Chris McPherson

mcpher3@pdx.edu

916415689

Requirements Document

4/24/17 10:00

Term Project: *JavaMess*

Design Document

Table of Contents

1. Introduction 2

1.1. Purpose and Scope 2

1.2. Target Audience 2

1.3. Terms and Definitions 2

2. Design Considerations 3

2.1. Constraints and Dependencies 3

2.2. Methodology 3

3. System Overview 4

4. System Architecture 5

4.1. Client Program 5

4.1.1. Log in 5

4.2. Server 6

4.2.1. User 7

4.2.2. UsersTable 8

4.2.3. Handler 8

5. Detailed System Design 9

5.1. Client 9

5.1.1. Login 9

5.2. Server 9

5.3. Handler 10

1. Introduction

JavaMess is a distributed chat system written in Java with both a client and server in order to send text based messages over the internet. This document touches on the rational as to some of the design decisions and methodology. It is also description of the system architecture and all of the subsystems that make it up. Each of the subsystems will be examined at the class level with UMLs

1.1. Purpose and Scope

The design document acts as a description and a blue print for the implementation of the client and server subsystems for JavaMess as well as how all the classes interact with each other. From the description of how each class works, it should be possible to implement JavaMess with a set of classes and their relationships.

1.2. Target Audience

The target audience for this document is primarily for stakeholders and engineers associated with JavaMess. However this document may be shared by the original creator for educational purposes for other engineers.

1.3. Terms and Definitions

packets: unparsed strings that may contain messages or commands

2. Design Considerations

Prior to developing a comprehensive plan for JavaMess some design considerations were chosen that shaped the direction of the project. While all of these changes might not be visible in the final product, the decisions helped give the project a sense of direction.

2.1. Constraints and Dependencies

While developing a set of requirements, It became clear that some design considerations would be necessary in order to make this project feasible. One of the early decisions in the process was using the Java language for the project. In order to send messages over the internet, the Java.nio library will be needed to send packets of data and receive them. In order to have a GUI with the client, the JavaFX package will be used.

2.2. Methodology

This project is being developed using the waterfall methodology. At this point the set of requirements has been set and the specific details of how the program will be implemented are contained in this document. Waterfall was chosen in order to have a holistic view of the planning phase from the top down. Once the program is in the implementation stage, the project may move into the agile methodology. Agile is being considered due to it being easily measured and granular nature. However this project involves some unfamiliar libraries which make it difficult accurately estimate the time for implementation.

3. System Overview

JavaMess is a distributed chat system written in Java with both a client and server in order to send text based messages over the internet. The client will be the program most users will use. From the client program users will be able to send and receive messages to other users. Users will also be able to retrieve message history as well maintain a personal friends list. The security of each user will be protected through a login gateway which will require a username and password. When a user wishes to end a session, a log out function will protect a users personal information.

The server subsystem will handle all of the messages, storage and authentication of users, and retain all of the chat histories. All of the users will be stored in an external file which will be loaded when the server starts. Each conversation between users will be stored on the server as an external file in order to have the information available between any possible downtimes between the server. The server will also have two linked list queues in order to send and receive messages. All messages between users will be sent to the server, placed in a queue, copied into the appropriate history, and sent to the appropriate user.

4. System Architecture

The system architecture is the total set of relationship and organization of all the subsystems of JavaMess. The primary subsystems of JavaMess are the client and server components. Both work together in order to have all features available to the user.

