Molly Shove

Term Project: *Chatterbox*

Design Document

Table of Contents

Introduction

Hello reader, and welcome to the design doc for Chatterbox. In many ways, the design is the most important part of a project, and doing it right can save 100's of hours and millions of dollars (depending on the scope of the project.) This should break the system down into easily understood chunks which will illuminate how exactly the program will fit together.

Purpose and Scope

This document is intended to inform the reader of the current design setup for Chatterbox. This is subject to change as even the best laid of plans can go awry, and in software, doubly so.

Target Audience

The target audience for this document are two very important people: the TA for CS 300 and the one who will be implementing the program (me).

Terms and Definitions

Netty is a client/server framework I will be building off of for this assignment. Additional information about netty can be found following this link.

Design Considerations

Below are some considerations for the design of Chatterbox. A few general rules of design which are relevant to this project is the desire to use existing tools to the best of their ability, to document my code extensively and to test features regularly.

Constraints and Dependencies

The two main design constraints for this project are time and money. Frankly, I have one month left to build this system and a budget of $0. Thankfully, there are a lot of good, free tools out there which will help make sure I can finish this project in both temporal and financial scope

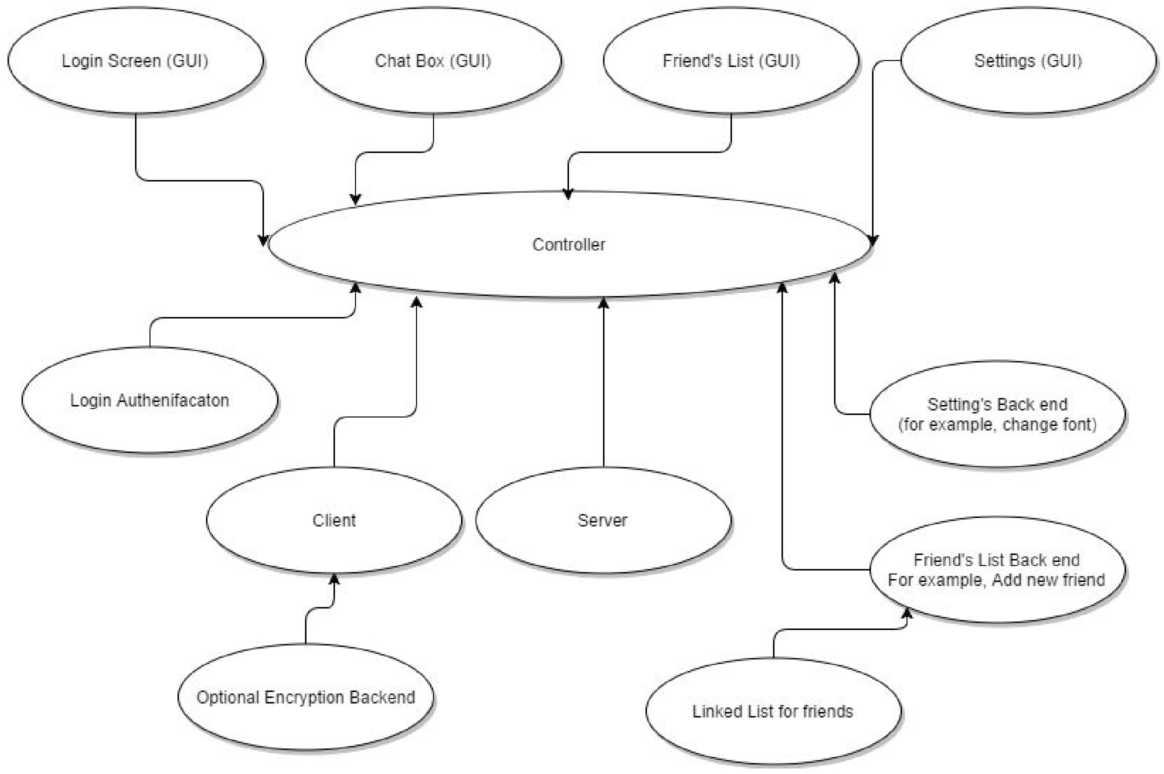
Methodology

The goal is always to create extendable, testable modules which make the addition of new features easy. Thankfully, I don't have to do all of the work myself as Netty is a solid framework which I can build off to add the features I need without having to recreate the wheel. This brings me to a second directive: don't build anything that has been built better somewhere else. I plan to take full advantage of the libraries that already exist. Using already tested code guarantees more efficient, reliable code with better documentation then I am currently able to create. I want to keep it simple and clean as possible.

System Overview

So, the system has 3 major layers: the GUI, the controller and the backend/helper functions. Each of the UI functions will call a controller function written to do the thing which the button on the UI is intended to do. The controller will instantiate all of the necessary objects and organize the self contained tools into instructions the user wishes to carry out. In order to build the application, I will have several self contained backend modules which each do a specific task, or organize a specific part of the app. This include the client, the server, the friends list, the login authentication and a settings class. Each of these will be testable in relative isolation, which should ensure a clean and relatively bug free construction. A diagram is below for your viewing convenience.

Figure 1.1



System Architecture

In short, a chat system only has a few main components. A client, which can send and receive messages, a server, which can store and relay aforementioned messages and a UI to make the client look pretty for the user. There are many important features beyond the scope of these three (for example, a list of friends or other clients the user can chat with, or the secure login backend) but those three still make up the core of a chat server. Let's jump in.

