Berserker Chat: Specification, Design and Implementation Report

Dan Tomosoiu, Tony Lau, Hector Grebbell
Berserker
DIM3
Student numbers
{1102486T, 1102266L, 1007414G}@student.gla.ac.uk

ABSTRACT

This report details the design and implementation of Berserker Chat – a web-based instant messenger and file sharing application which allows many users to collaboratively chat and share files. Berserker Chat is built using the Django Web Framework[1] which is written in the Python programming language.

1. AIM OF APPLICATION

The main aim of the application is to provide a service which allows users to communicate with each other in a real-time chat environment, not dissimilar to other chat applications which the user may have used before.

1.1 Goals and Objectives

The main goals and objectives are listed below:

1. Anonymous use of the application to chat with others in public chat rooms
2. Ability for a logged-in user to have a private chat with another logged-in user
3. Public chat rooms based on category/interests, e.g. public chat room for users interested in photography
4. Creation of public chat rooms based on interest which allow other people to find the created room, join and chat
5. Ability for file sharing between users – files are uploaded and can be downloaded by other users

1.2 Functionality

More specifically, the desired functionality of the web application is as follows:

1. Allow user to login and logout
2. Start a public chat
3. Start a private chat
4. Choose a chat by category
5. Allow user to filter chats specifically by searching for a category
6. A display of the recent chats that the user has been involved in
7. A file sharing interface in the chat room for users to share files with each other

DIM3 2013, Glasgow, UK
ACM X-XXXXX-XX-X/XX/XX.

1.3 Assumptions

- It is assumed that the application is to be used by adults and as such, no filtering of inappropriate language before they are displayed to other users in the chat rooms has been undertaken.
- The application will be displayed to the user in English.

1.4 Constraints

- The Django web framework must be used to build the application
- The application must be completed within the time allotted for the DIM3 course
- The application is to be designed and implemented by 3 people.

1.5 Reflections on scope and design goals

This application has considerable complexity, particularly in the methods which have to be used to achieve as close to instantaneous display of messages as they are sent by the user. The categorisation of chat rooms, adds to the complexity, as does the file sharing capability.

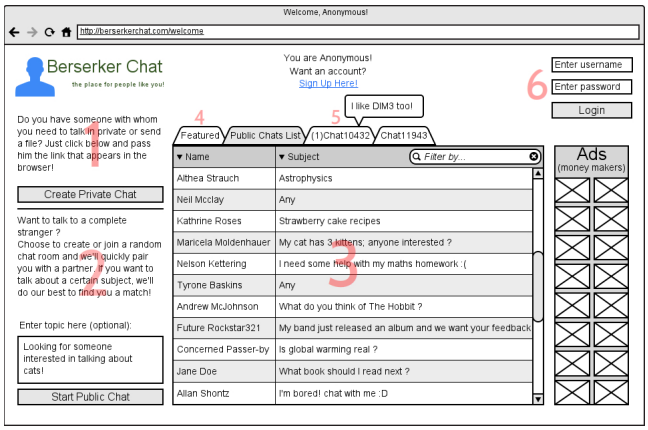
The team feels that the design goals are realistic for the most part. The functionality for users to chat to each other in different chat rooms is realistic and achievable. The only part which may not be implemented is the file sharing capability. The actual functionality implemented is discussed in Section 6.

This type of application is wholly suitable for distribution across the web. By distributing it over the web, anyone with access to an internet browser can use it to communicate with other people across the world. Furthermore, the categorisation of chat rooms allows people to talk to other like-minded people with similar interests.

2. CLIENT INTERFACE

2.1 Wireframe

The wireframe of the user interface of the main screen is shown below. This



2.1.1 *Description of numbered items*

1. This allows the user to create a private chat. When the 'create private chat' button is clicked, a link appears in the browser which the user can click to redirect them to the correct chat room. The private chat link can be passed to others: only those with the link can chat in that specific chat room.

