Team ABC

ECSE 428 - Software Engineering Practice

Proposal

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

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.

Through the use of advanced technologies like the SQLAlchemy object-relational manager and Genshi HTML Python formatting, Turbogears allows development of complex web applications entirely in the Python programming language.

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.

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 log in so that he can use the software to chat with his friends. Tim enters his unique username and password and is logged in.

   Test Cases:
   - Normal Flow:
     - Tim visits the chat homepage. He is presented with a log in screen. Tim enters his unique username and password. He is redirected to his buddy list page.
   - Error Flow:
     - Tim enters an invalid username and password combination. The system redirects Tim to the homepage and displays a failed login attempt message.

2. Joe wants to be able to register with the chat application, so he can chat with his friends.

   Test Cases:
   - Normal Flow:
     - Joe visits the chat homepage. He is presented with a log in screen. He selects the register option. He enters a username and password and password verification. Joe is then redirected to the home page and a message confirms his registration.
   - Error Flow:
     - If Joe enters invalid or incomplete credentials, he is alerted of this and is redirected to the registration page to try again.
     - If Joe enters a username which is already taken by another user, he is alerted of this and is redirected to the registration page to try again.

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 his buddy list.

   Test Cases:

   - Normal Flow:
     - After logging in, Joe is redirected to the log in root page with his buddy list.
   - Error Flow:
     - If Joe tries to access the login root page using the URL without logging in, Joe is redirected to the login page.

4. Tim has logged on successfully and wants to message his friend Joe so they may start chatting.

   Test Cases:

   - Normal Flow:
     - Tim selects Joe's name from his buddy list for messaging. A message window with Joe appears and Tim can send Joe messages. These messages display in a similar message window for Tim.
   - Error Flow:

   1. If Joe is not online and Tim messaged Joe, a message is returned to Tim telling him that Joe is not online.

5. Joe receives a message from Tim.

   Test Cases:

   - Normal Flow:
     - Tim sends a message to Joe. In Joe's chat window, a list of recently received messages from Tim are displayed in chronological order. Each message is preceded by Tim's username.
   - Alternate Flow:
     - If Joe receives a message from another user, it is displayed in a separate chat panel in the same format.

6. Joe and John are logged in. Joe wants to talk to John, but they are not buddies. Joe sends John a friend request.

   Test Cases:

   - Normal Flow:
     - Joe selects to add a new buddy. He searches John's username and selects to add John. John receives the request and confirms. Joe and John are now buddies.
   - Error Flow:
     - If John does not confirm the request, they will not become buddies.

7. Joe no longer wants to chat with John. From his buddy list, Joe selects Johns name and selects to remove John as a buddy. Joe confirms the operation and John is removed as one of Joe's buddies.

   Test Cases:
   - Normal Flow:
     - Joe logs in and selects Johns name on his buddy list. He selects defriend and confirms the operation. John and Joe are no longer buddies and do not appear on each others buddy lists.
   - Error Flow:
     - If Joe does not confirm the operation, they remain friends.
     - Joe and John can no longer message each other when they are not friends.

8. Joe is logged in to the system. He wants to have a conversation, but none of his buddies are online. Joe wants to chat to a random user that is not on his friends list.

   Test Cases:
   - Normal Flow:
     - Joe logs in and selects the random button and a chat window opens with a random user (if any random users are available).

9. Jim wants to use the chat program, but he does not want to create an account.

   Test Cases:
   - Normal Flow:
     - Jim goes to the login page and selects the login as guest option. He is then redirected to the guest user home page.
   - Error Flow:
     - If Jim tries to login as a guest while he is logged in as another user, he is alerted that he must log out before logging back in and the login request is denied.

10. 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 users.

    Test Cases:
    - Normal Flow:
      - Jim logs in as a guest user. From the guest homepage, Jim selects the option to chat with a random person and a chat window pops up connecting him to a random user.

