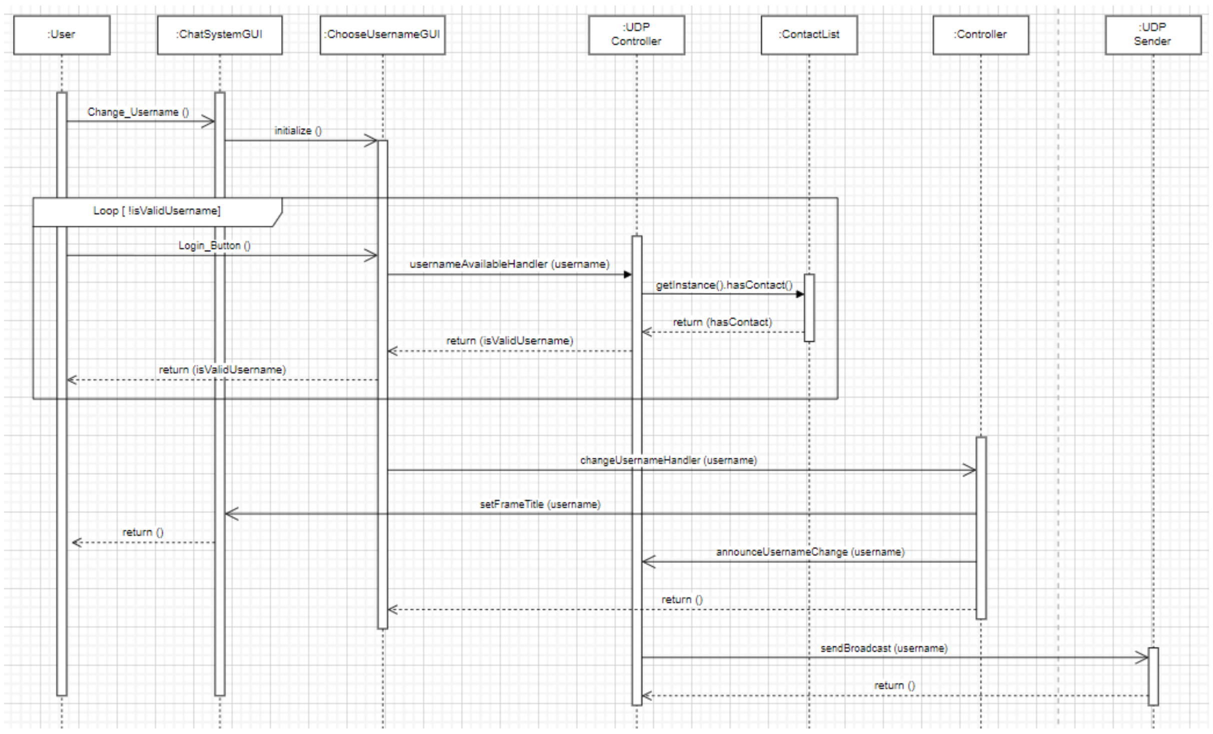
Sequence Diagram - Send Message :



Sequence Diagram - View Message History :

