Online Realtime Chat Application (ORCA)

'Meet new people and make friends'

Contents

ANALYSIS

Description of my Stakeholder

A Local children's hospital has over 350 beds for sickly children and a further 50 for terminally ill and long-term care. The hospital also acts as a mental health centre for the local community, helping young people deal with mental health issues and giving them a safe place of refuge.

My Stakeholder's Problem

The local children's hospital wants to solve the problem of loneliness and isolation many of the children and young people face during their stay at the hospital. Either for mental health problems or physical ailments, they have found that they recover slower when isolated. Especially those who have extended stays at the hospital, suffer the most and find it hardest to readjust back into their old life. Furthermore, affecting them on their return to school and creating a sense of alienation for many of the children and young people. Moreover, the children are more likely to get sick when their mental health is at a low, their stay at the hospital normally only worsens their condition. Improving their mental health may improve recovery time for the children and young people at the hospital helping them return to normal life.

My Stakeholder's Solution

The hospital wants a website accessible to all young people from anywhere on their network. They desire the website to be a chat site where other young people in the hospital can communicate with one another. They want the website to be safe for young people to use and prevent unauthorised people who are not patients from using the website. Finally, they want there to be a feature where the hospital can create topic channels where young people with the same interests can chat, like a gaming or cooking topic channel.

My Research Plan

The research plan will be a simple questionnaire sent out to a sample of 100 random patients in the ward, to understand the demand and wanted features of the website. This research will give me a clear plan on how I can properly fulfil my task and find any additional requirements for the website.

Questionnaire

The hospital's plan for a new website questionnaire

1. Which of these options is the biggest problem you face during your stay?

(Input answer in text box)

2. Does the hospital in its current state fulfil your emotional needs?

A) Yes
B) No

3. Which of these options would help improve your situation the most?

A) Stuffed animals
B) An online chat rooms
C) More staff

4. Which of these features would you want to see most on the website?

A) Topic channels
B) Image sharing
C) Voice chat

5. Which of these options is your biggest concern when using an online platform?

A) Bullying

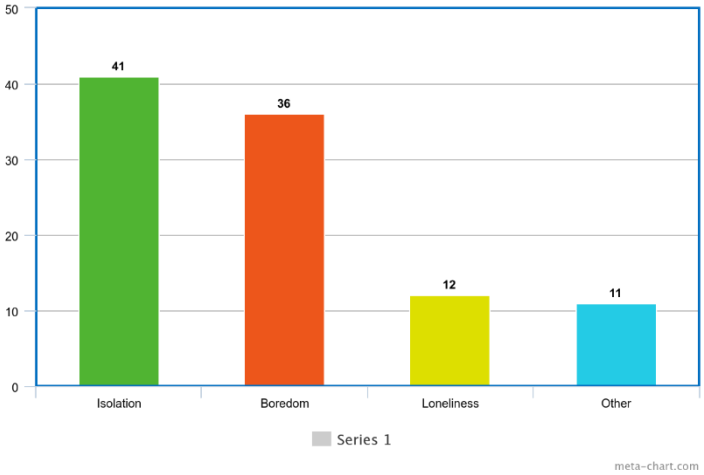
B) Strangers
C) Not meeting people with the same interest

6) Do you believe that this website would be helpful for you?
A) Yes
B) No

Questionnaire Results

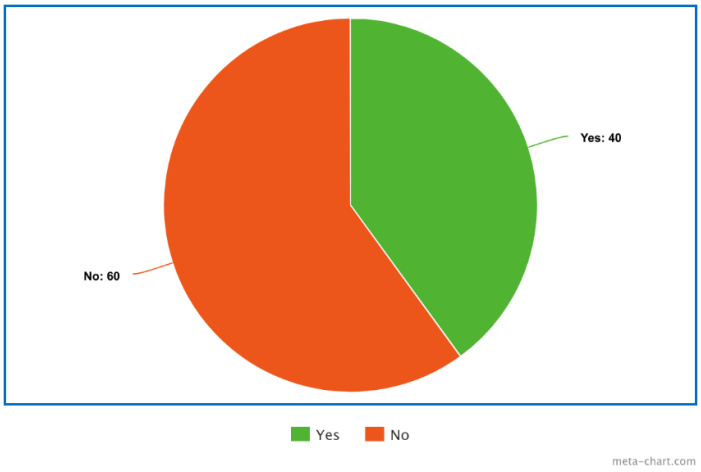
Question 1

The 3 most occurring problems faced by the respondents were isolation, boredom loneliness, and boredom in that order of most frequent. This proves that the patients face the problems as stated in my stakeholder's solution. This also tells me that the website must be engaging and interactive for users while allowing people to chat and conversate over the website.



