CS3002F: NETWORKS ASSIGNMENT 2018 (STAGE 2)

Tony Guo, Martin Flanagan, Anran Chen

GXXHEN001, FLNMAR011, CHNANR001

Features

Enter key has various features:

Sending a text-message:

When the user is typing a text message, and presses the enter button, the message which is typed automatically sends. This helps the user by saving the time needed to move the mouse to the send button and click send.

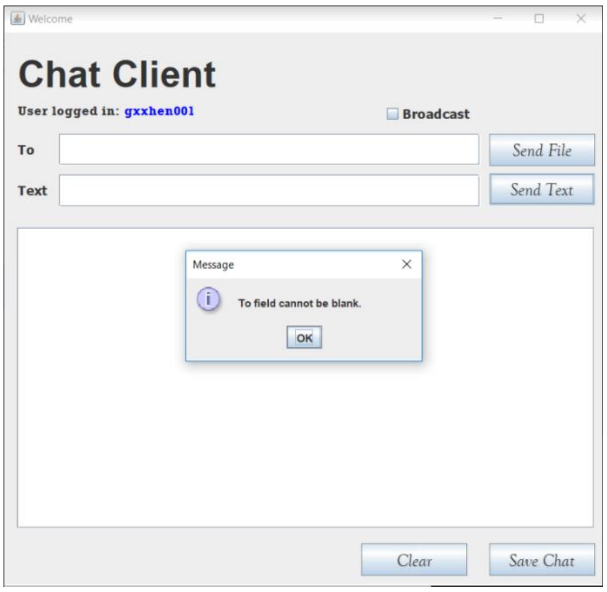
Logging in:

When the user logs in, the enter key on the keyboard can be used as an alternative to the log in button in the GUI.

Error Checking:

Sending a text-message:

When a user sends a text message, the client-side application checks whether the To field is not blank as the To field must contain a value to send a text message.

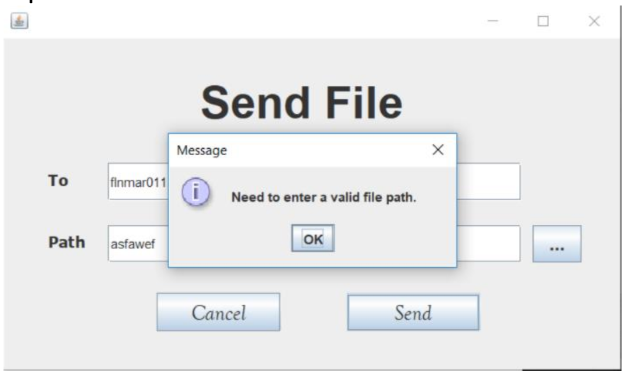


To field cannot be blank.

Figure 1

File transfer:

When the user sends a file, the client-side application checks whether the file specified is a valid file and the file path is correct.



flnmar011

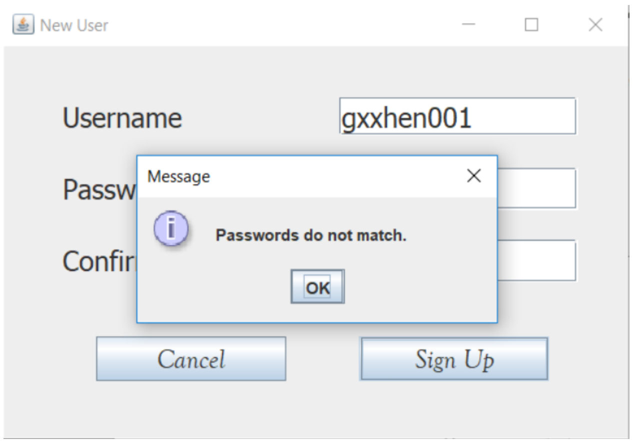
asfawef

Need to enter a valid file path.

Figure 2

Sign-Up:

When a user sign's up, both password fields are checked to see whether they match.


New User

Username gxxhen001

Passw

Confir

Message

Passwords do not match.

