Logan Miller

Term Project: *LCAP*

Design Document

Table of Contents

1 Introduction 3

1.1 Purpose and Scope 3

1.2 Target Audience 3

1.3 Terms and Definitions 3

2 Design Considerations 4

2.1 Constraints and Dependencies 4

2.2 Methodology 4

3 System Overview 5

4 System Architecture 6

4.1 Server 6

4.2 Client 7

4.3 MessageManager 7

4.3 UserManager 8

5 Detailed System Design 10

5.1 Server 10

5.1.1 *void startServer( )* 10

5.1.2 *void disconnect( )* 10

5.1.3 *bool createUser(string name, string pass)* 11

5.1.4 *bool checkUsername(string name)* 11

5.1.5 *bool checkPassword(string name, string pass)* 11

5.1.6 *bool sendMessage(string user1, string user2, string message)* 11

5.1.7 *bool displayMessage(string user1, string user2, string message)* 11

5.1.8 *void displayOnline(string user)* 11

5.1.9 *bool userOnline(string name)* 12

5.2 Client 12

5.2.1 *bool logIn(string username, string pass)* 12

5.2.2 *bool createAccount(string username, string pass)* 12
5.2.3 *void disconnect( )* 12
5.2.4 *void getChatHistory(string receiver, string message)* 12
5.2.5 *bool getGlobalHistory( )* 13
5.2.6 *bool displayMessage( )* 13
5.3 MessageManager *13*
5.3.1 *bool saveToFile(string user1, string user2, string message)* 13
5.3.2 *bool checkForFile(string user1, string user2)* 13
5.3.3 *bool createFile(string user1, string user2)* 13
5.3.4 *bool createGlobal( )* 14
5.3.5 *bool displayConversation(string user1, string user2)* 14
5.3.6 *bool displayGlobal( )* 14
5.4 UserManager *14*
5.4.1 *bool saveToFile(string name, string pass)* 14
5.4.2 *bool checkUsername(string name)* 15
5.4.3 *bool checkPassword(string name, string pass)* 15
5.4.4 *bool setStatus(string name, bool status)* 15
5.4.5 *bool getStatus(string name, bool status)* 15

Introduction

LCAP will be a chat application through which multiple users can connect to a server and communicate with each other over the internet. The following document is broken up into multiple sections. The design choices and considerations outline a framework for how the program will interface with a user. Within the following document information on design constraints and considerations will be enumerated, a high-level overview of the system's architecture will be given, and a more detailed system design will be explained.

Purpose and Scope

The purpose of this document is to give a detailed description of the system architecture and design of the LCAP. Through this document the reader should be able to attain a full idea of the capabilities, limitations, and bounds of the software.

Target Audience

This document is prescribed for managers and software engineers who will use the document as a blueprint for the design and implementation of the software. Some readers may wish to use the document in order to receive a detailed view of the software.

Terms and Definitions

- Client (User) - Any person who creates an account on the application in order to communicate with other users
- Global Chat - The stream of inputs that all online users are able to view
- LCAP - Logan's Chat Application
- Private Chat - The stream of inputs between two distinct users
- Receiver - A client that receives a message through either the public or private chat
- Sender - A client that sends a message through either the public or private chat
- Server - The actor that will manage message traffic as well as user privileges
- UI - The user interface, the interface the user will use in order to engage with LCAP

Design Considerations

The following section explain constraints and dependencies that shaped the design choices when designing the software as well as the methodology explaining why certain choices were made.

Constraints and Dependencies

The system requires a user to send messages entered through the standard input stream to be sent to other users using the software. A server should direct all message traffic between users as well as handle saving all user data and information for continuity. Users must be able to create an account, log in to the software, participate in a global chat, open privates chat instances with other users, and request to view message history. The server must save all relevant user data, direct all messages, allow users to connect to the server, supply the user with message history when available upon request, and display all online users. Additionally the software had to be created using Oracle's Java programming language.