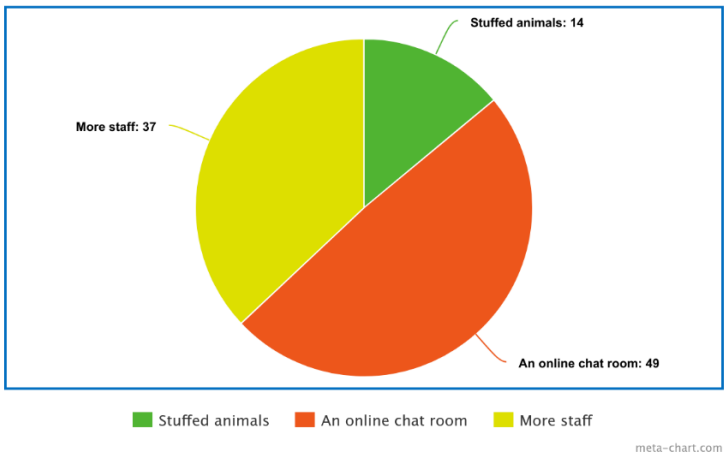
Question 2

This shows that overall, the hospital lacks in fulfilling the emotional needs of the children, proving the hospital's problem to be factual. Furthermore, this helps me understand the demand for a website that could fulfil the emotional needs of the patients.



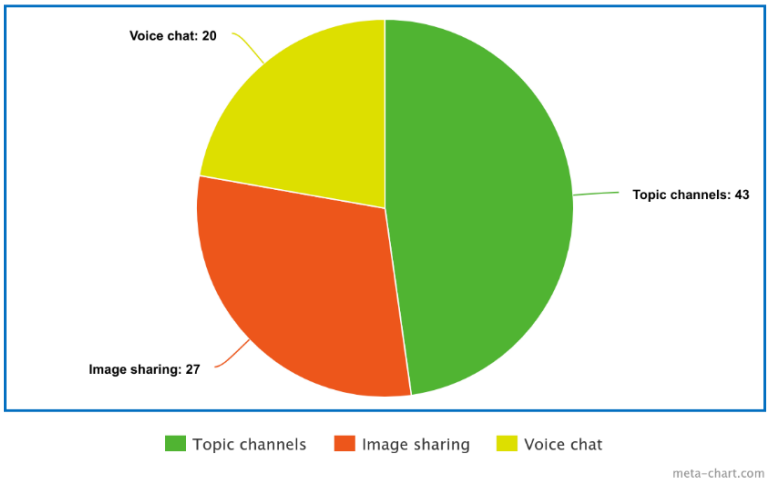
Question 3

This proves the desire and demand for an online chat website, as most respondents believe that, an online chat website could improve their situation. This helps justify that the hospital's solution to the problem is valid.



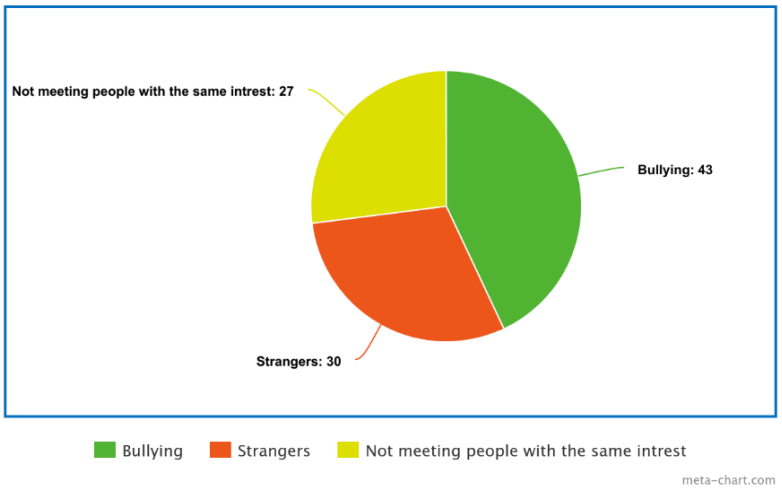
Question 4

This presents the demand for chat channels on the website, validating the hospital's solution. Also, one of my primary features to implement is chat channels. Furthermore, image sharing and voice chat are also popular choices so they will be included if I complete the key features first – like the real-time chat and topic channels.