11. Joe is having a conversation with Jim, but his friend Tim is also online. Joe wants to talk to Tim as well.

    Test Cases:
    - Normal Flow:

    - From his buddy list, Joe selects Tims name and opens up a new chat panel with Tim. Joe can now chat with both Jim and Tim at the same time.
- Alternate Flow:
    - Joe creates a chat room and gives the name to Jim and Tim. Jim and Tim then join the chatroom and all three users can talk together.

12. Joe wants to review his conversation history with Jim.

    Test Cases:
    - Normal Flow:
        - Upon opening a new chat window to chat with Jim, the history of their communication is shown just like any other old chat.
    - Alternative Flow:
        - If Joe and Jim have not chatted before, no prior conversation is shown in the chat window.

13. Joe and Jay want to join Jim in a chat room that he has created, so all three users can participate in a conversation.

    Test Cases:
    - Normal Flow:
        - 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.
    - Alternate Flow:
        - Jim, from the chat room, selects to invite members. He then selects to invite Joe and Jay. Joe and Jay are then invited to the chatroom. They accept the invitation and are connected.
    - Error Flow:
        - If Jim enters an incorrect chatroom name, he is informed of this and is not connected to any chat.
        - If Jay declines the invitation to join the chatroom, he does not connect to the chat room.

14. Joe is misbehaving in the chatroom. The chatroom administrator, Jim, decides to mute Joe for a while.

    Test Cases:
    - Normal Flow:
        - Jim selects Joe's name from the list of people connected to the chat room and selects to mute him. All of Joe's messages are then ignored by the chatroom.
    - Error Flow:
        - If Jim tries to mute himself, he is notified that the admin cannot mute himself and nothing happens.

15. Joe continues misbehaving in the chatroom. The administrator, Jim, wants to remove Joe from the room.

    Test Cases:
    - Normal Flow:
        - Jim selects Joe's name from the list of people connected to the chat room and selects to kick him. Joe is kicked from the conversation.
    - Error Flow:
        - Jim attempts to kick himself out of the chat room and is notified that the admin cannot kick himself out.

16. 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.

    Test Cases:
- Normal Flow:
    - Jim selects Joe's name from the list of people connected to the chat room and selects to ban him.
- Error Flow:
    - Jim attempts to ban himself from the chat room and is notified that the administrator cannot be banned from the chat room.

17. Joe sends Jim an emoticon.

    Test Cases:
    - Normal Flow:
        - Joe types in text from a list of possible emoticons and the text is translated into an image representation of the emoticon, which is embedded in the chat.
    - Error Flow:
        - Joe types in text that is not part of the list of emoticons, his text is not translated into an image, but is displayed as plaintext.

18. Joe wants to send Jim a picture through the chat application by sending the URL of the picture 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.

    Test Cases:
    - Normal Flow:
        - Joe sends the URL of the picture that is uploaded to the web through the chat window and the picture displays in Jim's chat window.
    - Alternate Flow:
        - Joe sends the URL of a picture that is on the web the picture does does not display itself, Jim has to click the link to be redirected to the picture on the web.

19. Joe wants to send Jim a Youtube movie. Joe sends the URL of the video to Jim.

Test Cases:

- Normal Flow:
  - Jim receives the video and it is embedded in the chat. Jim can view it directly from his chat.
- Alternative Flow:
  - Jim clicks on the URL of the video. A new tab opens in his browser at the Youtube page.

20. Joe wants to send Jim a file from his computer so that they can collaborate on it.

    Test Cases:

    - Normal Flow:
      - 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.
    - Error Flow:
      - Joe attempts to send the file by dragging and dropping the file in the chat window. This functionality is not available so he receives and error message.

21. Tim sends Jim a large number of messages per second, more than Jim can read. Tim is blocked from sending messages for ten seconds because of his behaviour.

    Test Cases:

    - Normal Flow
      - Tim’s messages are blocked to avoid flooding, for 5 seconds after the incident.

22. Jim wants to let all of his buddies know something. He decides to set a status message.

    Test Cases:

    - Normal Flow:
      - Jim selects to set his status and enters the message he wants to be his status. When finished, he hits enter and his status is set.
    - Error Flow:
      - Jim attempts to set an invalid status. He is alerted of this and his status is not set.