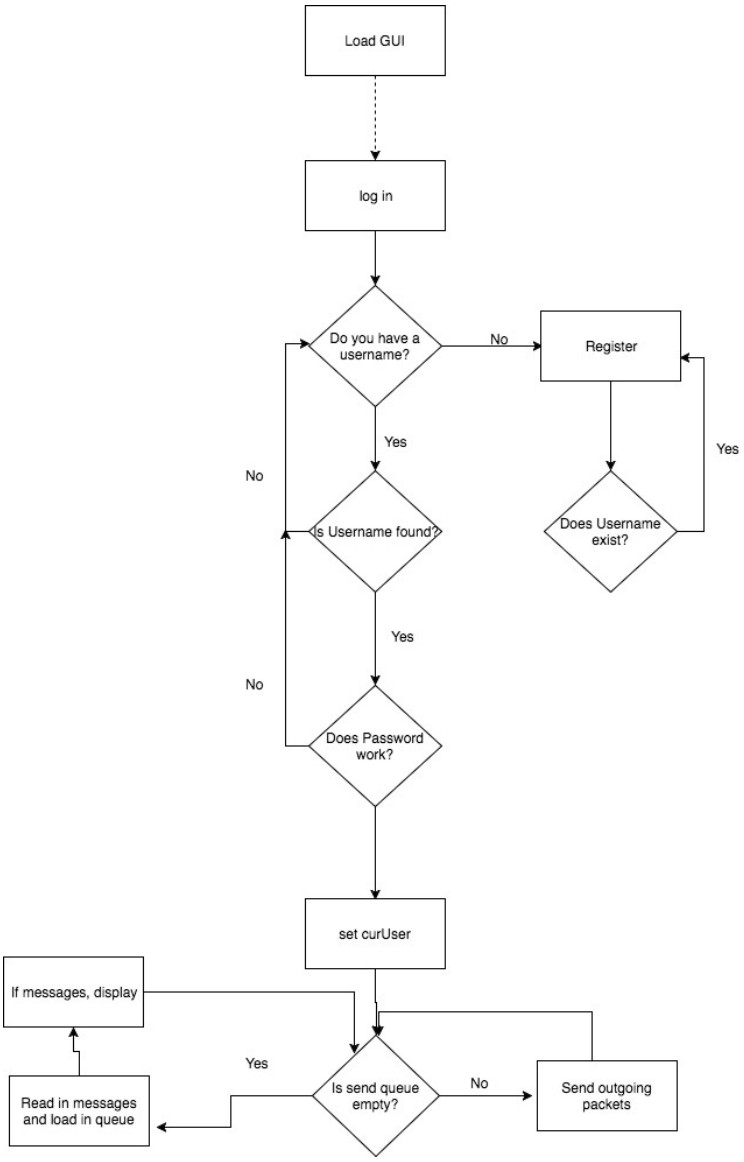
4.1. Client Program

The client Program is where all of the user interactions will take place. As soon as the client program starts running, it will load the GUI. In order for the client program to be used, a user must have an account associated with the program. First time user can create an account through the registration function. If a user has an account, they can log in with their username and password. After the username and password have been matched, The server sends the user information be loaded by the client. A socket is then opened on the client side and all the messages from the server are received.

While the user is logged in, it can send packets to the server. Each packet will be a string leading with a character to denote if it is a message or a command. These packets will be parsed and processed by the server. The packets will contain the sender and receiver with the outgoing message. Simple commands will also be sent the same way. These commands will include retrieve chat history, logout, and delete account.

4.1.1. Log in

The client program will take in username and password from the user through standard in. The username will be sent to server to check for a match. If there is a match, the password is hashed, salted, and then sent to the server for comparison. If the comparison is true, the user object is sent to the client.