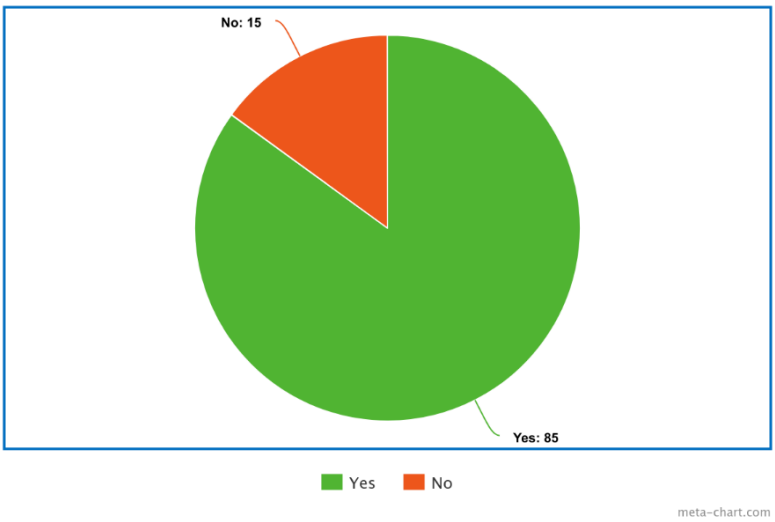
Question 5

This validates the hospital's concern for having the site be moderated and prevents people outside the hospital from accessing the website. It also, shows that the respondents want to meet others who share their interests. This helps to reinforce the solution presented by the hospital.



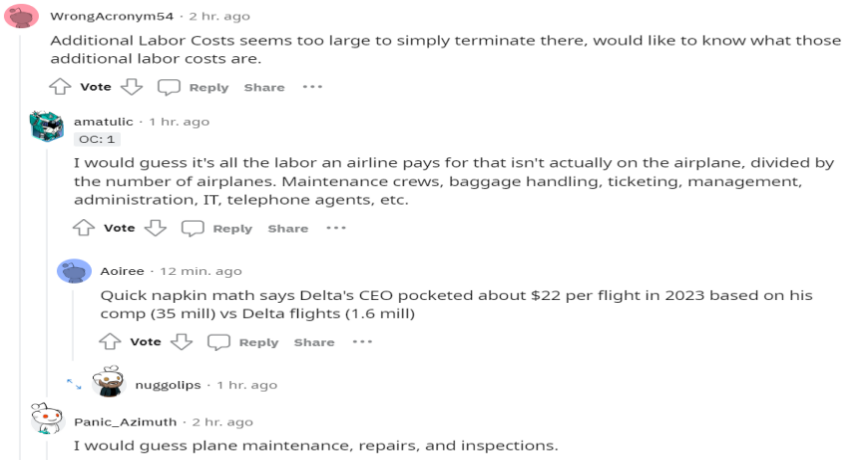
Question 6

This confirms that this website will be useful for the patients to use and strengthens the solution presented by the hospital. Allowing me to be able to understand the level of demand also.



Existing Solutions (Reddit)

Reddit allows users to be able to create 'threads' which are like topic channels where relevant discussion takes place. Users can make comments on the thread can create meaningful discussion on the thread's topic, where moderation teams can moderate the thread. Creating a peaceful and harmonious discussion, mitigating off-topic discussion and anti-social behaviour like bullying. I plan to add this feature where admin users can create threads where discussion on a topic a take place but also allow admins to be able to moderate discussion.