23. Jim is being harassed by Tim. Jim would like to stop receiving chats and friend requests from Tim.

    Test Cases:

    - Normal Flow:
      - Jim opens his user account information and selects to view the list of blocked users. He adds Tim to the list, and Tim is blocked from all communication with Jim.
    - Alternative Flow:
      - Jim selects Tim's name on his buddy list and selects to block Tim.

Planning

Sprints

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

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. Some of the internal tasks required in completing these stories involve server configuration, database design, basic middleware code, webpage design, and documentation—including updating the proposal document and creating a spreadsheet form.

By the end of the first scrum, we would like to have implemented the basic functionality, as well as login and registration. We would like buddy lists to be working, as well as single-session user-to-user chat. In order to accomplish these tasks, we need to setup the server, design the database, install the necessary software, setup the entire database back-end, and document our efforts. We would also like to update the proposal document to a spreadsheet, for ease of use by the team members.

Sprint One Work Breakdown

Team Member	**Tasks**
Armen	- Documentation work - Login middleware implementation - Registration middleware implementation
Christian	- Login/registration page design - Login page integration - Dynamic data page design from the client side
Farzad	- Buddy list implementation - Buddy list design
Iain	- Database implementation - ORM configuration - Server setup - Scrum master - Login session management and authentication/authorization
Julian	- Database design - Backend database test code - Multiple person chat management
Pierre-Alexandre	- User-to-user chat frontend and backend

First Week Work Done

Team Member	**Tasks**
Armen	Improved documentation, including the proposal
Christian	Login page prototype, main page design, integration into the system
Farzad	Working on the buddy list backend and frontend, including dynamic polling
Iain	Database implementation, ORM configuration, Server setup, and Scrum master
Julian	Database design – high level and low level
Pierre-Alexandre	Worked on dynamic polling of server for messages

Armen has updated the proposal to include test cases and some stories modified. Additional information has been added regarding the work breakdown and development process. The next stage is to migrate this to a spreadsheet, so that everyone can use it easily in cross referencing their work with the proposal.

The login page has been prototyped and added into the system by Christian. It is clean with links to the necessary functions. The login page has been integrated as the index page of the server. Registration still needs to be prototyped, but that just involves changing the post form around a bit. The dynamic pages, such as buddy list and chat will be worked on over the next couple of weeks by Christian in an attempt to dynamically update the data they display.

Farzad has been working on the page designs for the buddy list and log in root page. He is still working on these pages, but they are nearing completion. He is focusing on design at this point and will move on to implementing the web-server middleware over the next week or two.

Iain sat down with Julian to design the database. They finalized the design and it shouldn't need to be altered for the rest of the project, aside from some minor modifications to accommodate Repoze.What authentication and authorization for next week. The database schema was implemented in MySQL and Iain coded the entire data model last week using the SQLAlchemy object-relational manager. The database interface is fully functional and has been tested, it is ready for use. This entire effort was documented by Iain and Julian in the docs folder. Iain also managed some of the first sprint activities, including server setup with Turbogears and MySQL, Assembla setup for SVN management of the code, and task delegation. Over the next week, authentication and authorization will be added into the project in order to manage user privileges. Additionally, some middleware will be written to manage user interactions with the system, from a lower level.

Pierre-Alexandre worked on dynamic polling of the server using Javascript and HTML integration. He also helped a bit with the design of some of the front-end pages and eased the workload for some of the other team members.

Next Week Work Plan

Team Member	**Tasks**
Armen	- Create spreadsheet version of user stories and tests - Finish and test login and registration middleware
Christian	- Front end prototype including dynamic viewing of pages using HTML and

	Javascript
Farzad	- Buddy list design and implementation
Iain	- Repoze.what authentication and authorization and session management - Scrum Master
Julian	- Database testing scripts - Back-end data authentication using the ORM
Pierre-Alexandre	- Further work on user-to-user chat feature - Help others with their tasks