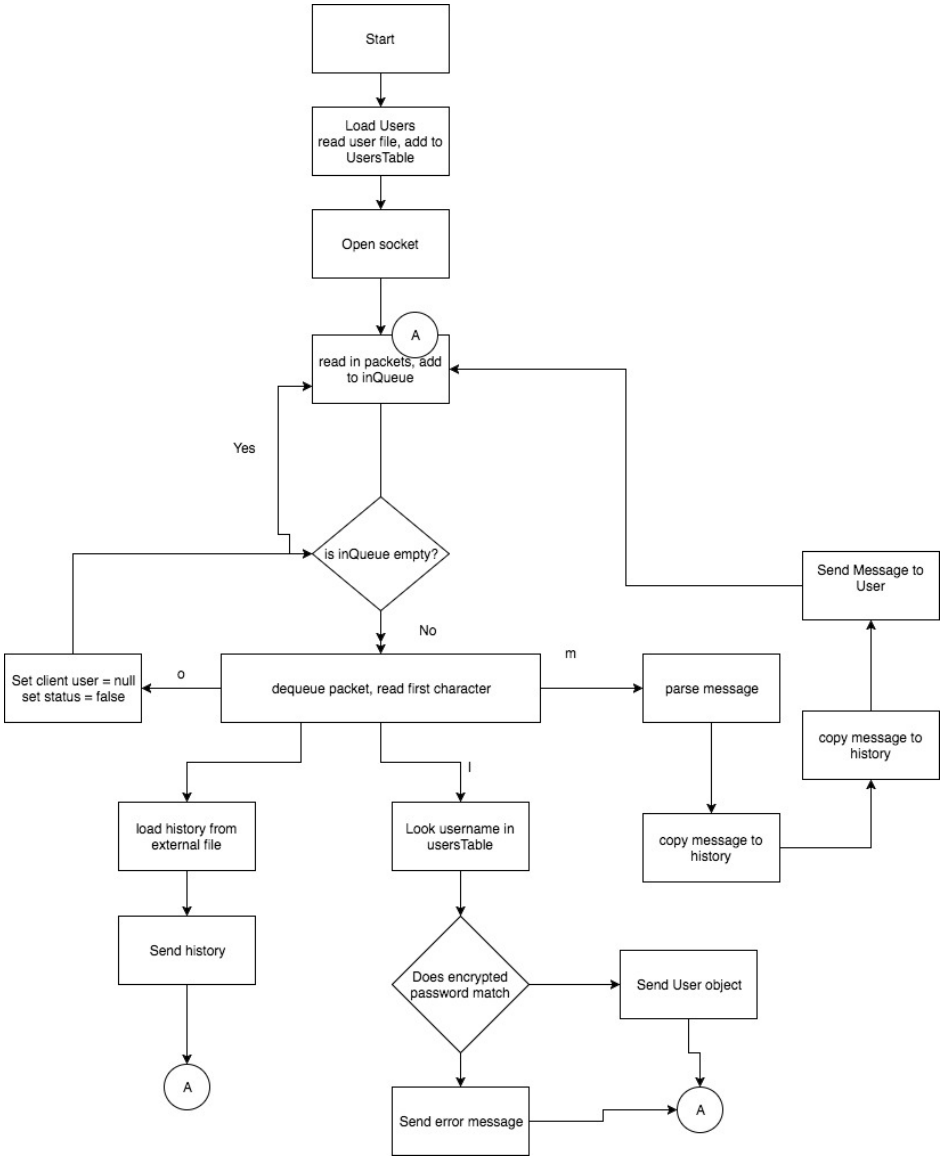
4.2. Server

The server will store all of the needed information and process all of the packets being sent by the client. As soon as the server has started running it will load all of the user objects from an external file into a hash table called usersTable. The server will then open up a socket and read all packets and store them in a queue. Based on the first character of the packet, a switch will execute the appropriate command. A message will be parsed in order to find the senders and recipients, the message contents copied into the conversation file, and sent to the recipient. The load history command will cause the

history file to be opened, copied, and sent to the client. Users may also logout or delete their account using these commands.

4.2.1. User

Each user will have their information stored in the User class. The user class will contain the username, password, friends list, and online status. Each username will be unique and used a key in the usersTable hash table. The passwords of each user will be



hashed and salted using the Java security package. The friends list will be a linear linked list of usernames.

4.2.2. UsersTable

All of the user objects will be stored in the hash table usersTable which will be provided by java.util. The username will act as a key for retrieval.

4.2.3. Handler

The handler will sort packets and call the appropriate function checking the first character in a switch. The packet will either be a message, history request, or a user command. A message or history request is sent to the history function, while user commands were sent to the UsersTable.

5. Detailed System Design

5.1. Client

The client will have the value curUser and outQueue. curUser sets who is logged and from that loads linear linked list of strings for the friends list. outQueue is a queue of strings implemented with a linear linked list. All packets are queued in outQueue and while outQueue is not empty, it sends them out to the socket. While outQueue is empty, the socket reads in messages and displays them in the appropriate window.

5.1.1. Login

The login will mostly be a series of branches to ensure that the user information matches up with the users who are already in memory.

```
If (user == found)
{
        pw = encrypt(password)
        if (pw = user.pw){
        curUser = User;
        }
}
else{
        error message;
}
```

5.2. Server

The server will contain the classes, UsersTable, inQueue, and history. As described earlier, UsersTable will be a hash table of user objects using the username as a key. The hash table will have the add, lookup, and delete functions. inQueue will be a

queue implemented with a linear linked list of strings read in from the socket. The history class will be a data repository class in order to handle an individual history. Each history will have the usernames who are a part of the history and a collection of strings. Histories will be stored in external files.

5.3. Handler

The handler class will be the main processor of all the information for all of the packets. As stated earlier, the first character of a packet will be read by the handler and handled by switch. An ‘m’ packet will be a message and will have a message parser. The parser will break read a ‘*’ limited string to extract the sender any or all recipients. If there is no recipient the message is treated as public. The contents are copied into the history with all matching usernames, unless it is a public message. All public message history is held in the same file. An ‘l’ packet will contain a username and encrypted password. The username is used to look up a user object and then match the password. If successful, the object information is sent back as a string, else null is returned. If an ‘o’ message is sent, the server sets the user to be offline and sends a signal to the client to set curUser to null. If a client sends an ‘h’ packet, it will contain a set of usernames that will be used to look up a history file. The file will be read and have the contents copied into a linked list of strings in the history object. The strings will then be sent to the user.