Figure 3

Text-Messages:

Multicasting:

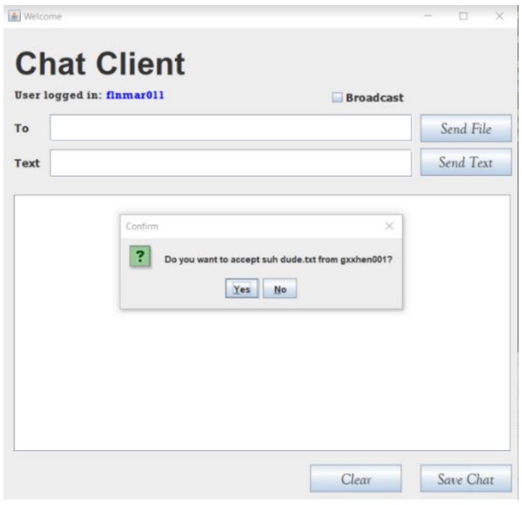
This is when a text message is sent to other users specified by the user (in the To: text field split by commas) sending the message.

Broadcasting:

This is when a text message is sent to all the users who are currently online and does not depend on the user specifying the username.

User can choose to accept or reject a file:

When sending a file, the receiving client must choose whether to accept the file or reject it. If the user accepts it, the file will begin to download, if the client rejects it, the client will not receive the file.


Welcome

Chat Client

User logged in: flnmar011

☐ Broadcast

To

Text

Confirm

Do you want to accept suh dude.txt from gxxhen001?

Figure 4

Clear text:

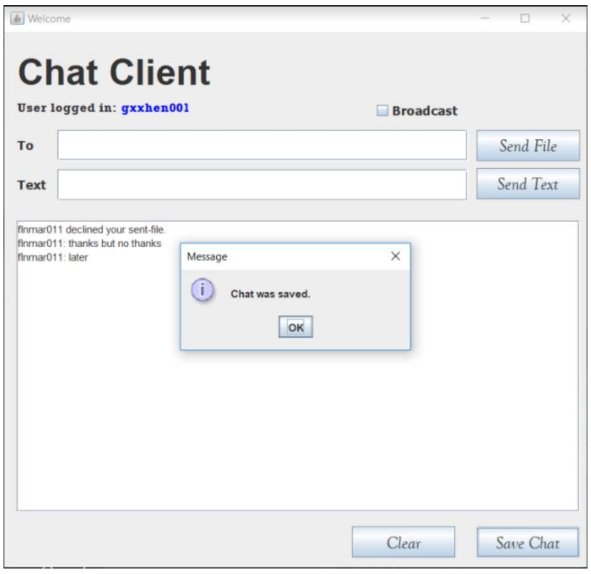
Clients have the option to clear the text in the text area of the GUI. When the button is clicked, the text area will clear all the text-messages that were seen in the area.

Scrolling:

The text area will also be scrollable allowing the client to see all the messages received even if the messages are too long to read.

Save to a text file:

The client can choose whether they would like to save the chat to a text file. It will be saved as their username and the date on which the chat was saved.



User logged in: gxxhen001

flnmar011 declined your sent-file.
flnmar011: thanks but no thanks
flnmar011: later

Chat was saved.

Figure 5

Protocol Design – The MAT Protocol

Message Structure:

Header

MESSAGE TYPE: <message type>|
ACTION: <action>|
TO: <to>|
FROM: <from>|

Body

Data for action of message type. E.g:
TEXT: How you doing?|

Fields are separated by the delimiter "|".
No return/nextline characters as the DataInputStream's readline() method is used.

Hence a protocol message structure would look like:

MESSAGE TYPE: <message type>|ACTION: <action>|TO: <to>|FROM: <from>|TEXT: How you doing?|

Message Formats:

The fields are on separate lines for clarity purposes, and the numbering specify the communication rules for each operation.

Sign Up/New User

1. **Client sends credentials to server to validate for signup:**

    MESSAGE TYPE: Command|
    ACTION: New-Client|
    TO: Server|
    FROM: Client|
    USERNAME: <username>|
    PASSWORD: <password>|

2. **Server sends response back to client regarding its signup request:**

    MESSAGE TYPE: Command|
    ACTION: New-Client|
    TO: <client's username>|
    FROM: Server|
    RESPONSE: <OK/XD/Username is already taken.>|

    OK response – successful
    XD response – unsuccessful
    Username is already taken – unsuccessful

Authenticate

1. **Client sends credentials to server to verify for sign in:**

    MESSAGE TYPE: Command|
    ACTION: Authentication|
    TO: Server|
    FROM: Client|
    USERNAME: <username>|
    PASSWORD: <password>|

2. **Server sends response back to client regarding its authentication:**

    MESSAGE TYPE: Command|
    ACTION: Authentication|
    TO: <client's username>|
    FROM: Server|
    RESPONSE: <OK/XD>|