2. This allows the user to start a public chat. When the 'start public chat' button is clicked the user is redirected to a random public chat room. If the user specifies a topic, then the user is redirected to a room for that topic, if it exists. If it does not exist, the room for the topic is created and the user can invite others to join, or wait for other people with the same interest to join.

3. This shows the list of public chats and the name of the person who started it. Clicking on a particular chat subject takes the user into that chatroom.

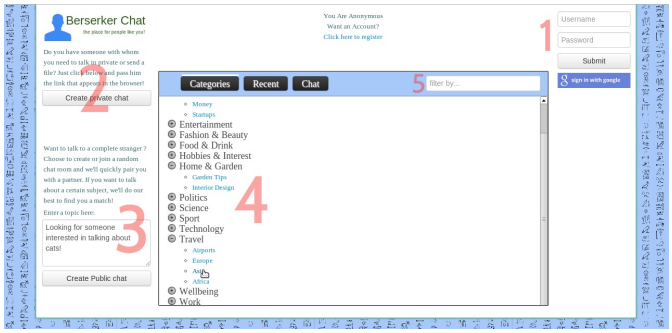
4. The featured tab shows the chats which have had the most users and are therefore very popular. This allows the user to see what the main topics of chat are.

5. A tab shows the chats the user is currently participating in with a pop-up of the most recently received message.

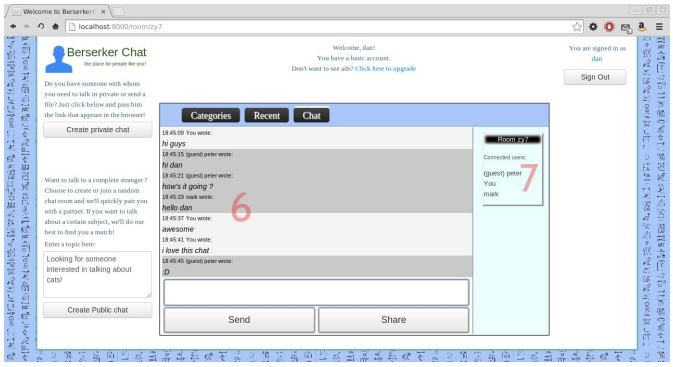
6. Allows the user to login and logout of the site

2.2 Implemented Client Interface

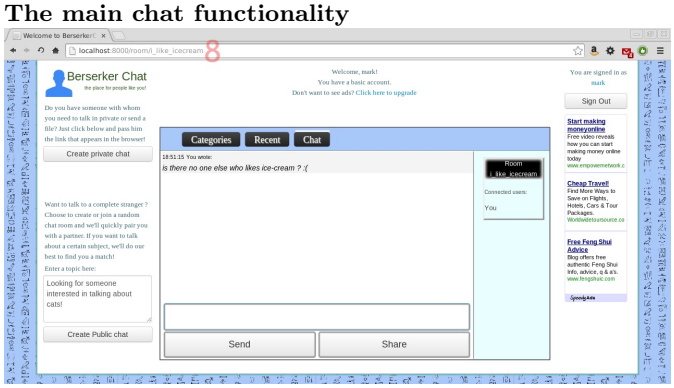
Some screenshots are shown below of the client interface implemented in Berserker Chat.



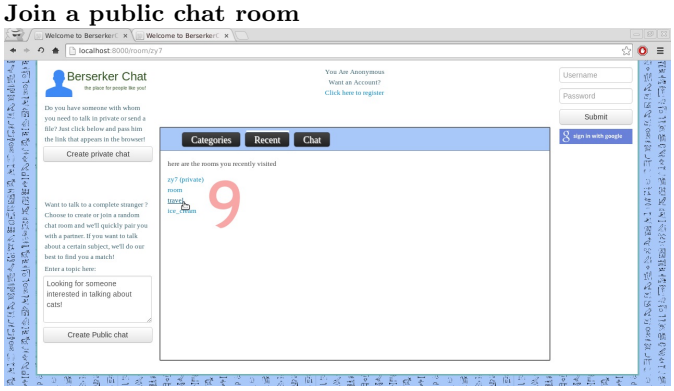
Main screen and category browsing



The main chat functionality



Join a public chat room



Recent view

2.2.1 *Description of numbered items*

1. Allows the user to login and logout of the site

2. Allows the creation of a private chat room. A randomly generated link is created and passed to the user

through the browser. The user can click the 'chat' tab to start chatting in the private room. The private chat link can be passed to others: only those with the link can chat in that specific chat room.