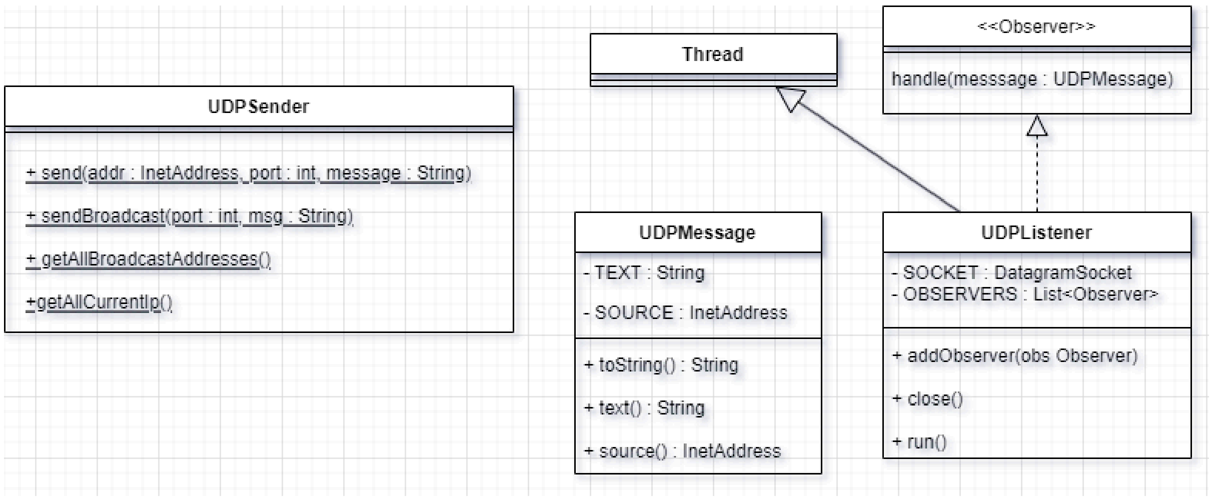


Sequence Diagram - Change Username :

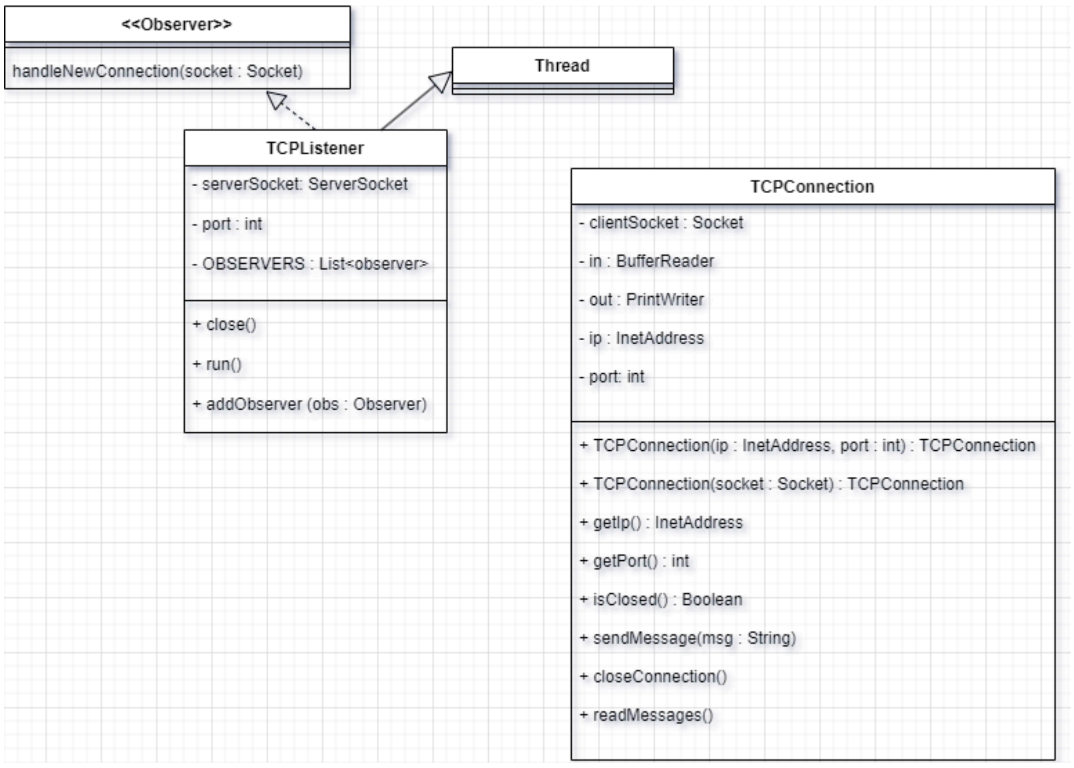


Class Diagrams :