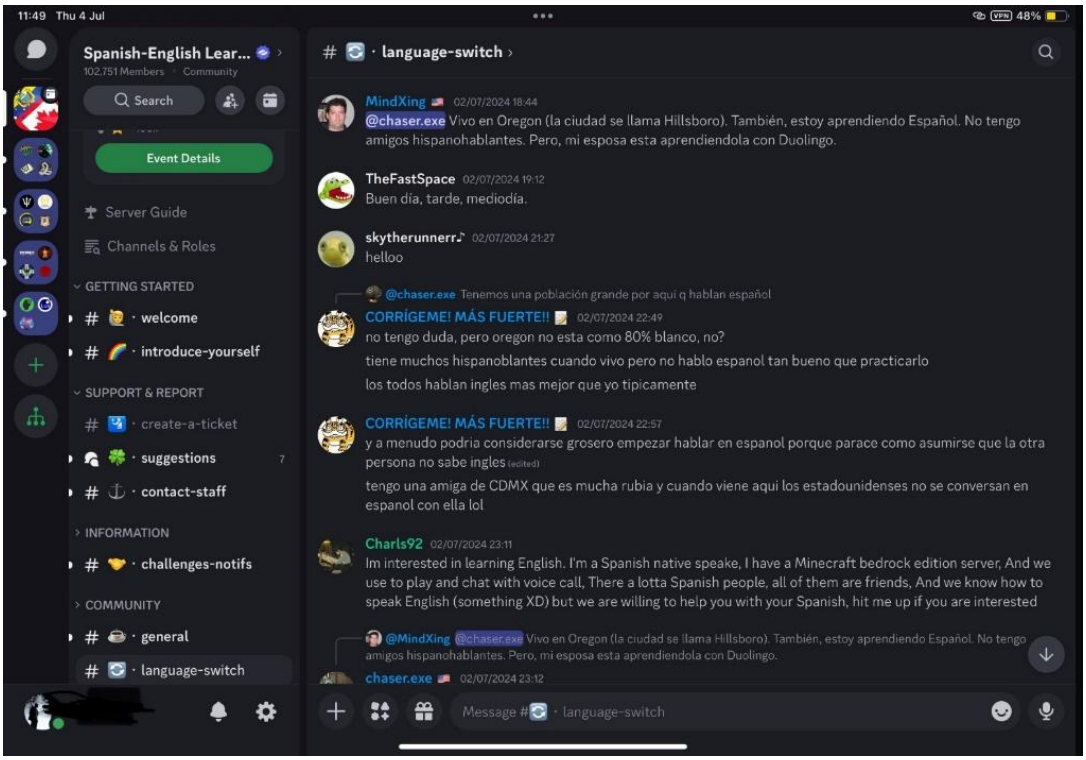


However, the effectiveness of the moderation is limited since the most severe moderators are volunteers, which could present an issue of bias, over-moderation, and unprofessionalism. Furthermore, these threads are not 'live chat,' which means it may take some time before another user can view your comment.

In addition, this website is open to the public as anyone can make an account, making it dangerous for children to use the website safely. Especially, since there is NSFW (Not Safe for Work) content on the website, which is easily accessible by all users.

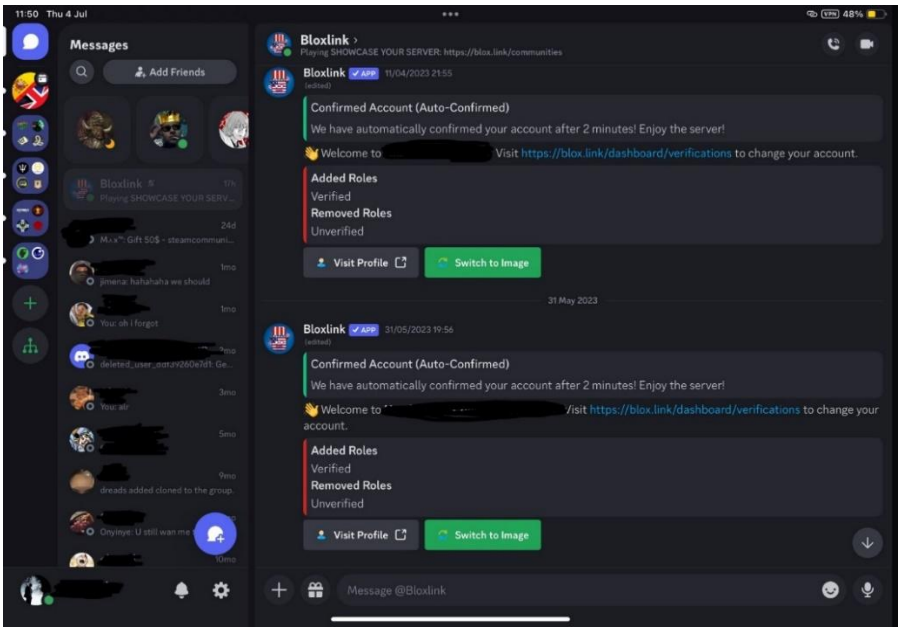
I plan to make the topic channels live chat so that people can have a discussion that flows without interruption. There will be no NSFW content on the website and there will be a chat filter that blocks certain words and phrases like profanities. Also, to create an account to access the website, you will first need an 'institute key' which will be a set of four numbers and letters that can be set by the hospital to prevent people outside the hospital from accessing the website.

