

CPSC 441

Agni - A Command Line Messenger

Design Document

Group 45:

Enoch Tsang

Bea Esguerra

Moath Althawad

James Inglis

Table of Contents

1	Network System Overview	3
2	System Analysis	4
3	Use Cases	5
4	Message Format	7
5	Order of Message Exchange	11
6	Message Handling	15
7	User Interface	17
8	Object-Oriented Design of the Implementation	19
9	Expectations	23
10	Plan	24

1 Network System Overview

Description and Motivation

Modern technology is becoming ever more aimed at the average consumer, making command line programs a thing of the past compared to rich guided user interfaces. But still, many developers who create these rich user interfaces commonly prefer the lightweight productivity of the command line. There are many amazing tools, `cp` `mv` `rm`, that are all great for managing files, `grep` `locate` `find` for searching for files or words, and even `scp` `wget` for remote file transfer, but what about instant messaging?

Introducing Agni, a lightweight purely command line instant messenger for developers. The name comes from the Hindu deity Agni who was a swift messenger and keeper of divine knowledge. Agni will support multiple users in the same chat, real-time updates of peers, security with encryption, and high customizability in messages.

Full Feature List

- User accounts (profiles) on the messenger server
- User login/logout (no encryption needed)
- Create and maintain a friend list
- Support chat
- List friends' online/offline status in real time
- Maintain chat history/log
- List all online friends
- Group chat (invite friends to join)

Additional Features

- Secure login/logout with encryption and salted hashes (using a hash function and a unique value to determine a unique hash code for passwords)
- Message customization with colors
- Customizable macros (similar to emoticons) saved per user (users will be given default macros)

2 System Analysis

Loss Tolerance

The Agni system should not tolerate data loss. Data loss should be prevented because it is crucial for users to receive all data sent to them, whether it be a message.

Throughput

The network throughput required for the system varies. For sending and receiving messages, the throughput required would be 69.5B/s.

Data for Average Throughput Calculation

Datagram item	Bytes
Empty TCP datagram	64 bytes
Message length & message type	5 bytes
Average message size	70 bytes
Average datagram	139 bytes
Required throughput for 4 concurrent users (datagram bytes/ average message occurrence rate)	69.5B/s

Transport-Layer Protocol(s) Use

The Agni system will be implemented based on TCP because the reliability of sending and receiving messages is crucial in a messenger program. In addition, using TCP Agile development methodology will be used throughout the development lifecycle of the application. This will allow us to better test each milestone as soon as it is developed. Moreover the use of Agile will enable shorter development cycles, early feedback and continuous improvement.

3 Use Cases

Connection No Connection Yet

Successful Connection

1. User starts program with server information as command line arguments
2. Server accepts connection from client and keeps track of client as TCP socket
3. Client sends consistent heartbeats
4. Server sends consistent heartbeats

Unavailable Server

1. User starts program with server information as command line arguments
2. Server is unreachable for any reason
3. Client attempting connection times out
4. Client exits

Client Heartbeat Timeout

1. User starts program with server information as command line arguments
2. Server accepts connection from client and keeps track of client as TCP socket
3. Client sends consistent heartbeats
4. Client receives no heartbeat from server, timeout
5. Client closes

Server Heartbeat Timeout

1. User starts program with server information as command line arguments
2. Server accepts connection from client and keeps track of client as TCP socket
3. Server sends consistent heartbeats
4. Server receives no heartbeat from client, timeout
5. Server forgets client via deleting associated TCP socket

Login

Successful Login

1. Client sends login request
2. Server notifies successful authentication

Unsuccessful Login

1. Client sends login request
2. Server notifies bad login/password

Server Information Request

Information Request

1. Clients sends information request
2. Server sends requested information

Information Request Types

- Server Name, IP, and Port
- Current users online
- Current chats

User Actions

Successful User Request

1. Client sends user request
2. Server notifies completion of user request
3. Server sends additional information in response to user request

Bad User Request

1. Client sends user request
2. Server notifies error in user request

Adding Friends

1. Clients A adds Client B as friend
2. Server notifies completion of user request
3. Server notifies Client B that Client A has added him/her

4. Client B adds Client A as friend (server acknowledges)
5. Server notifies clients of successful friendship

User Request Types

- Join Chat
- Leave Chat
- See friends list
- Check friend status
- Add friend
- Logout

Chats

Regular Chat

1. Client A, B, C in chat X; Client D not in any chat
2. A sends chat message "hello"
3. Server finds clients in same chat as A, server finds B, C
4. Server sends message "hello" to B, C from A
- 1.

4 Message Format

Reserved Bytes on All Messages

Field Name	Byte Number	Purpose and Usage
Length	0 to 3	- unsigned int representing length in bytes of entire message
Message Type	4	- identify message type - byte - type numbers listed below

Client Sent Messages

Heartbeat - 0x01

A message sent in regular intervals to let the server know the client is still alive.