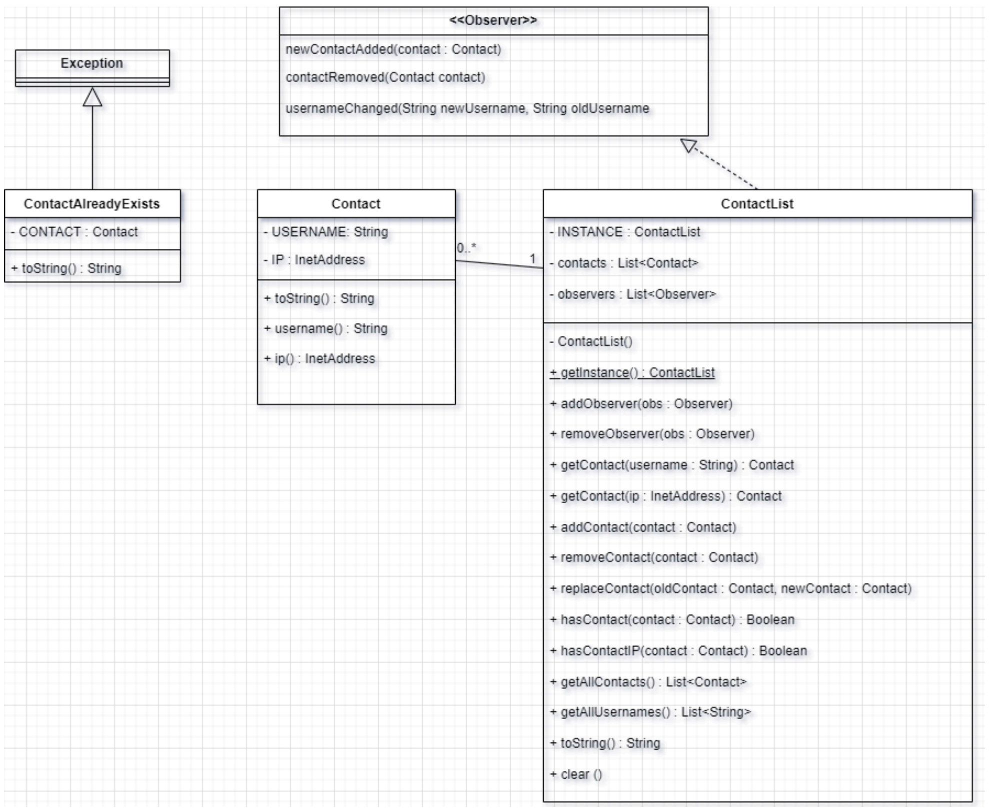
Class Diagram for the udp package :


UDPSender
+ send(addr : InetAddress, port : int, message : String)
+ sendBroadcast(port : int, msg : String)
+ getAllBroadcastAddresses()
+getAllCurrentIp()
Thread
<<Observer>>
handle(messsage : UDPMessage)
UDPMessage
- TEXT : String
- SOURCE : InetAddress
+ toString() : String
+ text() : String
+ source() : InetAddress
UDPListener
- SOCKET : DatagramSocket
- OBSERVERS : List<Observer>
+ addObserver(obs Observer)
+ close()
+ run()