3. Allows the creation of a public chat room. If the user specifies a topic, then the user is entered into a room for that topic, if it exists. If it does not exist, the room for the topic is created and the user can invite others to join, or wait for other people with the same interest to join.

4. This a list of categories and subcategories. The user can click on one of the links to enter a chat room.

5. This allows the filtering of categories. The user enters a query and presses enter; the matching rooms will be shown.

6. The main chat view with messages shown from different users.

7. Lists the connected users in the chat room.

8. The URL localhost:8000/room/zy7 shows a private room.

9. A list of the most recent rooms which have been visited by the user are shown.

2.3 Remarks on the User Interface

It was remarked in the earlier stages of the project that the user interface shown in the wireframe above was too complicated and over ambitious. We set out to simplify the user interface and feel that it has been simplified to an extent which makes it usable. At the left hand side, it gives clear instructions of how to create rooms and the names of the tabs in the central part of the interface is sensible. The main functionality of chat can be, for example, easily accessed by clicking the chat tab.

HTML, CSS and JavaScript has been used to put together the user interface and to implement some functionality such as the filtering of categories. These technologies are widely used and are supported by all main browsers. The design is responsive to a degree, but is not as fully responsive as we would like. Using a responsive CSS toolkit such as Twitter's Bootstrap would solve this issue and it is something which is definitely to be implemented in the future.

To update the data on the client side, AJAX is used. The messages are dynamically updated on the browser as they appear in the chat window.

3. APPLICATION ARCHITECTURE

The application is designed to use a 3-tier architecture as illustrated in the diagram below. Using this type of architecture allows a separation of concerns between the display of the content on the client browser, the web application framework, and the backend.

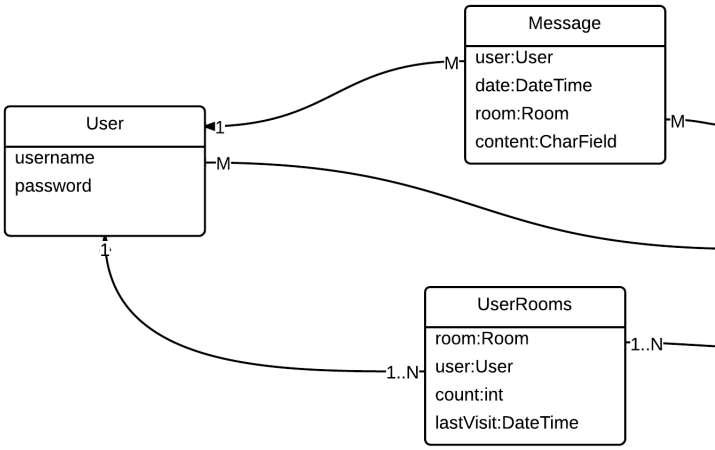
The 3-tier architecture consists of:

Client
The client (user) interacts with the Berserker Chat application through a web browser

Middleware
The middleware is where the Django web framework is positioned. It essentially glues together the client and the backend, providing views based on data contained within the database and also updating the database based on the interactions of the user at the browser interface level.

Database
The database stores information about the models which can be requested by the middleware by an Object Relational Model request. The database is implemented using SQLite3.

4. ER DIAGRAM



- The data model requires details for three main entities, Users, Messages and rooms. For a user a username and password is required, Room a name. Messages have a room, a user, a timestamp and their content. Since previous messages are not shown to the user we only need store these for a few seconds to allow for ajax calls to retrieve them, but currently they are kept indefinitely.