Field Name	Byte Number	Purpose and Usage
Status	5	- byte representing status of client - 0x00 offline, 0x01 Online, 0x02 Away

Information Request 0x03

A message to send a request for Agni server information

Field Name	Byte Number	Purpose and Usage
Request Type	5	- byte representing the type of request information - Server IP (0x00) , Port (0x01), and Name (0x02) - Current users online (0x03) - Current chats (0x04)

Login - 0x02

A message to request login at the IP that the packet was sent from

Field Name	Byte Number	Purpose and Usage
Type	5	-represents whether it is a login or new user reg -login (0x00) -reg (0x01)
Username Length	6	- byte representing length of sent username - max 255
Username	7 to (7+ user length)	- ASCII encoding

Password	(8 + user length) to (end of message)	- ASCII encoding
----------	---	------------------

User Action - 0x04

A message sent by the client to request an action

Field Name	Byte Number	Purpose and Usage
Action Type	5	- byte representing the type of user action - Join Chat(0x00) - Leave Chat(0x01) - See friends list(0x02) - Check friend status(0x03) - Add friend(0x04) - Logout(0x05)
Argument	6 to 6 + user length	An additional argument for a user action (sometimes empty)

Chat - 0x05

A message sent from the client to the current chat

Field Name	Byte Number	Purpose and Usage
Message	5 to (end of message)	- ASCII encoding - Max ~65017 bytes

Server Sent Messages

Heartbeat - 0x07

A message sent in regular intervals to let the client know the server is still alive.

Field Name	Byte Number	Purpose and Usage
No Field	-	- length of server heartbeat message is always 5

Information - 0x08

Information sent from the server to give the user notifications related to the server (i.e. notifications about errors, bad login, server timeout, etc.)

Field Name	Byte Number	Purpose and Usage
Notification Message	5	- ASCII encoding - max ~65017 bytes

Chat - 0x09

The chat message sent from the server that was received from a client. Includes the text to send and information about the sender

Field Name	Byte Number	Purpose and Usage
Sender Length	5	- byte representing length of sender name - max 255
Sender Name	6 to (6 + sender length)	- ASCII encoding - max length 255
Message	(7 + sender length) to (end of message)	- ASCII encoding - Max ~65017 bytes

Status Notification - 0x0b

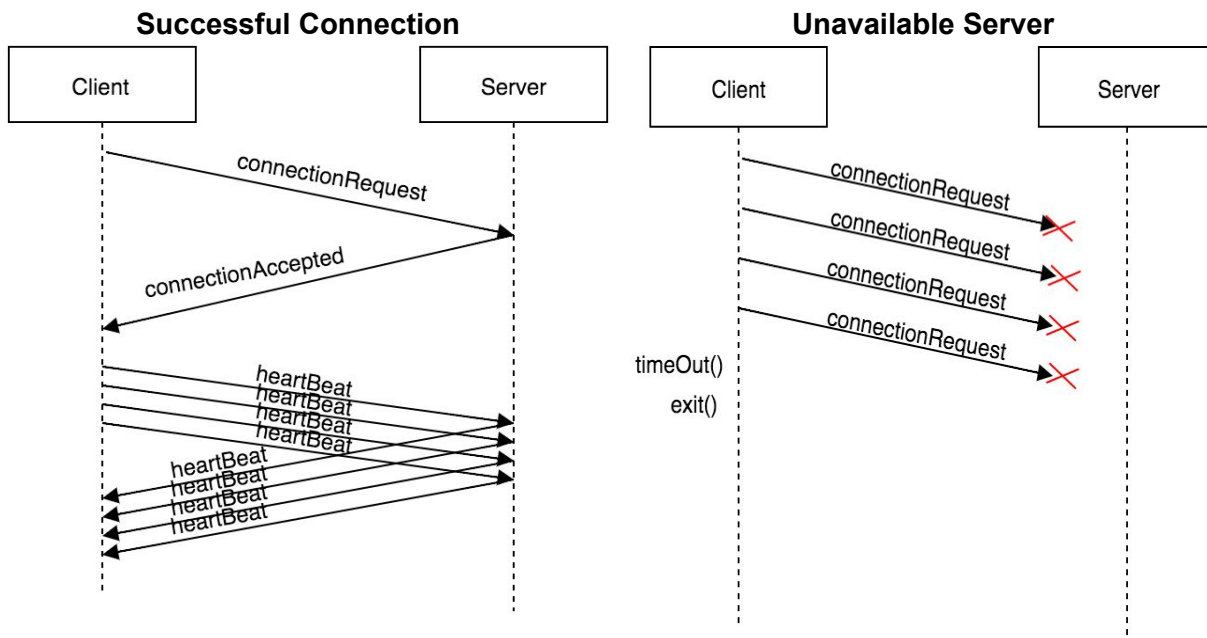
Information about changes of friend's statuses (i.e., when a user logs in or logs out, his/her friends will be sent a status notification message).