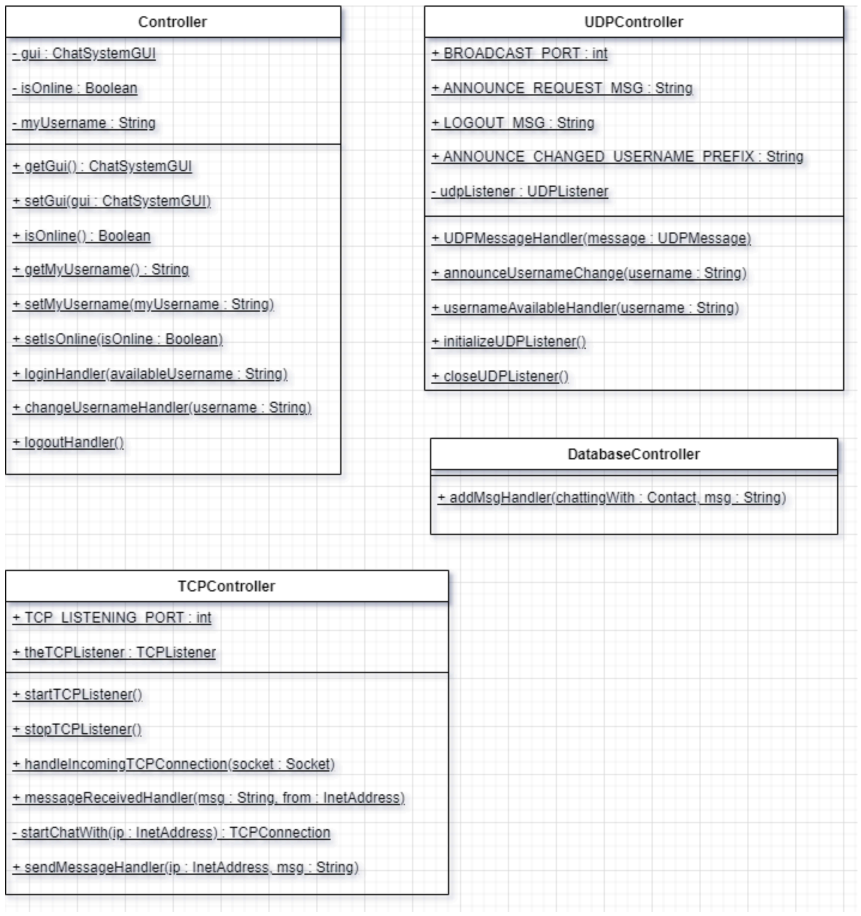
Class Diagram for the tcp package :


<<Observer>>
handleNewConnection(socket : Socket)
Thread
TCPListener
- serverSocket: ServerSocket
- port : int
- OBSERVERS : List<observer>
+ close()
+ run()
+ addObserver (obs : Observer)
TCPConnection
- clientSocket : Socket
- in : BufferReader
- out : PrintWriter
- ip : InetAddress
- port: int
+ TCPConnection(ip : InetAddress, port : int) : TCPConnection
+ TCPConnection(socket : Socket) : TCPConnection
+ getIp() : InetAddress
+ getPort() : int
+ isClosed() : Boolean
+ sendMessage(msg : String)
+ closeConnection()
+ readMessages()

