Zoey Lee

Term Project: *TalkyTalky*

Design Document

Table of Contents

1 Introduction

The introduction of the Analysis and Design document provides an overview of the entire document with purpose, scope, target audience, terms, design consideration, system overview/architecture, and detailed design for the Chat Application. The aim of this document is to gather and analyze and give an in-depth insight of the complete Chat Application named **TalkyTalky** by defining the problem statement in detail. Nevertheless, it mainly concentrates on detailed designs and major components of **TalkyTalky** on method level. The implementation details of the methods of all the components of TalkyTalky, as defined in the Requirement Document, are presented in this document.

1.1 Purpose and Scope

The purpose of this document is to present a detailed description of the implementation of TalkyTalky chat application described in the Requirement Document. TalkyTalky is designed to create a place for users to communicate each other on the Internet in real time. The scope of the Chat Application includes its features such as:

- the server provides connections between users and each user can message to either a specific user or all other users.
- the new users must register before using the application.
- the users must log in before communicating with the server or other users.
- the users can check the history of chat for their own.
- the server informs all online users when a user login or logout.
- all users can check who is online or offline.

1.2 Target Audience

The target audience for this Requirements Document includes individual users and stakeholders. Specifically, users shall be the ones who are willing to communicate with other people through Internet in real time. As described software is implemented using Java, the reader should have some knowledge of Java programming language.

1.3 Terms and Definitions

Term	Definition
Talker/Use/Client	Any person who is using the application and is communicating with others over Internet in real time.
Server	A program that provides connections between users and delivers messages from one user to others.
Stakeholder	Any person with an interest in the project including developers.
Use Case	A list of actions or event steps, typically defining the interactions between a role and a system, to achieve a goal.
Socket	One endpoint of a two-way communication link between two programs running on the network.
Port	A connection point or interface between a computer and an external or internal device.
TCP/IP	The basic communication language or protocol of the Internet. It can be used as a communications protocol in a private network.
GUI	Graphic User Interface. Type of user interface that allows users to interact with electronic devices.

2 Design Considerations

This section is concerned with performance related issues, and the constraints and dependencies of TalkyTalky.

2.1 Constraints and Dependencies

- The basic TCP/IP protocol will be the only one used to access the application.
- The web browser will be the primary client used by users.
- TalkyTalky must be able to operate on many different type of computers.
- At this stage of the project, a mobile app version of TalkyTalky is not supported.
- Registration for the application is the first step to get started with TalkyTalky.
- Messages are delivered by the chat server between online Talkers only.

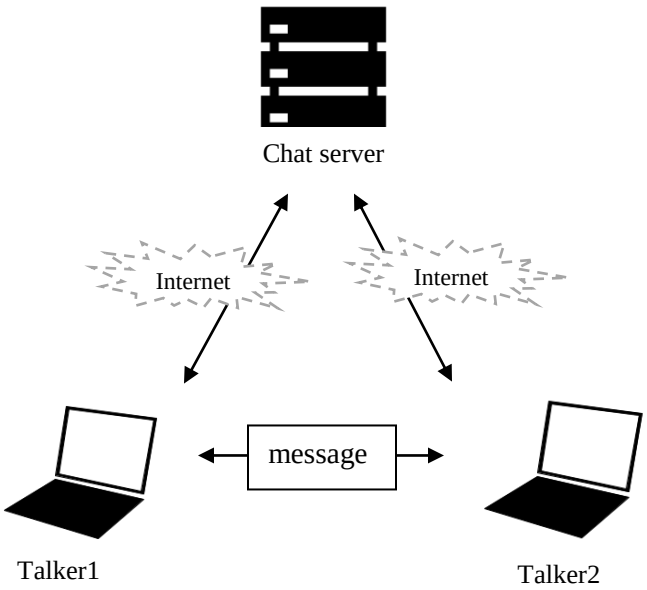
2.2 Methodology

The Waterfall Methodology will be used for this project. The project will be completed in phases, the analysis phase comes first, then the design phase, followed by the implementation phase, and finally by the texting phase. This methodology is suitable since this project has low complexity and requirements are static. The methodology will be implemented by documenting requirements of the application based on customer's demands and then performing this project stage by stage. The main disadvantage of the Waterfall methodology is that once an application is in the testing stage, it is very difficult to go back and make changes in the concept stage. Therefore, each phase must be completed perfectly for the next phase to begin.