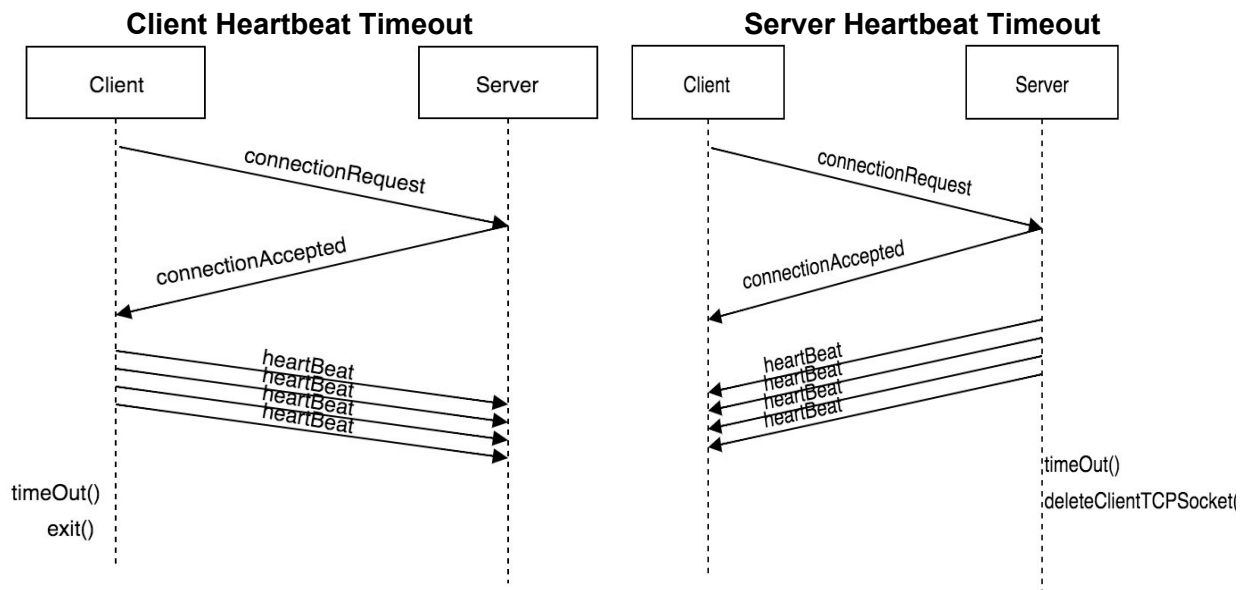
Field Name	Byte Number	Purpose and Usage
Status	5	- byte representing status of client - 0x00 offline, 0x01 Online, 0x02 Away
Friend Username	6 to (friend username length)	- ASCII encoded name of friend who logged in/out - Max 255 bytes

5 Order of Message Exchange

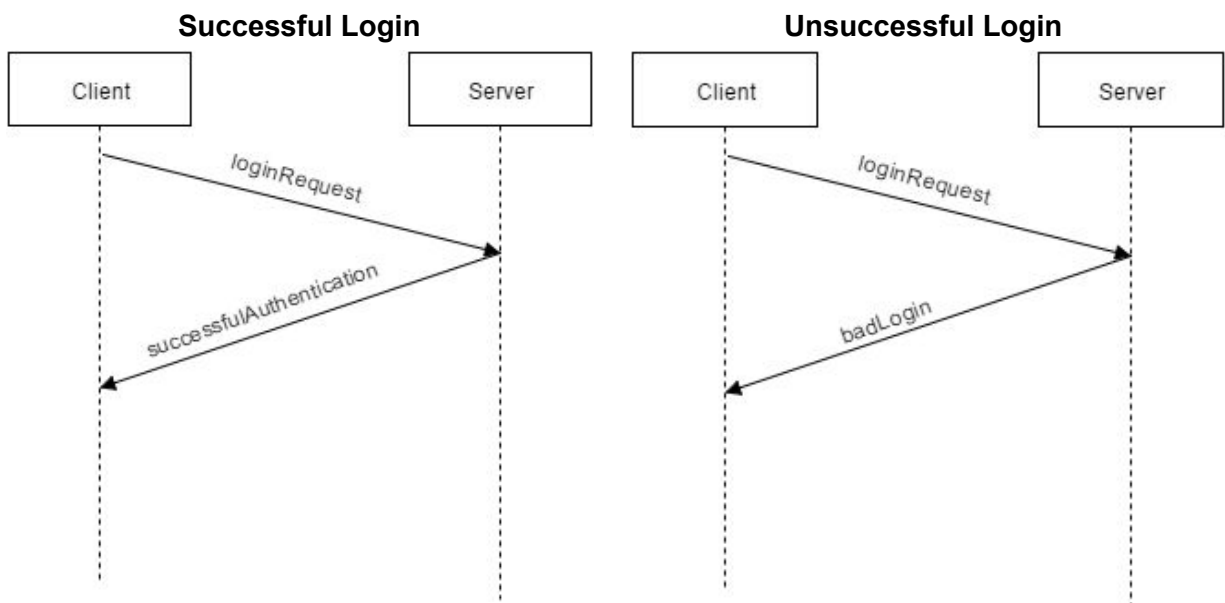
The following timing diagrams demonstrate the order of message exchange. Since the system will be using TCP, a TCP connection must be set up prior to each request. For simplicity in the diagrams, this TCP connection setup was abstracted away so that the diagrams could better demonstrate the order of message exchanges for the use cases described previously: login, server information requests, user actions, and regular chat.