- A large relation is required to keep track of a users history with a specific room.This allows the âĂIJrecent roomsâĂİ tab to be calculated. A count of visits is also kept, currently for usage analysis, but this will hopefully be future developed to a âĂIJfavouritesâĂİ tab. Each message has a Room and a User associated with it. Each room has a list of subscribers. Again currently this is not used but in future would allow access restriction for private rooms to a specific group of users.

- The diagram below shows the data model specified above. Password and subscribers are shown in red since they are not currently used but will be required for future work. The relations are also specified. One user can have several messages, but you cannot have a message associated with several users. THis is the

same for rooms.This is the same for the UserRooms relationship. One user can have up to N where N is the number of rooms. One Room can have up to N where N is the number of users. There is a many to many relationship between users and rooms. Multiple users can be subscribed with multiple rooms and vice-versa.

4.1 Reflections on Application Architecture

Separation of concerns is extremely important to ensure a consistent structure of the application and to ensure maintainability. It also allows development to happen in parallel, e.g. someone could be working on the HTML and CSS of a page, and someone could be working on the database backend. The separation of concerns in this application has been achieved by using the Model, View, Template pattern in Django.

This allows the display of the application on the user's browser to be separated from the business logic of the application. The view is responsible for mapping a URL to the data which should be displayed. The template controls how the data identified by a particular view is displayed to the user. The model is essentially a description or definition of the data that the application stores in a database. This maps easily to a relational database model. The view interacts with the model to obtain information for display using the template.

Other web technologies/frameworks could have been used instead of Django with some gaining a lot of traction and support from the web development community. Some of these are: Symphony, Ruby on Rails and Cakewalk.

Advantages and Disavantages of using Web Application Frameworks (WAFs)

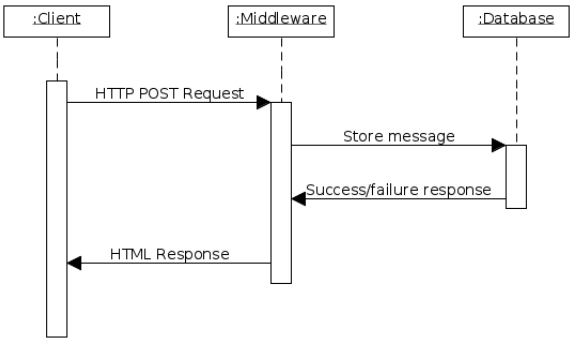
- Web application frameworks can decrease the time it takes to develop an application but also introduces the possibiity of a high learning curve to learn how to effectively and efficiently use the framework to enhance web development.

- Many WAFs are built upon some underlying design patterns, e.g. Django effectively uses the Model-View-Controller (MVC) pattern. This encourages the developer to adhere to software engineering best practices, which in turn allows high quality, maintanable applications to be built.

- Security issues and bugs found in the framework can affect all application that use that particular framework.

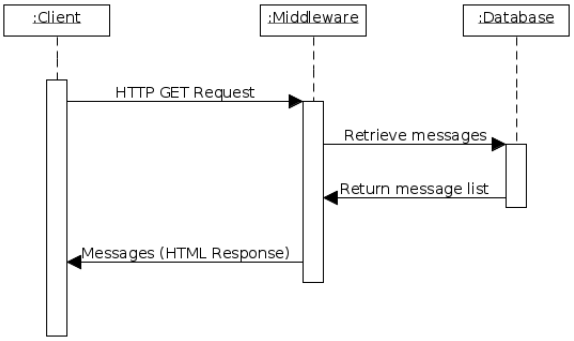
5. MESSAGE PARSING

5.1 Sending a chat message



On sending a chat message, a HTTP POST request is made to the Django web server. The message is then stored in the database and reponse requests are made.