Methodology

When designing the framework for LCAP a modified MVC software architectural pattern was implemented. The design for LCAP is one in which the client interfaces with a network through a given UI. The network directs and supplies all requests through fetching and sending data through the use of two additional UserManager and MessageManager classes. This design allows all data to be saved on the machine hosting the server, not allowing any data corruption to be caused on the client's side. The idea is that the client program should only allow the user to interface with the server and other users, no data should be saved by the client. This structure will allow for a distinct separation of responsibilities between the client and server side of the software.

System Overview

This system is designed specifically to allow users to communicate with other users in real time over the internet. Through the use of a server (host) all responsibilities connected to the creation, storage and allocation of data should be given to the server, taking unrequired stress from the client. There are four main objects which will communicate with one another to ensure message transmission and storage is handled appropriately. The four objects are as follows

1. The Server
2. The Client
3. The MessageManager
4. The UserManager

The Server's primary responsibility is to direct message traffic between users as well as to send relevant data to the requisite manager's for storage, as well as sending requests to the data managers in order to fetch requested data by the client. The server also will act as the conduit through which all clients will connect to one another.

The Client's primary responsibility is to interface with the user, allowing a space in which the user can send information to other client's. The client will send all data and relevant requests to the Server.

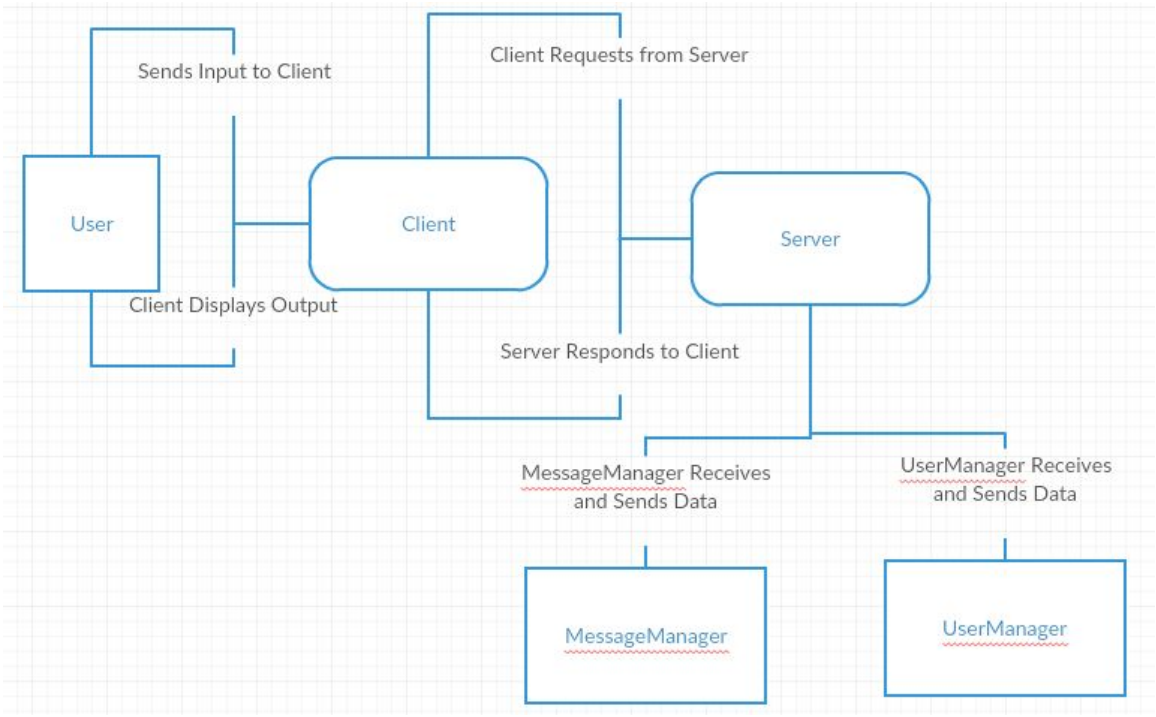
The MessageManager's responsibility will be the storage of user's chat history as well as supplying and saving data supplied or requested by the server.

The UserManager's responsibility will be the storage of all user's account information as well as supplying and saving data supplied or requested by the server.