Connection

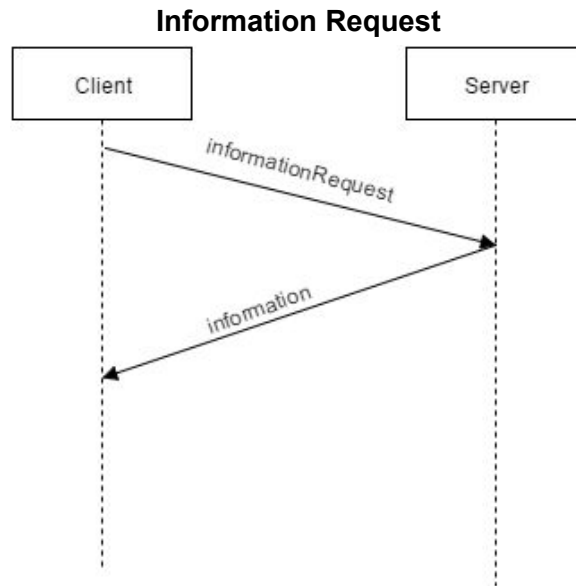




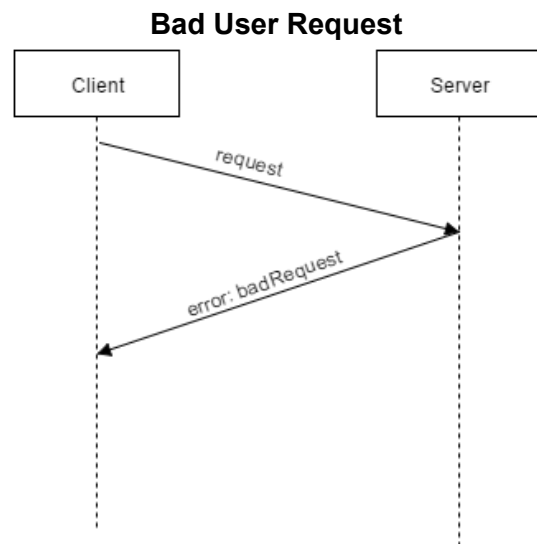
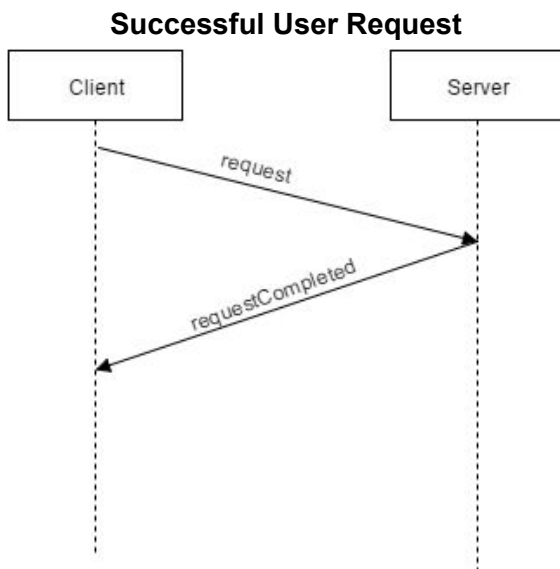
Login