3 System Overview

This section is to introduce and give a brief overview of the system. TalkyTalky is a chat application that consists of a chat server and an indefinite number of chat users called Talker(s). It provides a tool to manage and deliver messages between Talkers in real time. For more description of TalkyTaly and the relevant background, see the Requirement Document.

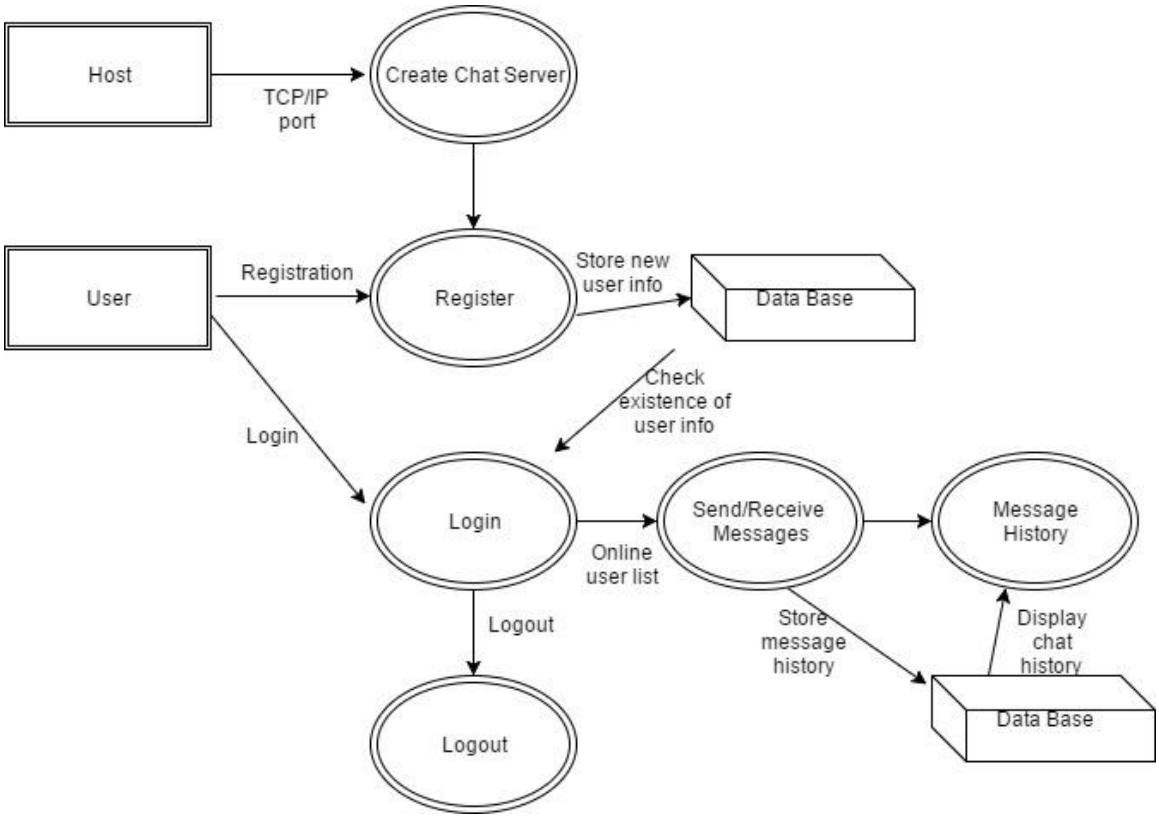
Figure 1 shows the model of TalkyTalky. Each Talker gets connections with the chat server running TalkyTalky by inserting TCP/IP port of the server. Messages between Talkers are delivered by the chat server in real time as long as they are online.