System Architecture

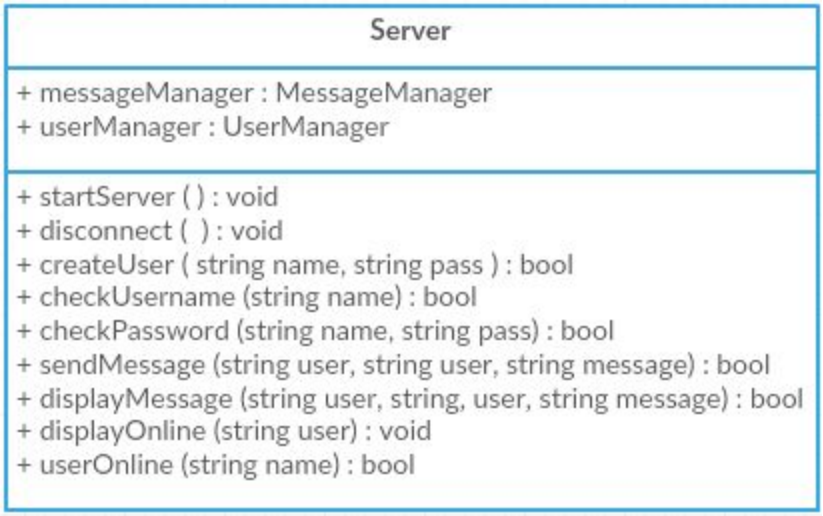
There will be two separate points of entry for the software. For the user hosting the server, they will be able to launch the Server which will be able to manage all relevant data through its two different managers. A user who wishes to use the platform to communicate with other users should launch the software from the Client package. The four subsystems will be the Server, the Client, the MessageManager, and the UserManager.

Figure 4.1 Dataflow Diagram



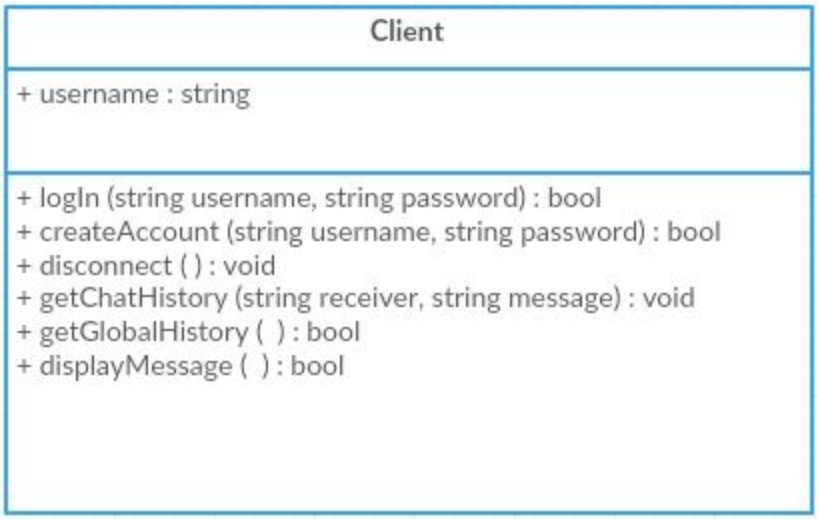
Server

The server class will be the base of the program that handles all traffic between a clients as well as the databases that will save all pertinent information. The server will accept connections, outgoing messages, and requests for message histories from clients. It will send all requests for messages or user account info to the respected repository managers. It can receive messages and user account information from respected repositories and then in turn supply requested information to a client.