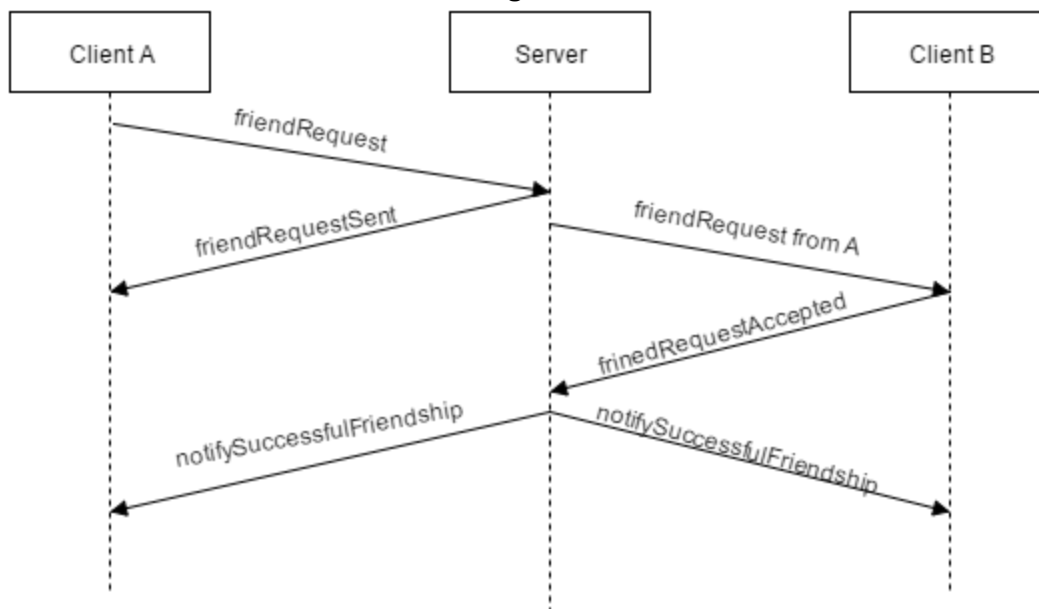
Server Information Request



User Actions

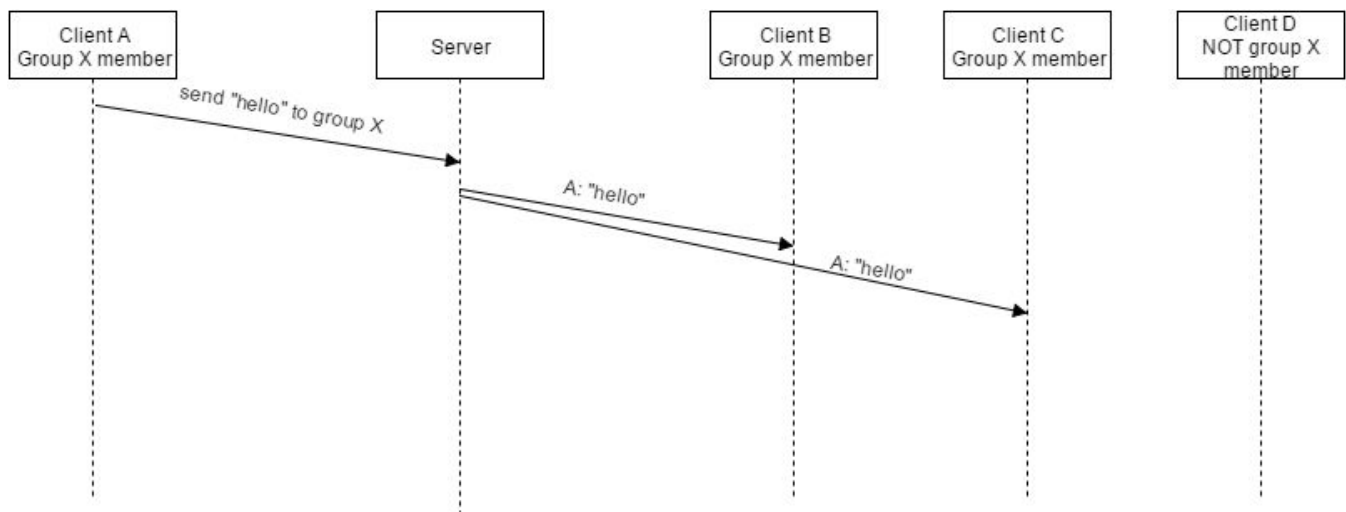


Adding Friends



Regular Chat

Regular Chat



6 Message Handling

Client

Receiving

All messages will be received by the 'MessageReceiver' class which parses the data and deciphers the message type based on the 5th byte. Depending on message type (HeartBeat, Info, Status, Chat) the message will be passed to the appropriate Receiver class. The receiver classes utilize the MessageParser class to parse the message and notify the handler classes which are receiver listeners. The handlers contain the logic required to perform the appropriate action based on the message type these actions include but are not limited to using standard out to display chat messages.

All transport layer errors including packet drops and ack losses will be handled by the TCP. Unexpected client-server disconnects will be detected via the heartbeat signal which allows the entities to monitor each other's status.

Sending

All messages that are slated for sending to the server originate in the view classes. depending on the message type, a handler class will be called which appends the 5 bytes containing message length and message type and passes the message on to the MessageSender class. The MessageSender passes the complete message to transport layer.

All transport layer errors including packet drops and ack losses will be handled by the TCP. Unexpected client-server disconnects will be detected via the heartbeat signal which allows the entities to monitor each other's status.

Server

Receiving

On the server side, all messages will be received by the 'MessageReceiver' class which parses the data and deciphers the message type based on the 5th byte. Depending on message type (HeartBeat, Info, Status, Chat) the message will be passed to the appropriate Receiver class. The receiver classes utilize the MessageParser class to parse the message and notify the manager classes which are receiver listeners. The managers contain the logic required to perform the appropriate action based on the message type these actions include but are not limited to accessing the database and reading/writing from/to it.

All transport layer errors including packet drops and ack losses will be handled by the TCP. Unexpected client-server disconnects will be detected via the heartbeat signal which allows the entities to monitor each other's status.

Sending

When a Manager class is notified about a message from a Receiver class, it sends a request to the proper Sender class depending on the message type. That request contains the data to be sent to a client.

