Team ABC

ECSE 428 - Software Engineering Practice

Proposal

Pierre-Alexandre Bastarache-Roberge – 260326732
Julian Cooper - 260246648
Christian Gallai – 260218797
Armen Kitbalian - 260232915
Iain Macdonald - 260270134
Farzad Towhidi - 260323083

Introduction

Since the creation of the Internet in the early 90s, it has caused a shift in the priorities of computer users. While computers had previously been used as a productivity tool, largely for work, the Internet promoted the use of computers as a communication tool. As a result of these new priorities, development of electronic mail protocols led eventually to instant messaging protocols. The need for instant communication with peers, similar to telephone conversations was driven forward by the demands of users. Now there are dozens of messaging protocols in widespread use.

Motivation

The need for instant communication with one's peers has become a central component of many Internet services. Google Mail, one of the most popular public mail servers, provides a chat protocol as well. Facebook, the largest social networking site also has an embedded chat protocol.

A web-based instant messaging protocol is a software engineering project which is very extensible, with many possible features. We would like to design and implement a web-based chat protocol using the scrum method of software engineering. Such a system has many possible extensions, allowing us to practice the scrum method in implementing these features.

Our messaging protocol will allow users to communicate instantly with their friends through a web browser. The system will focus on communication of information as the primary motivation. On this note, we will implement several features to allow users to more efficiently communicate with one another using the system.

Architecture

A chat protocol can be implemented using a client-server, or a peer-to-peer architecture. We will develop the system using a client-server architecture. All chat users will act as clients, reporting to a central server, which will gather and distribute communications. Using polling, clients will be able to retrieve messages directed to them.

We will use the Turbogears web framework for development and integration of the system. Turbogears permits rapid web development using many different technologies. The framework is largely written in the Python programming language and allows SQL-based databases, with HTML and CSS styled web pages. Turbogears will allow us to focus our attention more on the engineering aspects of the project, without getting caught up in the minor technical details of web server configuration, database integration, and web socket implementation.

Requirements

Functional Requirements

The system should permit users to register and log in. These should both be possible through a simple user interface. In registration, users should be required only to submit a username and password. E-mail verification is unnecessary, as are any personal information, for the first scrum iterations of the system.

After logging in, the user should be able to view their list of buddies, which is presented in an easy-to-read and easy-to-use format. From this buddy list, users can begin conversations with buddies, which will appear in the browser window. Users should be able to begin conversations with many different buddies simultaneously. Additionally, users can begin conversations with random users. This feature has motivated the development of sites like Omegle and ChatRoulette, where users can go to talk to a stranger.

For users who do not wish to make an account, they can log in as guests. Guest users may not have any buddies, for there is no memory of their use of the site. These users may only participate in random conversations. All random conversations will be paired on a first-come-first-serve basis. Those waiting to participate in a random conversation will be placed in a pool, and paired up as others become available.

When a chat is taking place, rather than opening a socket-to-socket connection between peers, or client and server, all messages will be sent to the server, and retrieved by the other person through polling. When in conversation, at specified intervals, the client will poll the server, to see if there are any new messages. The server will reply with the new messages. This system is simple to develop, but will work well for a small user base. By integrating the conversation feature with a database, logs can be kept of old conversations, allowing users to resume conversation in the event of an unexpected disruption.

Often in chatting, individuals share pictures, or videos. The common way to handle this is to present a link to the user. However, it is much more convenient to embed the image or video directly in the chat window. Our system will take this approach, embedding recognized image formats and Youtube movies directly into the chat window. This will allow clients to more easily communicate multimedia. This will be done using a URL recognition feature, and custom embedding code.

In addition to sharing images and videos, users often communicate using emoticons, such as the smiley face - :). In plain text, these often resemble the intended emotion. However, many chat protocols replace these emoticons with custom images. Our system will take this approach, for improved recognition of emoticons.

While many chat protocols offer person-to-person chat capabilities, only a handful support chat room communication. Our system will implement chat rooms. Using a separate portion of the window, a list of chat rooms can be viewed, and then rooms selected from that list to join. When in a chat room, it appears just as a standard chat window, however, there are multiple clients connected, all communicating with one another.

Chat rooms often require administration, to maintain a respectful tone of conversation. Our system will denote one person as the administrator of the chat room, giving him or her the power to kick, ban, and mute certain users. This feature will allow mediation of chat rooms, and maintain a respectful tone of conversation.

While many use chat to communicate with one another, some abuse the privilege. Our system will support a number of features to enhance the chat experience. When a user sends multiple messages in a short time frame, commonly known as “flooding,” it can be overwhelming for the user on the other end. This behaviour will not be permitted in our system. Users who flood will be blocked from chatting for a specified period of time.

In addition to flooding, some users may receive unwanted contact from other users. To resolve conflicts of this sort, our system will allow users to block chats from certain other users, of their choice. This will allow our users to avoid chatting with individuals they feel uncomfortable with. This seemingly minor feature can make a major difference in the success of a chat protocol, as it is often underestimated how serious some of the harassment can be.

Much of a chat protocol is based on the principle of communication. While most of the communication is person-to-person communication, there is often a component of broadcast based communication in chat clients. By allowing users to post statuses, they can broadcast messages to all of their buddies instantly. This feature is commonplace in almost every major chat client, and is what drove the development of microblogging platforms like Twitter.

In any system with user accounts, the ability to manage ones account is necessary. As the system advances, we may like to add additional user information into the system. Users must be able to update their information as they see fit. A method for doing this, such as a simple pop-up form, is required.

Nonfunctional Requirements

The system must be easy to use. The success of any chat protocol is largely based on its ease of use. The commonly used features must be made accessible, and complicated, bloat features must be removed.