Client

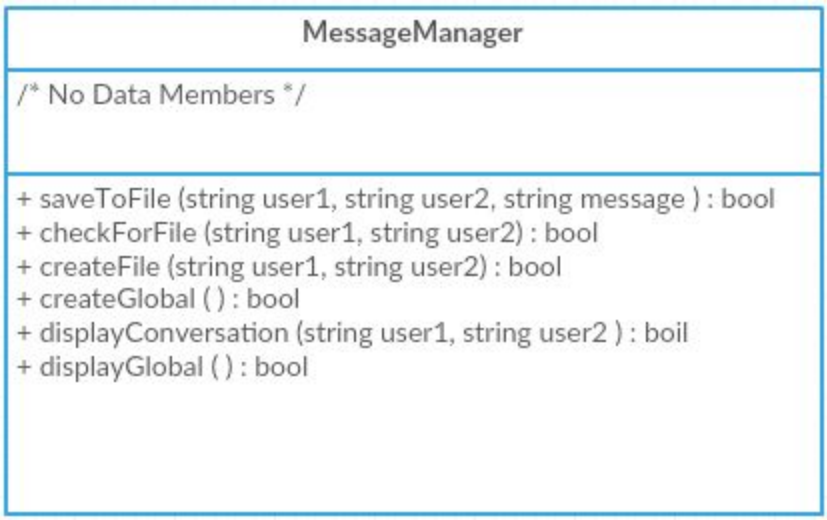
The Client object will act as the sole interface through with a user will interact with the server and other clients. Through this class users will be able to log in to their accounts, create new accounts, send messages to other online users, request their respective message histories, and disconnect from the server.



MessageManager

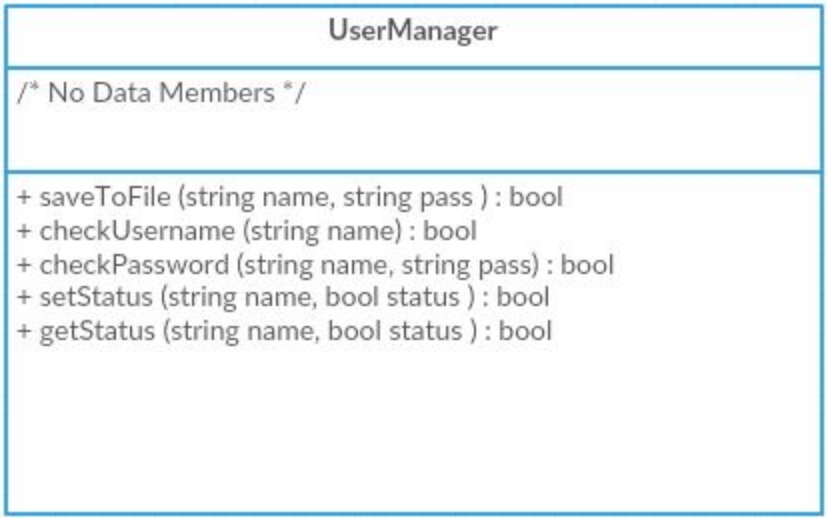
The MessageManager class will act as the sole custodian of all user's message histories as well as the global chat history. It will accept requests for information from the server as well as give the server pertinent messages. It will have the ability to save single messages to an overall conversation history, apend messages to a global chat history,

create a new global chat file, check for required files, as well as send the required messages back to the server. The MessageManager will use local text file storage on the host's machine in order to store all data. This will act as the software's underlying data structure for storing data.