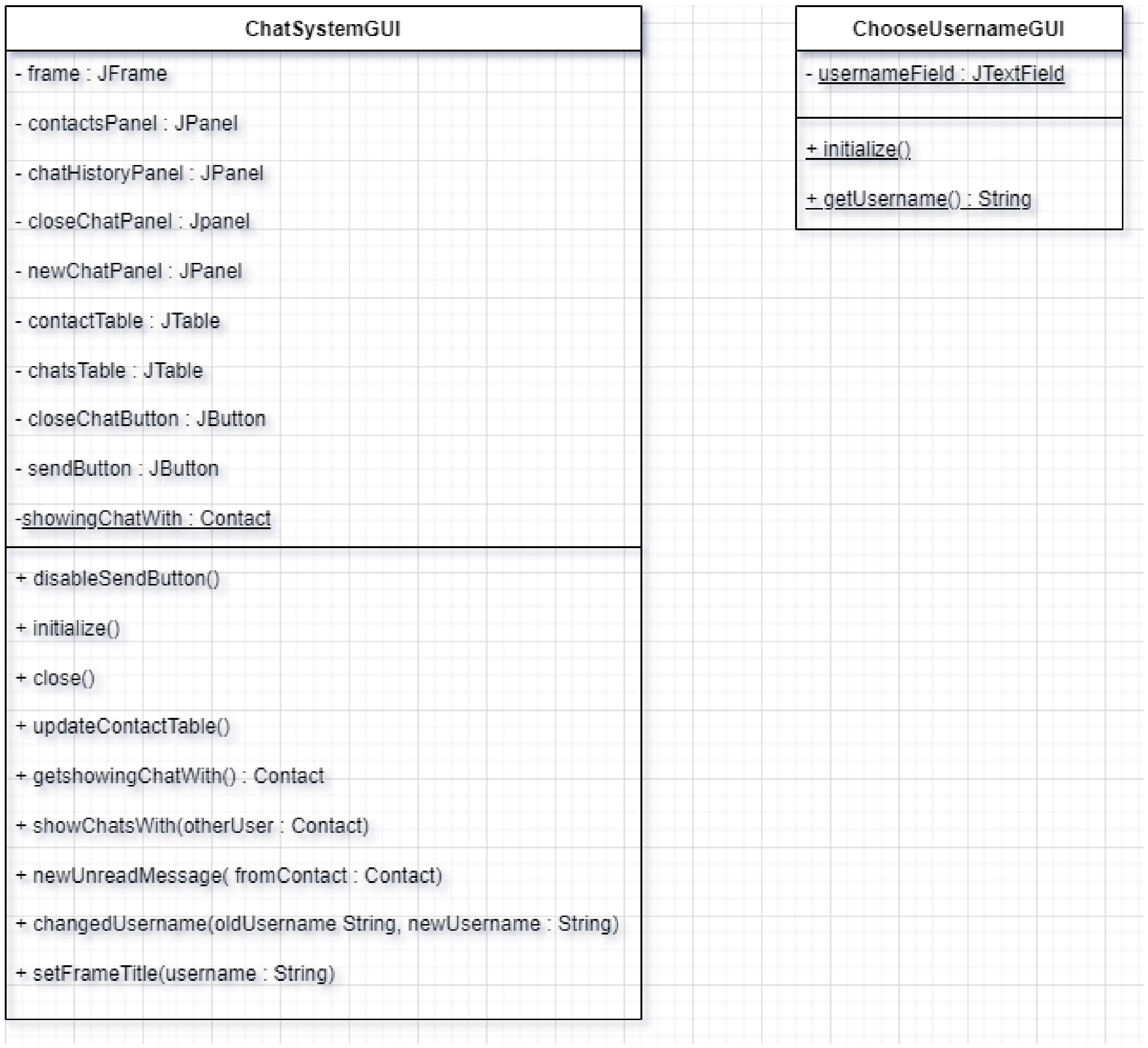
Class Diagram for the contacts package :


Exception
<<Observer>>
newContactAdded(contact : Contact)
contactRemoved(Contact contact)
usernameChanged(String newUsername, String oldUsername
ContactAlreadyExists
- CONTACT : Contact
+ toString() : String
Contact
- USERNAME: String
- IP : InetAddress
+ toString() : String
+ username() : String
+ ip() : InetAddress
0..*
1
ContactList
- INSTANCE : ContactList
- contacts : List<Contact>
- observers : List<Observer>
- ContactList()
+ getInstance() : ContactList
+ addObserver(obs : Observer)
+ removeObserver(obs : Observer)
+ getContact(username : String) : Contact
+ getContact(ip : InetAddress) : Contact
+ addContact(contact : Contact)
+ removeContact(contact : Contact)
+ replaceContact(oldContact : Contact, newContact : Contact)
+ hasContact(contact : Contact) : Boolean
+ hasContactIP(contact : Contact) : Boolean
+ getAllContacts() : List<Contact>
+ getAllUsernames() : List<String>
+ toString() : String
+ clear ()