Existing Solutions (Discord)



Discord allows users to be able to upload images, chat with one another in real-time and join voice call channels – where users can communicate using their voices. Furthermore, there are servers which can have a general topic with many topic channels (as shown in the image above) each with different themes, and topics of discussion. Discord allows for diverse communities to gather and meet over the website, as you can find different servers that cater to your interests, like Spanish. There are different levels of access on the website, called 'roles' which grant access and permission to different topic channels. Through users with moderation permissions, the server is moderated.

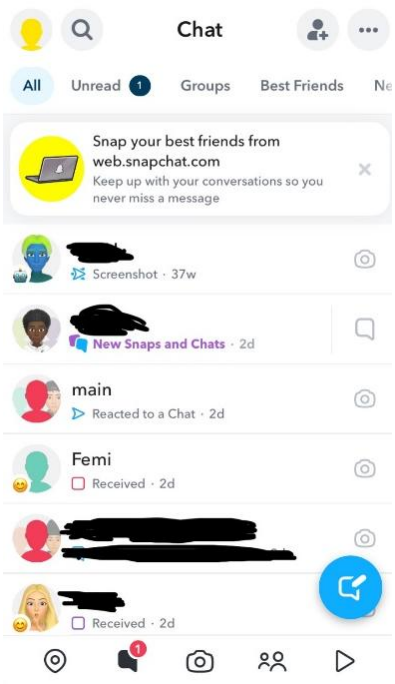
I plan to implement 3 levels of access on the website – Younger Children, Older Children and Admins. Where younger children may have limited access to certain topic channels. While Older children will have access to all channels. Finally, Admins will be able to create topic channels and moderate user and their messages. However, additional features like voice chat, and being able to upload images may not be feasible given the allowed time.



However, you must be invited to a server, which some users may find difficult as you would need to know someone already on the server. Moreover, there are many NSFW servers as any user can create a server, this stands as a danger to children as they could gain access to inappropriate content. In addition, users can send direct messages to one another. These DM are completely unmoderated which can embolden malicious users to send harmful messages and images to another user or possibly send dangerous links that could steal a user's details.

To prevent these shortcomings, there will be no direct message feature on the website, nor will non-admin users be able to create topic channels.

Existing Solutions (Snapchat)



Snapchat allows users to be able to send messages (in real-time) and 'Snaps' (photos) to other users. The snaps only can be viewed twice unless a user saves the image. The allows for private and secure photo sharing and messaging. Furtfhermore, you can create a group chat with your friends where you can all communicate with one another in real-time. Moreover, you can add people on Snapchat very easily by entering their usernames or scanning a QR code. I plan on people being able to send friend requests by entering the person's username and unique username identifier.

However, Snapchat can be dangerous for other children to use since it is extremely easy to add people as friends and send inappropriate messages or images. There is no safety measure to protect children from bad actors. I plan to create a safe environment for children and young people by restricting who can join the website.

Proposed Solution (Requirement Summary)

The hospital's requirements and the patient's questionnaire helped me to understand the essential features that they both would like to see in this project. The questionnaire also helped to support the hospital's proposed solution and gave me a

clear understanding of any additional features I should try including. Furthermore, the existing proposed solutions have helped me to understand what they accomplish well and what they do not. I hope to be able to solve these shortcomings.

Here I will list and explain the key features that I will create for this project, all contributed from the research and requirement above.

1) Creating a safe and open environment. The website must allow young people to be able to find others and communities with similar interests to them and discuss these topics openly.

2) An online website accessible via the internet. The website must be online so that young people all over the hospital can access the website from devices with varying specifications.

3) Creations of topic channels. Admins must be able to create topic channels so that young people can chat on a specific topic.

4) Website security. People outside the hospital must not be able to access or create an account on the website, and user data must be stored encrypted.

5) Real-time chat. There must be real-time chat or 'live chat' capability for every topic channel.

6) Multi-layer user access. Admins must be able to moderate messages and users to ensure the safety of the website. While users should only be able to chat and send messages.

7) Clear and usable graphics for the website. The layout of the website should be accessible for all ages.