[figure 1] Chat server and users of TalkyTalky

4 System Architecture

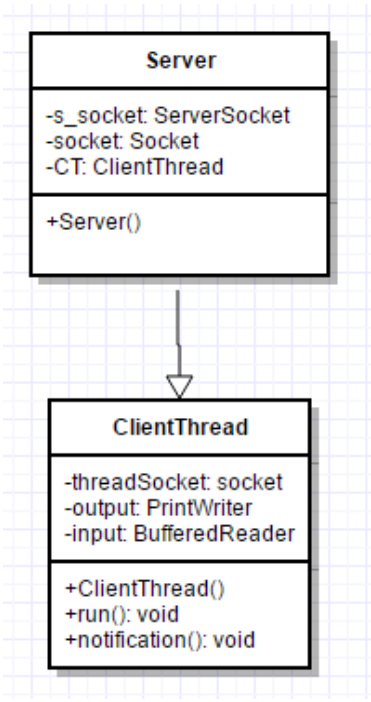
The system architecture is a way to give the overall view of TalkyTalky and to decompose of the application components. TalkyTalky is structured to partition tasks between a server and clients regarding to their perspectives on the system. So, there are server-side system and client-side system as main systems. Then, for each system there are subsystems and they describe the relation to external interfaces and the dependencies on other components. Figure 2 describes the data flow diagram of the entire system.



[figure 2] Data flow diagram of the entire system

4.1 Server side

A server, the provider of a service, runs a server program which share their resources with clients. The following subsections detail server's tasks and components to implement each task. Figure 3 shows the relations between server side components.



[Figure 3] UML diagram of Server side

4.1.1 Class Server

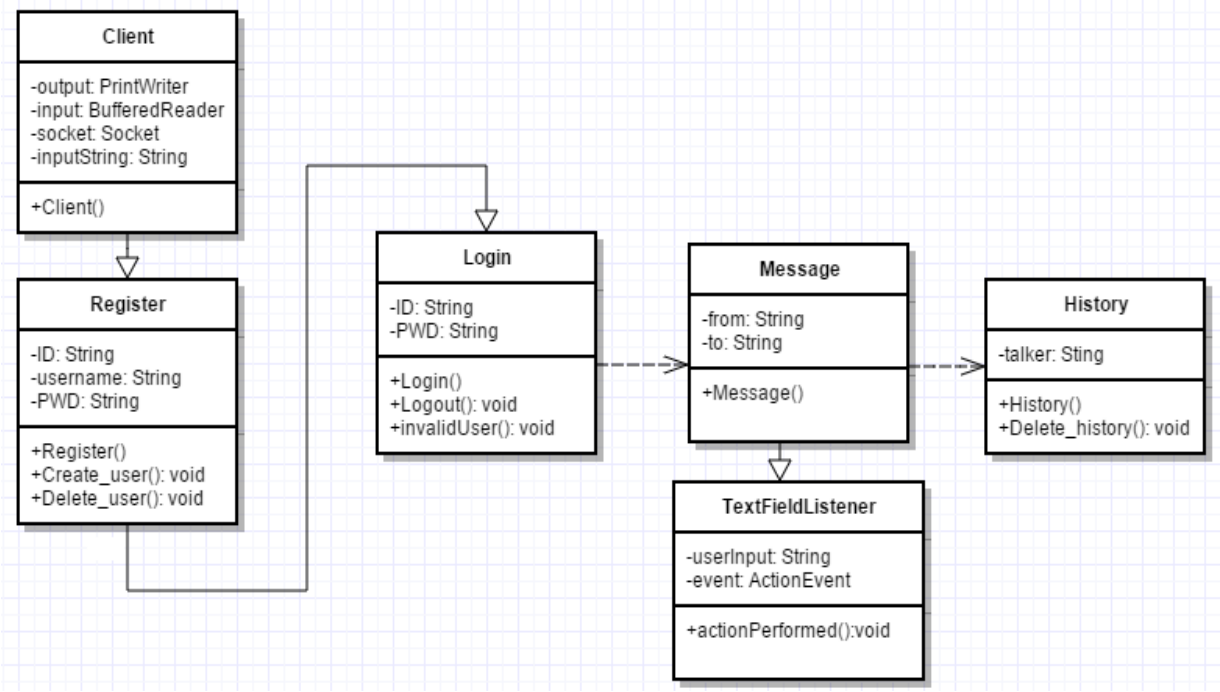
A server is built using ServerSockets in Java and a port number assigned by the program developer allows connection through it so that clients are able to connect to the server with the port number. To accept indefinite number of clients, server functions are written in a while loop. Swing GUI is used to create a user interface for the application such as setting up a layout for window, not allowing editing displayed texts and add scrolling.