The Controller Class

The controller class is the one which goes between a user and all of the other classes in the system. It has two types of functions, those which test the tools which it relies on, and combinations of functions used to create the direct features of the program. While it's largely a caller of other functions, it provides a unified place to see and understand how the program is functioning on it's highest level.

The Client Class

The client is the tool which sends user messages to the server and receives messages written by other users from the server. The client should have a function which runs in the background receiving messages so long as the user is logged in.

Optional Client Encryption Function

In order to keep the message secure, encrypting the message before it's sent to the server, and decrypting messages received would be a neat feature. This would require a function to take user input and garble it, and a separate function to decrypt the message.

The Server class

The server is like the central processing unit for all of the messages from all of the users. In a larger system, I would have to figure out now to scale up the program with more than one server. Thankfully, this project is small in scope. Regardless, the server needs to send and receive messages, and make sure everything is in the right place.

The User Interface (GUI)

This will consist of a variety of pages, including the following: A login screen, a chat window, a list of friends who can be chatted with, and a settings screen. All of these will be able to navigate and help the user do everything they need to do. The GUI will always call controller functions, so the hierarchy of the program design is simple. User talks to UI, UI talks to controler. Controller talks to client, server and any other helper and organization classes needed to make the program run.

System Design

The Controller Class: What it really controls

The controller class will have the following functions

- A logout function which cleans up any processes that need cleaning up and ends the program.
- A function which adds an additional user to the current chat.
- A function which adds a new friend to the friends list.
- A function which allows the user to type in something and send it away.
- A function which automatically checks the server upon login for any new messages, and periodically checks for more throughout the time logged in.

Friend's list

The controller will have access to a linked list of friends (known clients) which a user can add. The friends data will have a username, what client it's associated with and a jepg icon to represent the user. While the friend's list is it's own class, it's a small enough class to organize it as a collection of controller helper functions.

Logout

This will end the processes of the client and server, and return the user to the login screen.

Setting's class

This will contain a few functions which adjust some of the basic GUI settings. For example, this allows the user to change the font size, and whether or not they wish to appear online for others to see. This is also the place where a use can change their listed personal information

Login Authentication

This will contain a hashtable of existing users and passwords. If the user matches the password, the UI will show the chat window, friend's list etc. Otherwise, it will prompt to enter the password again. There will also be a “create new profile” function which prompts for user information and stores the new username/password combo into the hashtable.