The system should be responsive. A chat protocol is not very useful if there is high system latency. Responses from the server must be fast, and polling should take place at a sufficiently small interval that the user is not left waiting for any messages.

In order to best serve users, system maintenance should not require downtime. The system should be online a high proportion of the time.

User Stories

Backlog

1. Tim wants to be able to use chat with his friends. Tim visits the webpage and selects the option to register a new account. Tim enters a username, password, and confirms his password. Tim is redirected to the home page. Tim's username is unique to him.
2. Joe should be able to log in to use the system. Upon visiting the site, Joe is prompted with a log in page. He enters his username and password and is directed to a log in root page, which contains his buddy list.
3. Joe has logged on successfully and wants to see which of his friends are online. Upon logging in, Joe is redirected to a log in root page, which contains a list of his buddies.
4. Tim has logged on successfully and wants to message his friend Joe. But Joe is not currently online. Tim is alerted that Joe is not online.
5. Joe has logged on successfully and wants to message his friend Joe. Joe selects Tim's name on the buddy list and a window pops up. Joe types his message in the window, and it is sent to Tim.
6. When Joe receives a message from Tim, the message appears under the messages already sent. The user name appears before the message so Joe can see who each message is from.
7. Tim tries to send an assignment to Joe using copy/paste on his keyboard. The assignment is 500 words long and contains tabulations, line breaks and spaces as well as standard Unicode characters. Joe receives the correct text with proper formatting.
8. Joe is logged in and his friend John is also logged in. But Joe cannot talk to John because he does not have him in his buddy list. Joe can talk to John by adding John to his buddy list.
9. Joe is not friends with John anymore and he does not want to talk to him. After logging in, he decides to remove John from his Buddy List by clicking on "remove buddy" next to Johns name
10. Joe is logged in to the system. He wants to have a conversation, but none of his buddies are online. Joe selects "Random" and a chat window opens up, connecting him to a random user.
11. Jim wants to use the chat program, but he does not want to create an account. Jim visits the home page, and clicks to log in as a guest user.
12. Jim, a guest user, is logged in and wants to have a conversation. While Jim has no buddies, as he is a guest, he can have conversations with random others. Jim clicks "Random" and a chat window pops up connecting him to a random user.
13. Joe is having a conversation with Jim, but his friend Tim is also online. Joe wants to talk to Tim too. Joe selects to talk to Tim and a new window opens alongside Jim's chat window allowing Tim to talk to both people.
14. Joe wonders what previous conversations he has had with Jim. Upon opening a new chat window to chat with Jim, the history of their communication is shown above.
15. Jim, Joe, and Jay want to have a three-way conversation. When all three are logged on, Jim opens the chat room option bar and selects to create a new room. Jim enters the information he is prompted for, and the room is created. Jim is the administrator of the new chat room.
16. Joe and Jay want to join Jim in a recently created chat room. Joe and Jay open the chat room bar and select to join a chat room, they enter the information of the chat room, as provided by Jim and are connected. The three are now connected to the same chat room.
17. Joe is misbehaving in the chat room. The administrator, Jim, thinks Joe should be quiet for a while. Jim selects Joe's name from the list of people connected to the chat room and selects to mute him.

18. Joe continues misbehaving in the chat room. The administrator, Jim, thinks maybe Joe should be removed from the room. Jim selects Joe's name from the list of people connected to the chat room and selects to kick him.
19. Joe logs back into the chat room and is still misbehaving. The administrator, Jim, has decided it is time to remove Joe from the conversation permanently. Jim selects Joe's name from the list of people connected to the chat room and selects to ban him.
20. Joe wants to communicate to Jim that he is feeling happy. Joe sends Jim an emoticon. When Jim receives the message containing the emoticon, it is translated into an image representation of the emoticon.
21. Joe wants to send Jim a picture that they are talking about. Joe selects the URL of the picture and sends it to Jim. When it appears in Jim's chat window, it is embedded as an image, so that Jim can view it directly, or click on the image to see the full size picture.
22. Joe wants to send Jim a Youtube movie. Joe selects the URL of the video and sends it to Jim. When itappears in Jim's chat window, it is embedded as a video, so that Jim can view it directly, or click on the URL to go to the Youtube site.
23. Joe wants to send Jim a file so that they can collaborate. In the chat window, Joe selects "Send File." Joe is prompted to select the file from his hard disk. When he is finished, he clicks "OK" and the file is uploaded to the server. The link is sent to Jim who can then download the shared file.
24. Tim is a very eager chatter. He sends Jim a number of messages per second, more than Jim can read. Tim is blocked from sending messages for ten seconds because of his behaviour.
25. Jim wants to let all of his buddies know something. He decides to set a status message. Jim clicks "Set Status" and enters the message he wants to be his status. When finished, he hits enter and his status is set.
26. Jim would like to update his password. He selects "User Information" and is presented with a dialogue to change his information. When finished, he clicks "OK."
27. Jim is being harassed by Tim. Jim would like to stop receiving chats and friend requests from Tim. Jim opens his user account information and selects "Blocked Users." He adds Tim to the list, and Tim is blocked from all communication with Jim.

Planning

Scrums

The scrum master for the first iteration will be Iain Macdonald. The master for the second and third iterations will be decided soon.

Expected Functionality

By the end of the final scrum, we would like to have implemented the basic functionality, including accounts, guests, logging in, and buddy lists. Additionally, user-to-user chat and random chat should be implemented. Users should be able to chat with multiple people at once. Flood protection should be implemented, as should blocked buddies and file sharing and image embedding.