One messaging computing artifact report

Abstract

The web as a service is constantly seeking to modernise and adapt over its previous iterations, whether this is for optimisation or user experience. One of the key areas of this refactoring is the way that clients and administrators communicate in web pages. The previous mainstream method was static contact forms. Users would fill out simple form input fields, often directing them an email address to get answers to their query. This was adequate for providing a basic method of communicate on web pages, however flaws such as long response times and the growing need for direct communication and an engaging service became apparent. Therefore, the modern iteration of this service is known as ‘live chat’ this often being presented as a modal like window throughout sites that provides a real-time chat interface between the client and admin.

Introduction

The artifact I have designed and constructed is inspired by ‘live chat’ seeking to replicate its core features. These features being: one to one communicate through a messaging IO (input and output,) user login, registering and subsequent storage (In a site application of this service the site itself would handle these features.)

Recapping my USP, (unique selling point) identified in my technical poster, large sites misuse ‘live chat’ by attempting to cut down on the number of staff needed to maintain the service. This is often achieved through a ‘chatbot’ a simplistic form of AI (artificial intelligence) that matches the user’s query to a pre-defined question and answer style form. Mimicking a less effective customer support member. This reverts the modern principles provided by the service resulting in the same flaws as the contact form, as queries are less likely to be solved. Similarly, to get to the real admin the user will be redirected to another page, email or phone number exactly as there were with contact forms.

As a result, I wanted to direct my application in a way that it was easily packageable and applicable to web page modals by small sites specifically. Because they are less likely to revert to chatbots due to an easily managed amount of site traffic. Benefit their sites user experience and in turn conversion rates, site traffic and overall reputation. Whilst taking advantage of the need of multiple system for messaging and a potentially external database list this project as a distributed system.

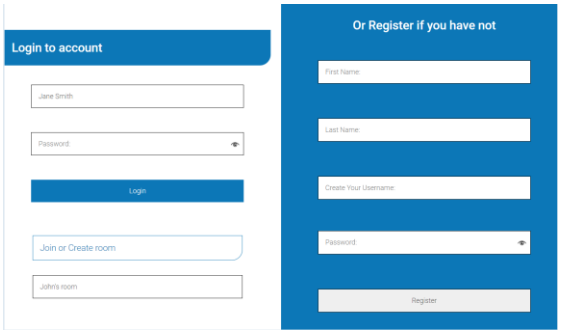
Literature Review

During my research before the development of this project I considered ‘does `livechat` benefit communication compared to the previous iterations?’ Research conducted in ‘Computers in Human Behaviour’ [1] found that live chat as a well delivered and maintained service provides a high user experience. It can be inferred that this experience level is greater than previous communication methods due to ‘live chats’ direct and swift interactions when compared a static contact form for

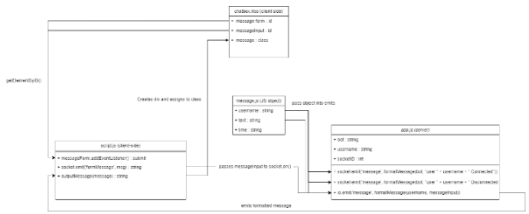
example. Chatbots, as mentioned, pose a negative impact on the service not only for their lack of direct communication and the undermining of what makes ‘live chat’ a unique service, but chatbots have been found to be immature as a technology [2] decreasing its trust as software. Although ‘live chat’ may share the same immaturity it’s benefits of user experience and direct communicate out way these concerns, with communication moderated by a real user likely to be safer for the end user. Furthermore, chatbot interactions are less likely to contain detailed messages and unique conversations from clients, when compared to a real-life user [3]. Also implying limited details of personal potentially contribution to a decreased rate of queries resolution. However even this small amount has been found to raise concerns of the privacy of information sent [2]. ‘Live chat’ also has a vast number of implementations, a trait it shares with previous iterations, for not only consumers and web customers but educational applications. Research even suggests [4] that providing this service in the recent form of online learning it can be provide additional interaction and engagement. Not dissimilar from the same benefits web users receive from the service.

Features

The minimum viable product for this application was a login and registering web page that would store credentials to a database and a subsequent page after a successful login in to send and receive messages. With messaging containing information such as the user it was sent from, what they sent and the time it was sent. With additional stretch features being the implementation of rooms and persistent storage.

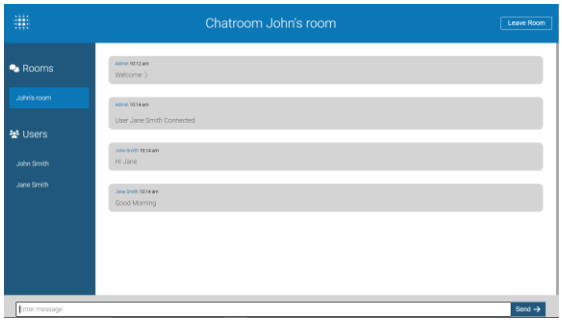


The stack used for the project was Sqlite, Express.js, Handlebars, CSS and Socket.io along with the moment package. These were all the same as listed in the technical poster aside from the addition of the moment package. This was because moment is a useful package with getting current dates and times, this could be easily added into my messaging object so any messages would have time automatically managed. I found Handlebars limiting when compared to the React library for example and would have preferred to use that if the artifact was to be constructure again. This is due to the number of features available client side in react such as states for functionally, class or function properties for dynamic content and mapping for creating components as often as conditions require. Moment could have also been excluded due to build in time functions.