    OK response – let user sign in
    XD response – incorrect username and password

Send Text Message

1. **Client sends text to another client/broadcasts message via the server:**

   MESSAGE TYPE: Data|
   ACTION: Send-Text|
   TO: <receiver's username/receivers' username/BROADCAST>|
   FROM: <client's username>|
   TEXT: <message>|

2. **Server reads message protocol and forwards it to corresponding client:**

   MESSAGE TYPE: Data|
   ACTION: Send-Text|
   TO: <receiver's username/All>|
   FROM: <client's username>|
   TEXT: <message>|

3. **If server could not find corresponding client, server notifies sender client:**

   MESSAGE TYPE: Data|
   ACTION: Send-Fail|
   TO: <receiver's username>|
   FROM: <client's username>|
   TEXT: <message>|

   <message> - contains failed to send message regarding a recipient
   *Also used in **Send File**

Send File

1. **Client sends file to server:**

   MESSAGE TYPE: Data|
   ACTION: Send-File|
   TO: <receiver's username/receivers' username>|
   FROM: <client's username>|
   FILENAME: <name of file>|
   SIZE: <size of file (bytes)>|

2. **Server stores file and sends corresponding clients permission requests:**

   MESSAGE TYPE: Data|
   ACTION: Permission|
   TO: <receiver's username>|
   FROM: <sender's username>|
   FILENAME: <name of file>|
   SIZE: <size of file (bytes)>|

3. **Receiver sends response back to Server:**

MESSAGE TYPE: Data|
ACTION: Permission|
TO: <sender's username>|
FROM: <receiver's username>|
RESPONSE: <YES/NO>|
FILENAME: <name of file>|
SIZE: <size of file (bytes)>|

4. **Server sends these responses back to the sender:**

   MESSAGE TYPE: Data|
   ACTION: Verdict|
   TO: <sender's username>|
   FROM: <receiver's username>|
   RESPONSE: <YES/NO>|

   YES – receiver accepted file transfer
   NO – receiver declined file transfer

5. **If receiver accepted file transfer, it notifies the server when it is ready for transfer:**

   MESSAGE TYPE: Data|
   ACTION: Ready|
   TO: Server|
   FROM: <receiver's username+"RECEIVE">|
   GET: <sender's username+"SEND">|
   FILENAME: <name of file>|
   SIZE: <size of file (bytes)>|

   FILNAME and SIZE used to find correct file in array

6. **Once file transfer is complete from server to the receiver, server notifies receiver of completed action:**

   MESSAGE TYPE: Control|
   ACTION: Transfer-Complete|
   TO: <receiver's username>|
   FROM: <sender's username>|
   TEXT: File transfer from <sender> is complete.

Justification

We formed our protocol this way to conform to the fragmented message structure of header and body. The header acts as metadata to the body for either the client or server to decipher the message. Our actions were grouped under three message types - covering all cases. Command messages to define initialization – sign up and authentication. Data messages for data transfer – file and text. Control messages for managing control of what is going on.