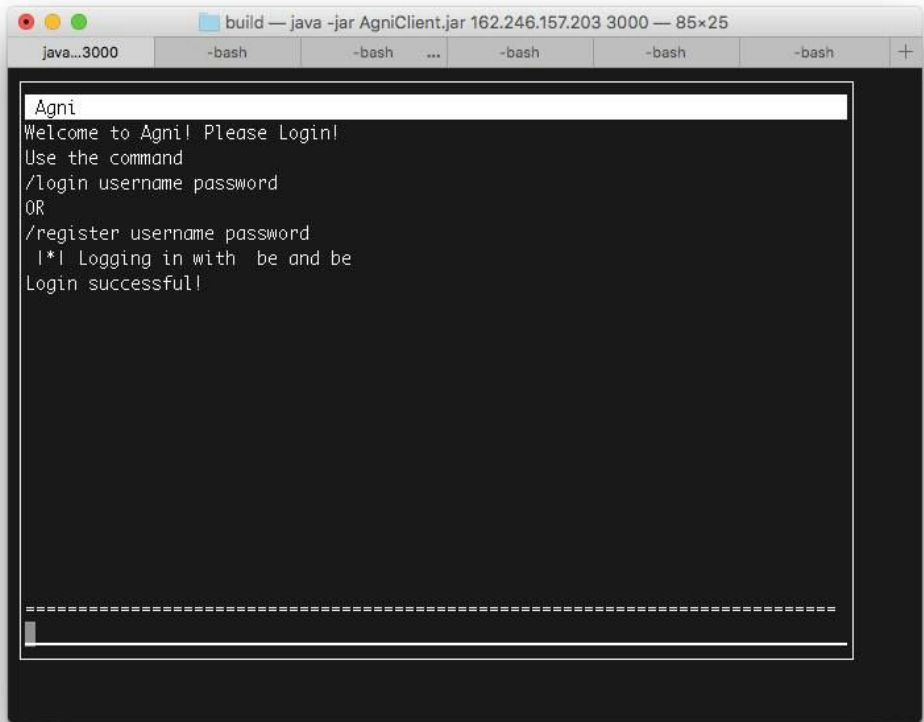
All transport layer errors including packet drops and ack losses will be handled by the TCP. Unexpected client-server disconnects will be detected via the heartbeat signal which allows the entities to monitor each other's status.

7 User Interface

The following images show the user interface that we plan on implementing for Agni Messenger. Since the application is run on a terminal, the interface will be displayed in the terminal. It is important to note that the commands inputted by the user (the grey text that starts with '>') shown below) would not be displayed in the interface. For the purpose of this report, the commands were shown to demonstrate their functionalities.

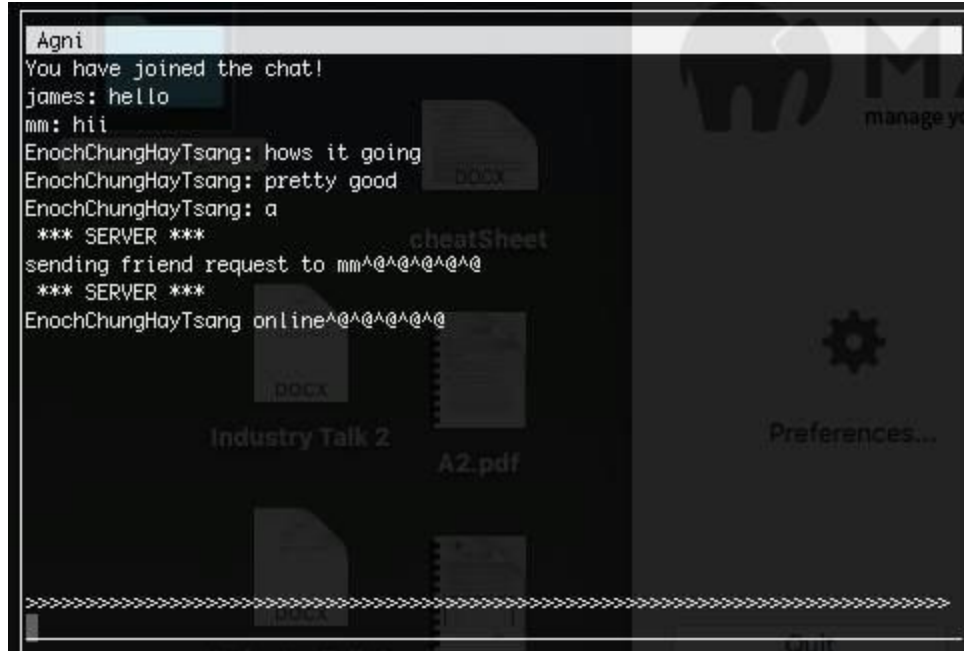
The functionalities shown in the images are:

- Connecting to the Server and Logging In (after the server and client is setup)
- Displaying Available Commands
- Chatroom Functionalities and Change of Friends' Status
- Listing Friends and Chat Rooms



```
build — java -jar AgniClient.jar 162.246.157.203 3000 — 85x25
java...3000  -bash  -bash  ...  -bash  -bash  -bash  +
Agni
Welcome to Agni! Please Login!
Use the command
/login username password
OR
/register username password
! Logging in with be and be
Login successful!
-----
```





8 Object-Oriented Design of the Implementation

The following diagrams below show the object-oriented design of the implementation.

For the client interaction, the main components of the design are the views, handlers, receivers, message receiver, and message sender.