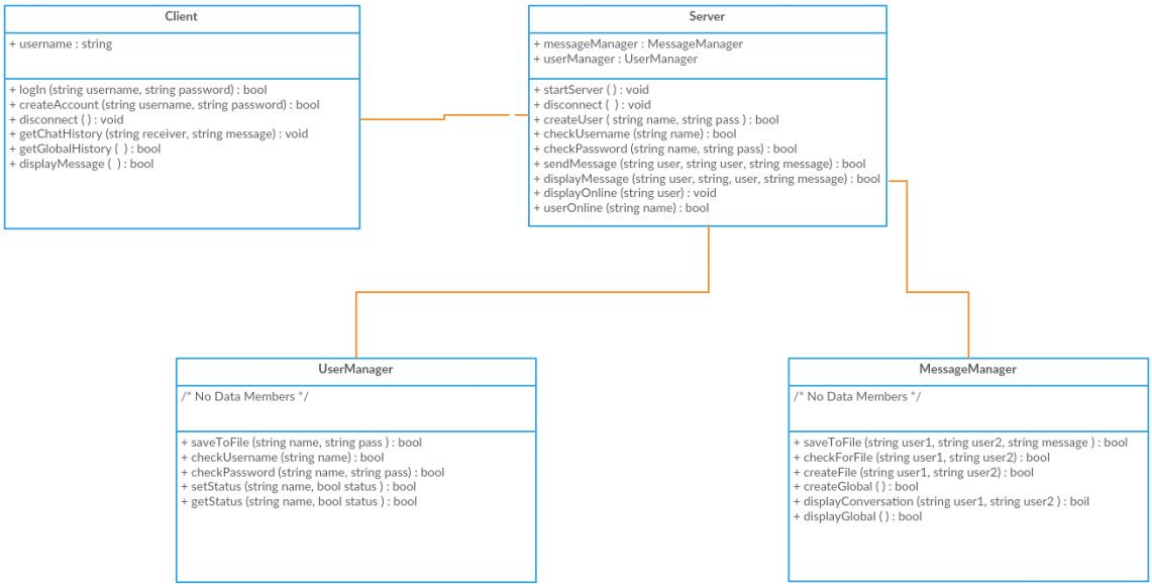
UserManager

The UserManager class will act as the sole custodian of all user's account information as. It will accept requests for usernames, passwords, or username-password combinations from the server as well as give the server pertinent account information upon request. It will have the ability to save usernames and passwords to a text file, check if usernames exist, check username password combinations, set online or offline statues of users, as well as supply the server with the online status of a user. The UserManagerwill use local text file storage on the host's machine in order to store all data. This will act as the software's underlying data structure for storing data.


UserManager
/* No Data Members */
+ saveToFile (string name, string pass) : bool
+ checkUsername (string name) : bool
+ checkPassword (string name, string pass) : bool
+ setStatus (string name, bool status) : bool
+ getStatus (string name, bool status) : bool

Detailed System Design

Figure 5.1 Class Diagram



Server

The server class is an object that will direct message traffic between clients for display, message traffic to the MessageManager for storage, and user account information to the UserManager for storage. The Server will host the connection through which clients will be able to communicate with one another. The Server will have a MessageManager object and a UserManager object through which it will store all data on the host's computer. The object will have the following functions with which to communicate with the other three primary objects.

void startServer( )

First action that the host can take upon opening the server application of LCAP will be starting the server, this will initiate a connection through which clients can connect to the server.

void disconnect( )

Upon the host deciding to shut down the server they may choose to disconnect the server, severing the connection the servers would be using.

bool createUser(string name, string pass)

Takes a desired user name in the form of a string called name and a desired password in the form of a string called pass. Queries the UserManager to create a new account. Returns the success or failure of the account creation to the client.

bool checkUsername(string name)

The server checks if a username is already created. Is given a string called name as an input. Queries the UserManager with the given string. Returns the success or failure of the search for the given username.

bool checkPassword(string name, string pass)

The server checks if a username and password match. Given an input of a string called name and a string called pass, the server supplies the data to the UserManager. Returns whether the username maps to the given password.

bool sendMessage(string user1, string user2, string message)

The server can send a message from one user to another user. Given a string called user1, a string called user2 and a string message. user1 is the the destination for the message, user2 is the source of the message, the message is the text entered into the standard input string to be delivered. The function sends the message to the MessageManager for storage and then send the message to the desired user for display. Returns whether or not the message sending was successful.

bool displayMessage(string user1, string user2, string message)

The server can send a user a requested message history. Given a string user1, string user2 and string message as input. user1 is the destination of the message, user2 is the source of the message, and the message is the log of messages between the user. The function queries the MessageManager for the user's chat history, upon successful return it delivers the message log to the client. It returns the success of the transaction.

void displayOnline(string user)

The server can display which users are currently connected to the server. Given a string called user as input, the server queries the UserManager in order to determine whether or not a user is currently online.

bool userOnline(string name)

The server can query the UserManager in order to determine which users are currently connected to the server. Given a string called user as input, the server queries the UserManager in order to determine whether or not a user is currently online. The function returns whether the truth value of whether the given user is online or offline.