Sequence Diagrams

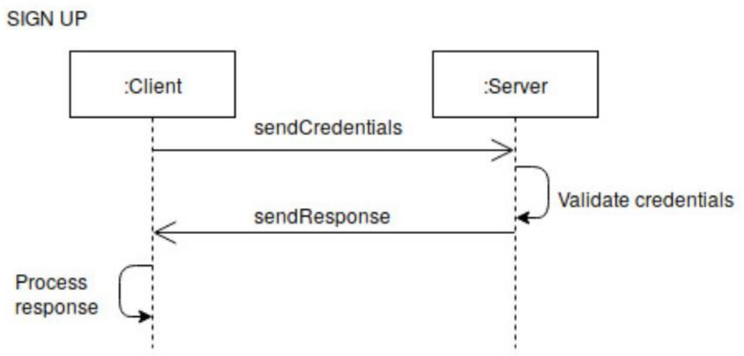


Figure 6: Sign up sequence diagram

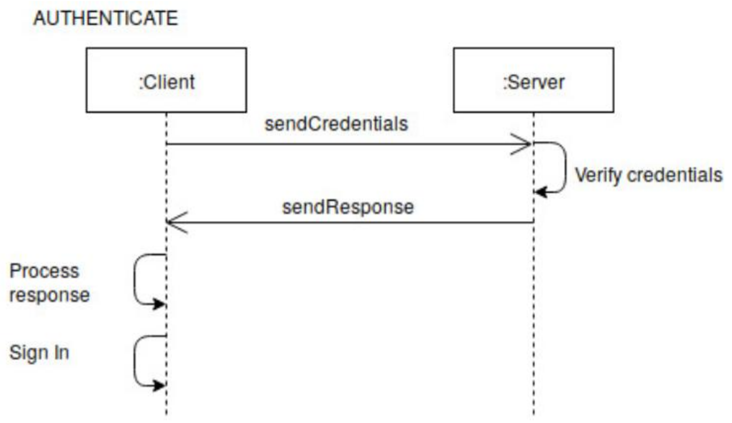


Figure 7: Authenticate sequence diagram

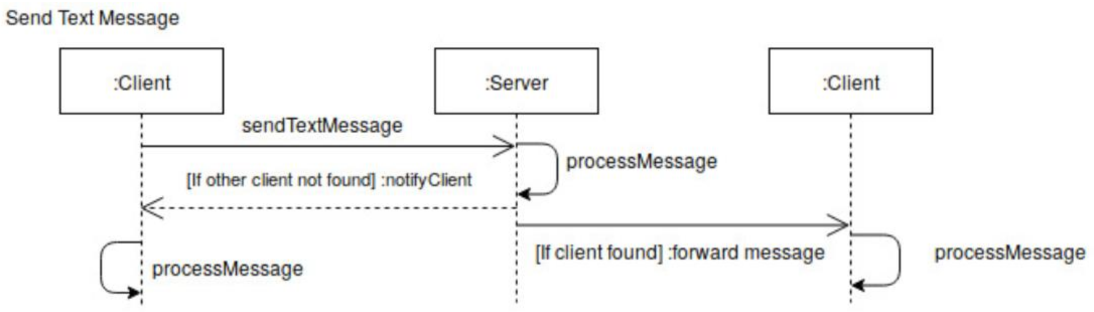


Figure 8: Send text message sequence diagram

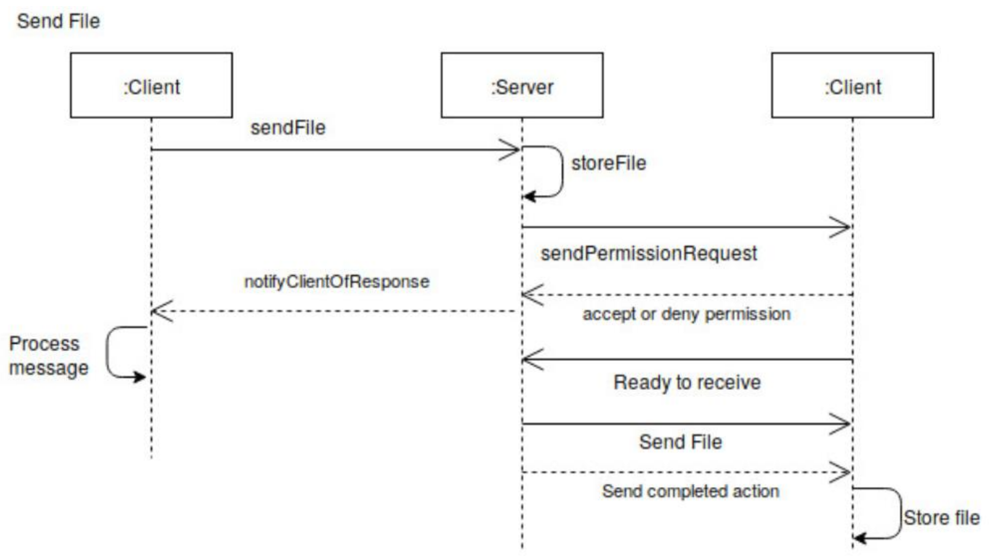


Figure 9: Send file sequence diagram

State Diagrams