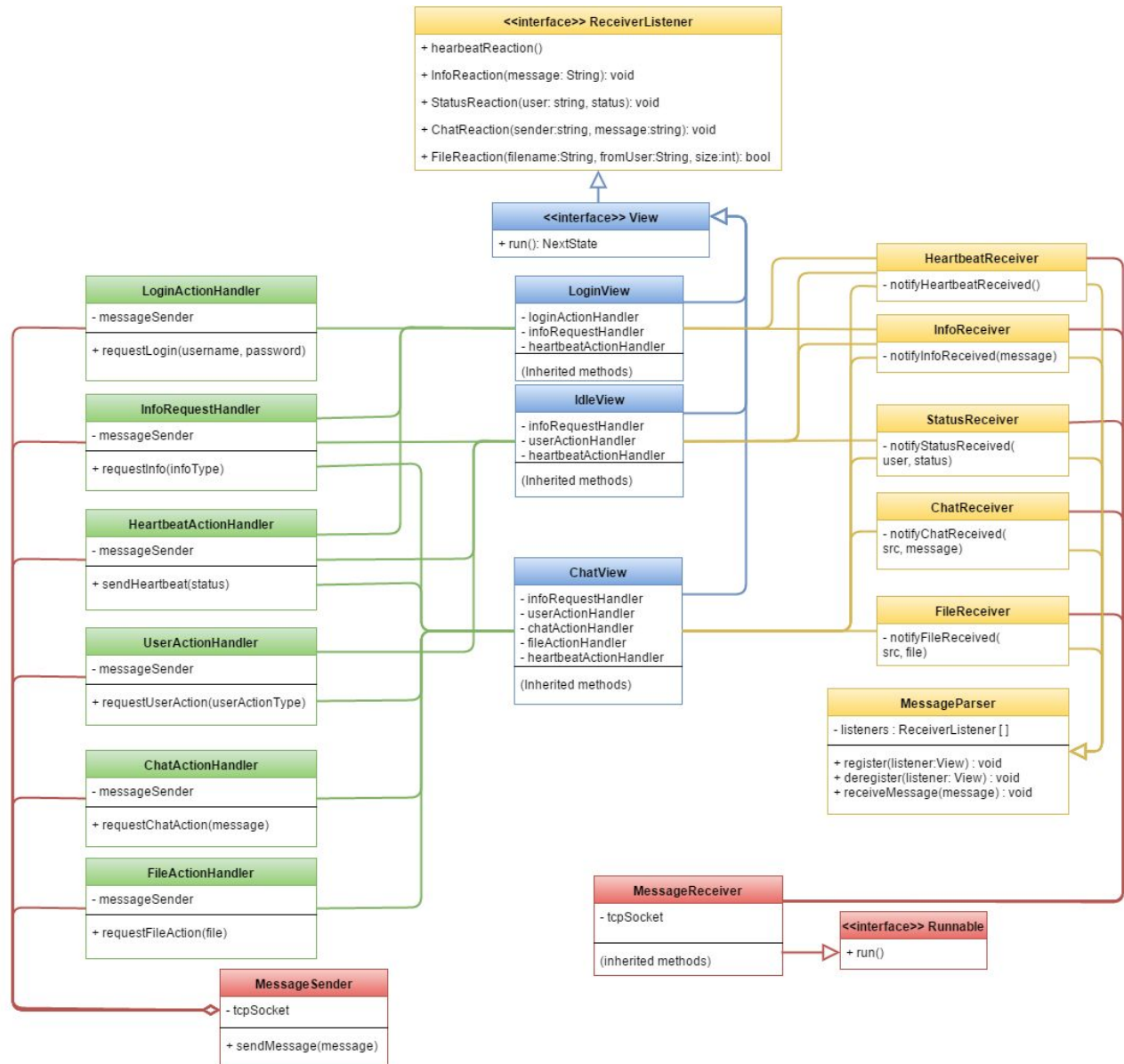
There are three views that deal with the state of the application: the login view, idle view, and chat view. Each view is associated with specific handlers to deal with specific types of requests to send. For example, the chat view deals with handling information requests, heartbeat actions (constant messages notifying that the connection is still alive), user actions, and chat actions. Each view is also associated with receivers to deal with receiving specific types of data. For example, the ChatView deals with receiving data from the receivers for chat, statuses, information, and heartbeats.

The message sender consists of the handlers and the message receiver consists of the receivers. The message sender and message receiver are the components that directly interact with the TCP socket that communicates with the server.