4.1.2 Class ClientThread

Class ClientThread is derived from Class Server and implements Runnable so that this class can be used as a thread. Once a client connects to the server, it notifies that the connection is successful. Notification function in this class notifies all online users whenever a user logs in or out.

4.2 Client side

A client, the requester of a service, requests a server's content or service rather than share its resources. Therefore, each client initiates communication sessions with a server while the server waits for incoming requests. The following sections detail client's tasks and components to implement each task. Figure 4 shows the relations between client side components.



[Figure 4] UML diagram of Client side

4.2.1 Class Client

A client application is built to connect to the server, which is built with the port number, using Socket. To allow running the program between two or more computers or devices, IP address of the server computer is required in the Socket. Swing GUI is used to create a user interface for the application such as setting up a layout for window, not allowing editing displayed texts and add scrolling.

4.2.2 Class Register

Class Register is derived from Class Client and it is required for all new users to start with TalkyTalky. New users are asked to insert unique ID, username, and password when

registering for the system. Then, the information of a new user is compared to existing information in data structure and then stored into the data structure of the user list.

4.2.3 Class Login

Class Login is derived from Class Register and has Class Message. A user is asked to insert their own ID and password to access to TalkyTalky. If inputted ID and password don't match correctly, the user is not signed in and has to try again until they match to the existing information in the data structure. Thus, this login process is written in a while loop. Logout process is implemented in a logout function by initializing ID and password to NULL and sending the user back to the login page.

4.2.4 Class Message

Class Message has Class history and only users logged in to TalkyTalky have an authority to communicate with other users. There is one chat box for all online users to join. Once a user clicks another user's username on the list of online users, another chat box pops up for those two users. Other users besides those two are not accessible to see or join the conversation.

4.2.5 Class History

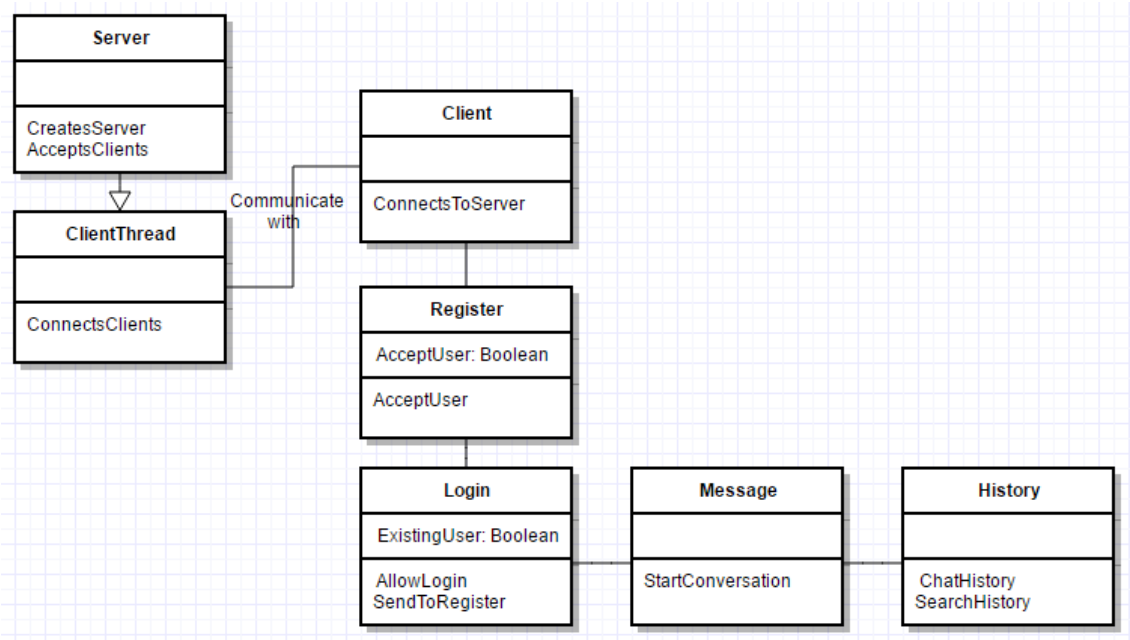
Class history has a data structure for chat history and it will be implemented with an array of linear linked lists where each index of the array is dates of conversation and linked with a list of chat boxes on the specific date.