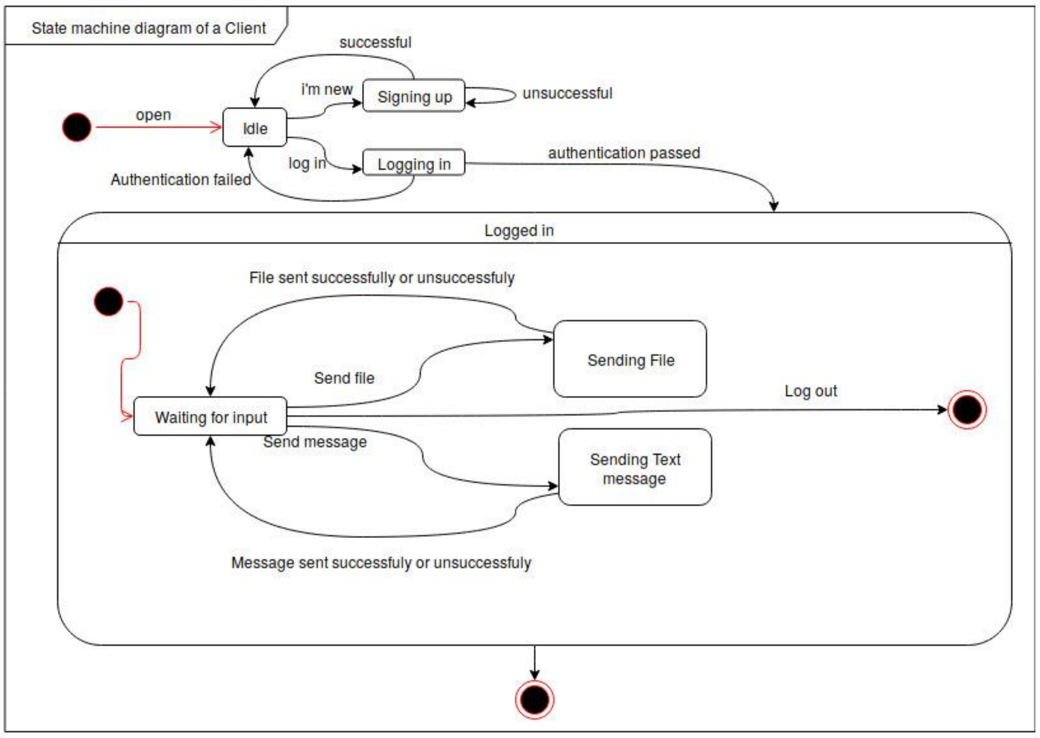


Figure 10: Client state diagram

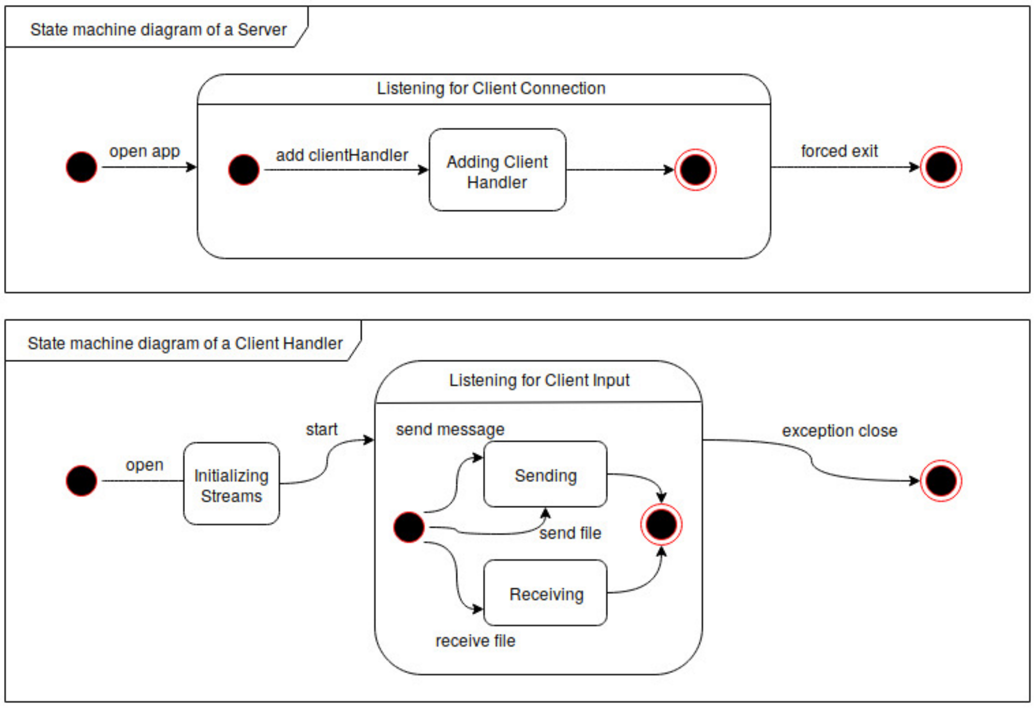


Figure 11: Server state diagram

Tests on Implementation

Many tests were ran to cover all possible scenarios for any exceptions and errors. For the uncovered exceptions and errors, they were catched and catered for - for the smooth running of the application.