Above is a class diagram showcasing the interactions between the systems when users create and send messages. The interaction is handled by three separate systems: the server (app.js), client-side javascript (script.js) and the client-side render (chatbox.hbs.) The server formats messages into ‘socket emits’ or ‘broadcasts’ either when the server starts

or when data has been received from the client-side. The difference being that an emit can be viewed by all users and a broadcast cannot be viewed by the current user. The message.js object is used for formatting messages into username, message, and time. It is presented as a separate object specifically so that the moment package will automatically handle date and time regardless of where this object has been implemented. The 'Welcome, connected and disconnected' messages are all handled automatically by the server when the conditions are met hence why they use the 'bot' variable to represent a message from the server. Alternatively, the 'message and username' emits are manipulate by the client-side JavaScript before being displayed. This is to create dynamic elements every time the conditions are met for example: every message creates its own 'div' element and 'innerHtml' to display the content. These created elements are then appended as children to parent elements in the Handlebars file.



Timeline

In the technical poster I had highlighted the following four milestones: Create and styling the front-end interfaces using handlebars and CSS, release a user study highlighting these interfaces, create the login and registering system with SQLite and finally add the messaging component with socket.io.

During the development there were deviations from this initial development plan. The first milestone was followed almost exactly, with most interfaces being created within the first few weeks, aside from additional fields such as rooms. Although code refracting of the styling specifically came after the initial creation. The user study was the first area of some deviation. Releasing the user study to just highlight the front-end interfaces seemed irrational compared to releasing one major user study to highlight all areas of the project towards the end of the development cycle. This method would provide a greater insight into not only the projects aesthetics but the functionality, allowing for greater enhancements across the project. Therefore, I created a quantitative user study in the final week to record any improvements to make to the project. Feedback I receive were mainly centred around stretch features such as persistent data, rooms and creating the application in more of a 'modal' style.

The remaining two milestones were swapped in their order or development. I foresaw that the database aspect was the area I had the least knowledge in before development and to now slow progress, to the point where core features might be missing by the end of development, the messaging functionality was added first. Although this established many new concepts such as the use of a function to format objects and socket.io's broadcast, on and emit features. With the main challenges coming after a minimum viable product was established. Such as creating unique and joinable rooms or presenting all currently logged in users in the side menu. Finally, my assumptions proved to be correct. User authentication in the login and registering system proved to be where most of the development time was spent. With login functionality specifically being

the area with the most issues. Verifying the username credentials matched already registered credentials was the easier of the two input fields to achieve, in the section, as passwords created a bug. By hashing any registered password when stored in the database for security and data protection, it results in the only acceptable entry being the then hashed and salted password. Therefore, my previous method of just checking the request against the corresponding field in the database wouldn't work. The solution was to check the requested password's hashed version along with the stored salt to see if the encrypted version of the request resulted in the same hashed salt as stored in the database.

Future Works

Future direction of the project given additional time is to move closer towards the packageable modal aspect, as mentioned. Comprising of correctly separating all code relating to the socket and messaging system, with additional styling and logic to create the 'pop-up' effect so this can be easily applied to existing web pages with their own login and registering systems. Having established the minimum viable product, set out by the original timeline, stretch features would include Allowing users to create and join private rooms through password keys. Private messaging by clicking a username to create a one-to-one direct message and finally persistent database storage for usernames, room name and message history so that frequent users can see their previous communications after the application has been closed.

Conclusion

In summary 'live chat' is the webs most modern approach to standard communication between site owners and site users. With its novelty it creates the potential for its USP of direct and quick communication to become misguided. Hence the rise of 'chatbots' on large sites. To mimic engaging and personal conversations as if they were with a real person. This is to no avail as research suggest users are shown to get more gratification and provide greater detail when a real user is the recipient. In turn increasing the need for an easily packageable and distributable system to uplift their communication methods. A gap that this artifact aims to fill. With its real-time messaging and potential for persistent storage and direct messaging.

Bibliography

[1] G. McLean and K. Osei-Frimpong, "Examining satisfaction with the experience during a live chat service encounter-implications for website providers", *Science Direct*, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0747563217304727#!. [Accessed: 28- Mar- 2022].

[2]A. Rese, L. Ganester and D. Baier, "Journal of Retailing and Consumer Services", *Science Direct*, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0969698920308286. [Accessed: 28- Mar- 2022].

[3]J. Hill, R. Ford and I. Farreras, "Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations", *Science Direct*, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0747563215001247. [Accessed: 28- Mar- 2022].

[4]J. Broadbent and J. Lodge, "Use of live chat in higher education to support self-regulated help seeking behaviours: a comparison of online and blended learner perspectives", *Springer Open*, 2021. [Online]. Available: https://educationaltechnologyjournal.springeropen.com/articles/10.1186/s41239-021-00253-2. [Accessed: 29- Mar- 2022].