4.2.6 Class TextFieldListener

Class TextFieldListener is derived from Class Message and implements ActionListener so that it can handle the text field. Whenever the user hits "enter", each line of texts is displayed in the chat box. The text is displayed in both, if one-to-one conversation, or all, if one-to-many conversation, users' chat boxes simultaneously.

5 Detailed System Design

This section contains the highly detailed description of subsystems, components or objects for TalkyTalky. Each subsystem includes pseudo code for a better understanding of the algorithm. Figure 5 is the detailed class diagram for TalkyTalky case study. For clarity, only those methods that cause an object to change its state are shown.



[Figure 5] Detailed class diagram for TalkyTalky

5.1 Class Server

Creating a server and accepting clients processes are written in try-catch block since there are lots of possible errors thrown. When creating a server, a port number assigned by the programmer is passed in and the port number allowing clients to be connected to the server is 9292 for TalkyTalky.

```
Create a server using ServerSocket with port number 9292
While a new client access to the server
        Accept a client using Socket
        Create a client thread to make connection
```

5.2 Class ClientThread

This class is used as a thread between clients and the server and the server has an access to see conversations between all users. Whenever a user is successfully connected to the server, it notifies the user about their status.

```
Notifies each user when the connection is successful.
While a client types texts and enter
        Create a string that reads each line
        Display each line on the chat box
```

5.3 Class Client

Creating a client and accessing to the server processes are written in try-catch block since there are lots of possible errors thrown. When connecting to the server, the port number 9292 and the host's IP address are passed in.

```
Create a client using Socket with port number 9292 and the host's IP address
```

5.4 Class Register

Registering for the application is written in a while loop since it keeps comparing inserted ID with the existing information in the data structure of user list. If a new user tries to create the same ID from the data structure, it fails to register for the application and they are asked to insert a unique ID. The data structure for user list is implemented in a binary search tree with alphabetically sorted order so that searching for matches takes log(N) for N number of users in the list.

```
While a new user doesn't insert a unique ID
        User tries another ID/username/password
        Compare if the ID is unique
The information is stored into the data structure of the user list.
```

5.5 Class Login

Login to TalkyTalky is written in a while loop since a user's information is supposed to exist in the binary search tree of user list to access to the system. Then, the program compares inserted ID and password with the data structure if they match correctly. It is required for users to input the right ID and the right password for the application.

```
While a user doesn't insert a correct ID and password
        User tries another ID/password
        Compare if they match correctly
Send the user to the main screen.
```

5.6 Class Message

Displaying conversations requires reading inputs and printing it out on the chat boxes in users' screens simultaneously. Any input from a user is stored into the data structure of chat history and the data structure is an array of linear linked lists. This data structure is needed for each user so that they can access to their own chat history whenever they want.

```
While a user inserts texts with enter key
        Each line is displayed on participating users' screen.
The entire conversation is stored into the data structure by
linking with the today's date.
```

5.7 Class HIstory

Checking chat history is written in a loop in order to iterate the array of linear linked lists. The chat history is sorted by dates so that it's easier to go back and check the conversation happened on a specific date.

```
If a user clicks chat history
        While a content in the data structure of chat history is not NULL
                Display chat history sorted by dates
```