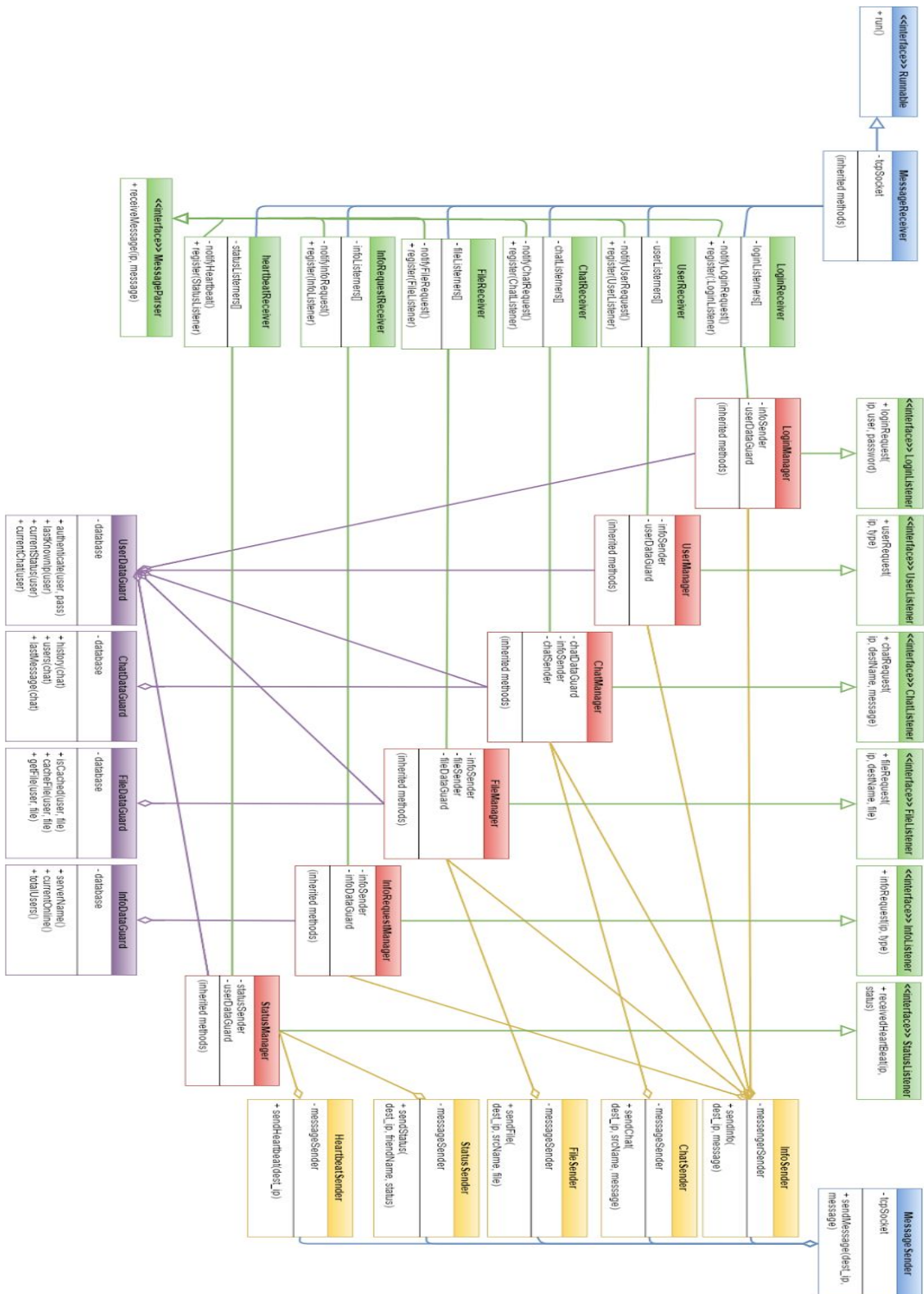
For the design of the server interaction, the main components are the managers, data guards, receivers, senders, message receiver, and message sender.

There are six types of managers that manage specific components of the system: login, user, chat, information request, and status managers. Managers have associated receivers that listen for received messages from the message receiver that is specific to what they manage. Each manager also interact with the user, chat, and information data guards, which serve to communicate with the database. Additionally, the managers use specific types of senders (information, chat, status, and heartbeat senders) to send certain types of data. The message sender consists of these types of senders. Similar to the message sender in the client, they directly interact with the TCP socket to communicate with the client.

Design of Client Interaction



Design of Server Interaction



9 Expectations

The program should be extremely quick and snappy because the entire motivation of the Agni is to remove all the extra clutter that comes with a GUI. All actions should feel almost instant. Downloading or uploading programs should occur at a similar speed to downloading from a web browser or the common linux command `wget`.

Another expectation about the quality of the project would be that the project should not lose any data when sending messages. The program should be reliable. It should also show all real time notifications accurately without any delays. For example, when a user logs on to the system, the user's friends should be notified immediately.

The chat logs are expected to contain the history of the chat room from the creation of the chat until the deletion of the chat. Only the host (the user who created the chat room) can remove the chat. It is important to note that although no users are not active in the room, the room would still exist and continue to maintain the chat logs. This means that if all the users of a chat log out, the room would still exist and the users would be able to access the history later on (if they are still in the chat room).

In addition, we expect the program to be secure since we planned to enhance login security by creating a unique hash code for user passwords.

Lastly, we also expect that the user interface should to be easy to use and intuitive for the users. Since we plan to implement a `/help` command that displays all available commands, users should be able to utilize functionalities easily since each command would be provided with a useful description. Additionally, since the program was designed modularly, it would not be too difficult to add extra functionalities and commands, thus, we also expect it to be simple to add more functions.

10 Plan

Agile development methodology will be used throughout the development lifecycle of the application. This will allow us to better test each milestone as soon as it is developed. Moreover the use of Agile will enable shorter development cycles, early feedback and continuous improvement.

Below is the initial list of milestones, description and their due dates. We will do our best to stick to the schedule, however, some tasks might be delivered later than other but the project will be delivered on time.

Task	Milestone Description	Due Date
Design	<ul style="list-style-type: none">• Conceptual design of the application• Code Skeleton• Message exchange order/handling• GUI design	Mar 4, 2016
Database/Profiles	<ul style="list-style-type: none">• User database• Chat history database• User accounts• User login/logout• Friend list	Mar 11, 2016
Connection/Chat	<ul style="list-style-type: none">• Chat logic and functionality• Real-time status updates• Chat history/log• Connection/disconnection• List of online friends	Mar 18, 2016
Groups	<ul style="list-style-type: none">• Group chat	Mar 25, 2016