Client

The Client will be the only way with which the a user will interface with other clients. The client is solely responsible with sending messages input from the user in the standard input stream to the server. No data will be saved by the client, the client is only an interface with some functionality for communication and display purposes. The Client will have the following functions with which to communicate with the server.

bool logIn(string username, string pass)

The client can allow a user to login to the server. In order to login a user will have to supply a username and password. The function takes a string username and a string pass. It then supplies the two strings to server. It returns the success or failure of the login attempt.

bool createAccount(string username, string pass)

The client can allow a user to create an account. In order to create an account the user will enter in a desired username or password. The function takes a string username and a string pass. It then supplies the two strings to the server. It returns the success or failure or attempt to create an account.

void disconnect( )

The client can terminate a connection to the server. The function terminates the connection to the server. It has no return value nor inputs.

void getChatHistory(string receiver, string message)

The client can query the server for a user's chat history. the function receives a string receiver and a string message. receiver being the desired user with which to view a chat history. The function sends prompts the server with the given data and the user’s

username. It has no return value.

bool getGlobalHistory( )

The client may query the server for the global chat history. The function has no inputs and returns a the success of the query as a boolean value. The function queries the server for the requested data.

bool displayMessage( )

The client can display messages after a user has sent a message. The function takes no inputs and returns the success of the message being displayed as a boolean value.

MessageManager

The MessageManager is responsible for storing all message histories between users as well as all user's participation in the global chat. The MessageManager will only communicate with the server. The object will be able to receive messages for storage as well as supply the server with messages upon request. The MessageManager will have the following functions allowing it to interface with the server.

bool saveToFile(string user1, string user2, string message)

The MessageManager can save messages to a text file. The function takes two strings, user1 and user2, and a string message. user1 and user2 are the usernames of the two users the message was passed between. The message string is the content of the user's inputs. The file writes the message to the required text file. The function returns the success of the message being saved to the correct file.

bool checkForFile(string user1, string user2)

The MessageManager can check if a file is in existence. The function takes inputs as two strings, user1 and user2. The inputs are the usernames of two users. The function searches for the correct log file for the two users chat. The function returns the success of the search as a boolean value.

bool createFile(string user1, string user2)

The MessageManager can create new log files for saving conversations between two users. Given inputs of two strings, user1 and user2. The function create a file with the

given inputs. It returns the success of the file's creation.

bool createGlobal( )

The MessageManager can create a log file for storing the contents of the global chat. The function has no inputs. The function creates a global chat log file. The function returns the success of the file's creation.

bool displayConversation(string user1, string user2)

The MessageManager can send the server a user's message history upon request from the server. The function receives inputs of two strings called user1 and user2. The two strings are two usernames. The function searches for the correct log file. If found, the function sends the server the message history. The function returns the success of the action as a boolean value.

bool displayGlobal( )

The MessageManager can send the server the global chat history upon request from the server. The function takes no inputs. The function searches for the correct log file. Upon successful search it sends the global chat history to the server. The function returns the success of the search as a boolean value.

UserManager

The UserManager is responsible for storing all data pertaining to user's account information. The UserManager will only communicate with the server. The object will be able to new user's account information for storage as well as supply the server with user account information. The UserManager will have the following functions allowing it to interface with the server.

bool saveToFile(string name, string pass)

The UserManager can save new user's account information to requisite log files. The function receives two inputs, a string name and a string pass. The function saves the two strings to a log file keeping track of all usernames and passwords. The function returns the success of the save as a boolean value.

bool checkUsername(string name)

The UserManager can check whether or not a username already exists. The function receives a string as an input. The function searches the log file for the given string. The function returns whether a matching string was found as a boolean value.

bool checkPassword(string name, string pass)

The UserManager can check whether a username and a password map to each other. The function takes two inputs, a string name, and a string pass. The function searches the log file in order to determine whether a given username and password map to each other. The function returns whether the two map to each other as a boolean value.

bool setStatus(string name, bool status)

The UserManager can set a user's status as either online or offline. The function receives two inputs, a string name bool status. The function searches for the supplied name in the list of usernames. It then sets the status of the user either being online or offline as the given boolean value. The function returns the success of the change as a boolean value.

bool getStatus(string name, bool status)

The UserManageer can search for whether a given user is online or offline. The function receives two inputs, a string name and a bool status. The function searches the log file for a given username. The function checks the status of the given user. The function returns the status of the given user to the server.