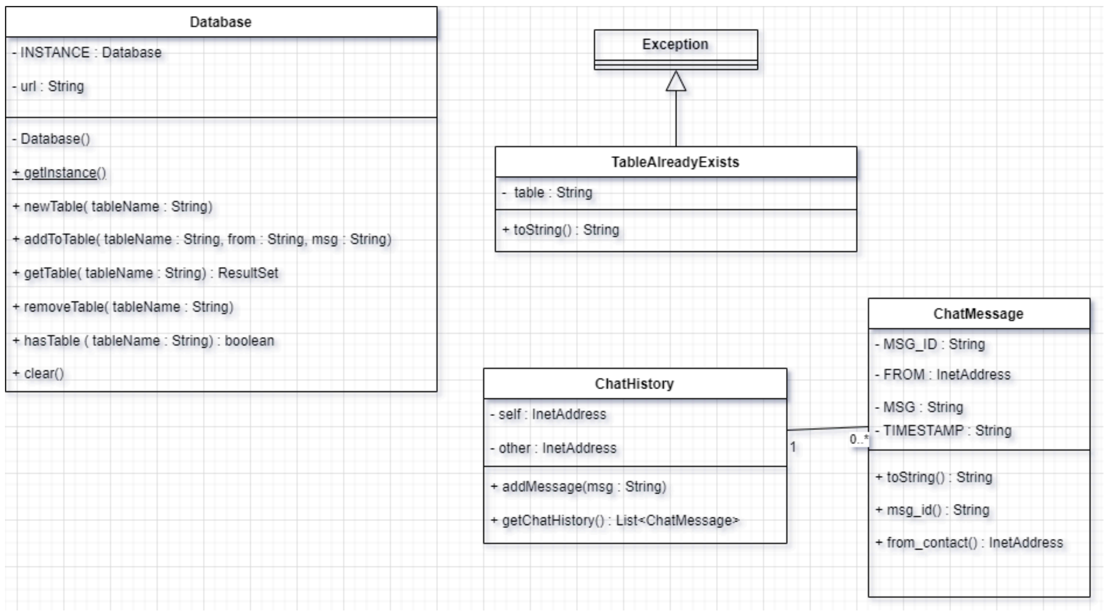
Class Diagram for the controller package:


Controller
- gui : ChatSystemGUI
- isOnline : Boolean
- myUsername : String
+ getGui() : ChatSystemGUI
+ setGui(gui : ChatSystemGUI)
+ isOnline() : Boolean
+ getMyUsername() : String
+ setMyUsername(myUsername : String)
+ setIsOnline(isOnline : Boolean)
+ loginHandler(availableUsername : String)
+ changeUsernameHandler(username : String)
+ logoutHandler()
UDPController
+ BROADCAST_PORT : int
+ ANNOUNCE_REQUEST_MSG : String
+ LOGOUT_MSG : String
+ ANNOUNCE_CHANGED_USERNAME_PREFIX : String
- udpListener : UDPListener
+ UDPMessageHandler(message : UDPMessage)
+ announceUsernameChange(username : String)
+ usernameAvailableHandler(username : String)
+ initializeUDPListener()
+ closeUDPListener()
DatabaseController
+ addMsgHandler(chattingWith : Contact, msg : String)
TCPController
+ TCP_LISTENING_PORT : int
+ theTCPListener : TCPListener
+ startTCPListener()
+ stopTCPListener()
+ handleIncomingTCPConnection(socket : Socket)
+ messageReceivedHandler(msg : String, from : InetAddress)
- startChatWith(ip : InetAddress) : TCPConnection
+ sendMessageHandler(ip : InetAddress, msg : String)

Class Diagram for the ui package:

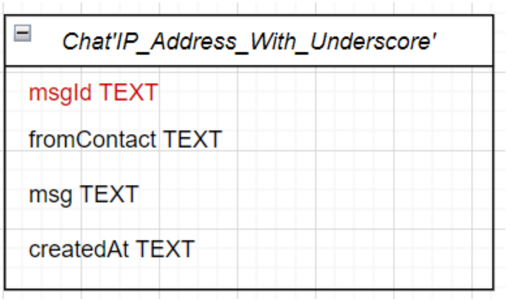

ChatSystemGUI
- frame : JFrame
- contactsPanel : JPanel
- chatHistoryPanel : JPanel
- closeChatPanel : Jpanel
- newChatPanel : JPanel
- contactTable : JTable
- chatsTable : JTable
- closeChatButton : JButton
- sendButton : JButton
-showingChatWith : Contact
+ disableSendButton()
+ initialize()
+ close()
+ updateContactTable()
+ getshowingChatWith() : Contact
+ showChatsWith(otherUser : Contact)
+ newUnreadMessage(fromContact : Contact)
+ changedUsername(oldUsername String, newUsername : String)
+ setFrameTitle(username : String)
ChooseUsernameGUI
- usernameField : JTextField
+ initialize()
+ getUsername() : String

Class Diagram for the log package:

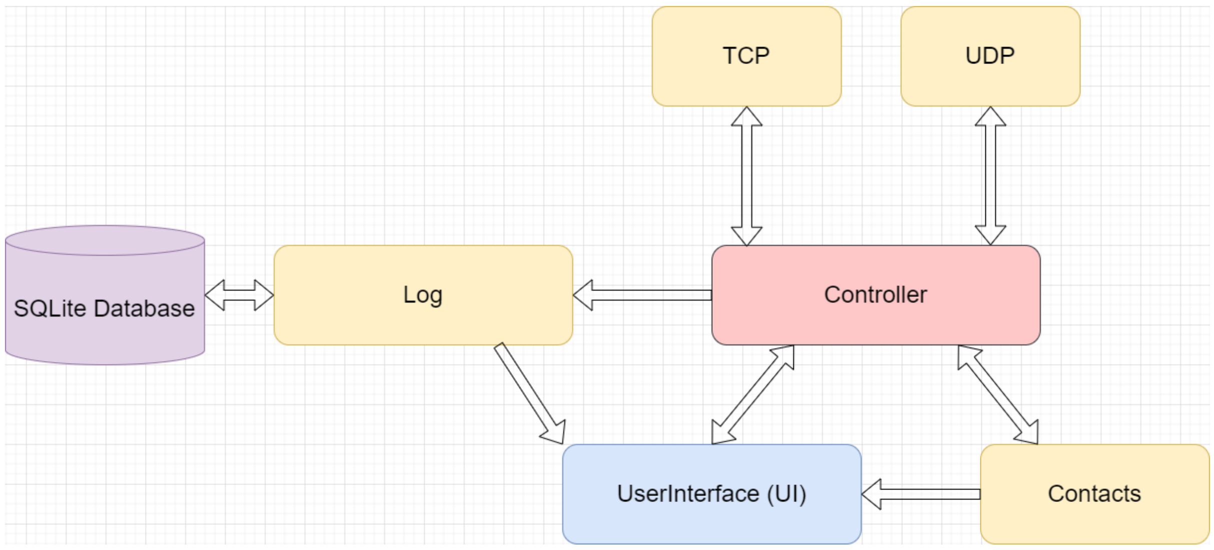


Database Schema :

Every table in the SQLite Database contains the ChatHistory with the IP address referenced in the table name.
For example; if the user is chatting with a remote user with the IP address 172.0.16.31, the name of the table storing the ChatHistory with this user would be 'Chat172_0_16_31'. This ensures that all chat histories with different IP addresses are stored in different tables. (SQLite table naming conventions specify that table names cannot start with a number and the use of dot '.' is forbidden)



Architecture Overview :



Project management :

At the start of the project we went over and discussed the requirements and specifications of the project, and set up a backlog for the newly defined tasks in the GitHub project. During the UML TD's we continued the planning process of the project by drawing up different diagrams, such as use case, sequence and class diagrams. We proceeded with mapping out the timeframes for when each feature should be finished and formulating the first sprint set with a timeframe of one week. We then divided the workload of the sprint between us and started implementing the first features. After working on the sprint for a week, we would meet up to present and discuss the progress, verify whether we had successfully completed the sprint and plan for the next upcoming sprint. By regular (weekly) planning of sprints and reviewing the work we produced each week, we were able to keep track of our progress and plan ahead to make sure we reached the deadlines. In this way, we incorporated the principles of the agile method in our project.

PDLA :

In our project we applied PDLA to better navigate through the project. We used github workflows for continuous integration. Github played a big part in organizing our teamwork using the agile method. We successfully used branches to keep our development process organized and effective.

For phase 1, "ContactDiscovery", we created a dedicated branch so we could work on implementing features collectively, while keeping the project configuration setup safely in the main branch.

The feedback we received after phase 1 was completed was crucial. We managed it by creating a post-feedback branch. This allowed us to make improvements to our project while having the original phase 1 branch to fall back on in case something went wrong. For the phase2 "message implementation (TCP)", we in similar fashion created a dedicated branch.

Our Github Actions workflow, defined in the java.yaml file ensures that every push triggers code compilation and testing. Thus following PDLA principles, we facilitated the development approach, collaboration and automation.