5.2 Displaying chat messages



On getting chat message, a HTTP GET request is made to the Django web server which in turn fetches the messages to be displayed to the user. Messages are polled for frequently to achieve an instantaneous feel to the application.

6. IMPLEMENTATION NOTES

6.1 Views

- Index – This is the home page and displays the tabs required for interaction with the application as well as the login/logout forms.

- Upgrade/My Account/Register – These views provide basic functionality for the users to upgrade, view their account details and register using a form.

- Categories – This view is included in the index view and allows the user to view categories and filter categories of chat rooms based on personal preference.

- Recent – This shows the most recent chat rooms that the user has visited and allows them to revisit these rooms.

- Room – This is the main view for the chat functionality. It allows users to send messages to each other and dynamically updates the messages in the chat room as they are entered by the users.

6.2 URL Mapping Schema and External Services

We set out to create URLs that were friendly and intuitive. As such, for the chat rooms, we have created the URLs in the form /room/something, where something is the name of a chat room. All other pages which don't start /room link to a specific page, for example, /register links to the registration page.

We have used Amazon EC2 to deploy our web application in the cloud. While this external service is not used for the core functionality of the application, it gave us experience in deploying a web application which will no doubt be useful in the future.

6.3 Functionality Checklist

The following functionality is implemented:

- Anonymous/guest chatting
- Register/login/login using Google
- Featured rooms
- Featured rooms filtering
- Private chat room creation
- Public chat room creation
- Auto-room creation through link (create private chat/public chat)
- Recently used rooms
- 1-to-1 chat/multi-user chat
- Advertisements

6.4 Known Issues

- The file sharing functionality is not fully implemented. A mock system of uploading and sending a file inside a chat when you click the 'share' button is currently implemented.
- Creating/switching to a room refreshes the page and creates the chat but does not switch to the chat tab; the user has to manually click the chat tab to start chatting in that room.
- The ability for a user to enter multiple chats is currently not available; they can only chat in 1 room at a time.
- The error message system e.g. 404 Not Found displayed on non-existent page is not currently active.

6.5 Technologies Used

- XHTML/CSS
- Javascript with jQuery
- AJAX
- Django/Python
- SQLite3

7. REFLECTIVE SUMMARY

The use of the Django framework at first slowed down the development process since there was quite a steep learning curve. As we got more comfortable with Django though, we felt the use of the Model-Template-View (MTV) that is central to Django, decreased the development time by quite a large amount. It meant that parts of the application could be developed in parallel which would perhaps not otherwise have been possible without the user of a web development framework.

2 of the 3 members of the team had no previous experience with Python which added some challenges, naturally, to the development of the application.

Furthermore, a range of technologies had to be used, some of which we were not too familiar with. This brought its own challenges and as a result of this project, we feel we have obtained a greater and more broader knowledge of these technologies.

We would say that our major achievement is going from the specification and design of an application to a product, which while not complete, includes most of the core functionality that we set out to implement. Another achievement is also the learning of the Django framework (and other technologies) in a short space of time which allowed us to implement the application. If we were to implement another web application, the process would be easier as we obtained some valuable experience in this project.

8. SUMMARY AND FUTURE WORK

To summarise, Berserker Chat is a web-based instant messenger which allows many users to interact with each other to chat. It includes functionality to find other users by interests, by the categories section and also to create both private and public rooms which have not existed before. In terms of future development, the file sharing could be fully implemented to allow users to upload files and for other users to download them. Furthermore, the user interface could be further enhanced to be fully responsive using a CSS toolkit.

9. CONTRIBUTIONS

Equal contributions. **Dan Tomosoiu** – design, templates, views **Tony Lau** – specification, templates, report **Hector Grebbell** – views, models

10. ACKNOWLEDGEMENTS

Thank you to Dr. Leif Azzopardi for his insightful suggestions and motivating comments. Thank you also to the demonstrators for their suggestions on the application.

11. REFERENCES

[1] *https://www.djangoproject.com/*