8) Chat filter and moderation. There will be a chat filter to censor inappropriate words.

Solution Success Criteria

Moderation and user safety – Ensuring security and safety on the website is especially important for my stakeholders since they are responsible for the care of the children and young people who will use the website. To achieve user safety, I will use a chat filter which checks every sent message on the website and replaces the inappropriate word with '#.' I will use an already pre-existing external database that stores commonly used bypasses and regular inappropriate words. This should not be hard to implement but I will be creating an SQL database that receives queries from the website which, could prove to be a challenge. However, under the constraints of time, it is a realistic goal since there are many resources available to create such a feature as this is a common feature most websites use.

Real-time chat – Creating a website with real-time functionality will be the most important feature of the project since it is the main solution of my stakeholders. It will also be the most difficult feature to create, however creating a real-time chat website is a quite common feature of many websites and has an abundance of resources to help direct me. Furthermore, since it is the first feature I created, giving me enough time to ensure that this feature works completely.

Website functionality and user interface – This is not as complex as real-time chat capabilities, but it will pose a challenge as it can be very time-consuming to create a user-friendly website with topic channel capabilities. Furthermore, I must create a user-friendly website since young people of all ages will be using the website. However, I plan to use pre-made assets and designs to help create a good user interface to minimise time spent on graphical improvements. In addition, the functionality side of the website should be less time-consuming compared to the interface since there are many resources on the many features I wish to add, since they are common features other many other websites.

Networking and backend – These are a complex feature of the project since it will be a new problem for me to solve. Fortunately, many applications like node.js can help me to manage most traffic to the website and network for the website. Node.js especially, allows me to be able to create multi-level user access and uses JavaScript as its programming language which makes it easier for me to transfer the skills I already have.

Computational Solution

Abstraction – This computational method is when unnecessary problems are removed. It will help speed up development time since removing unnecessary features helps me to program a robust framework for the project. Ultimately, saving me time and allowing me to implement more features, which will improve the solution of the stakeholder by the website having more desired features like voice chat and image sharing. However, abstraction could result in many bugs as many features and systems in the project need to be fully completed as they communicate with one another. Overall, it should help me especially when designing the interface of the project as I can just use temporary placeholders and focus more on the functionality.

Decomposition – This computational method helps to break down large problems into smaller more manageable problems. I plan to use this method to simplify the project so that I can better spend my time on each problem since the project is complex. This will overall reduce the complexity of the project which will allow me to be able to make any additional changes to the project easier if the stakeholder requests. Unintuitively, this can also increase complexity as it adds additional problems to the project that all interact with one another. Increasing the chance of errors. Nevertheless, the main isolated function of the website should be a helpful addition as the problem of the website functionality can be broken down into many small, isolated problems.

Algorithmic thinking – This method is a way of getting to a solution through a clear definition of the steps needed to solve a problem. It is most useful when creating the real-time chat function of the website and the networking since it is crucial for them to happen in a specific order. Otherwise, users could be desynced to the network and would not be able to receive or send messages on the website. This would prevent users from using the website, making the stakeholder's solution obsolete as users could become disconnected. Moreover, this method could increase development time since some functions are isolated programs or have pre-existing solutions. In conclusion, this method would be useful for the structure of the backend, and the real-time chat functionality since they must be complete and run step by step.

Hardware & Software Requirements

A desktop user may only need:

- Mouse, monitor and keyboard. For interacting with the website
- 4GB of RAM. This is for loading in storing the website
- Windows 10 or greater. The website will run on a browser, so a good operating system is needed
- 1.5 GHz CPU. There are minimum CPU requirements to run Windows 10

Moreover, the stakeholder will need to have a physical server to host the website or use a cloud service to host the website.

Likewise, I will need the same hardware specifications to develop and program the web application, but I will also need Visual Studio Code as my Integrated development environment (IDE).

Solution Limitations

The primary limitation of the project is not just the time/deadline but the broad scope of the project. The project incorporates many advanced and complex programming techniques and concepts. I believe that creating networking for the website will be exceedingly difficult as well as creating a website, that multiple users can use at once. Since these aspects of the project are the most difficult, they will also force me to spend the most time on them, depriving time for me to complete other functions of the website. However, I believe that I will be able to complete this aspect of the project with time to spare for other functionality because there are many online resources on this exact topic. This concept of a real-time chat website is not unique to this solution and a huge amount of documentation to help me solve problems that arise.

On the other hand, this may affect the stakeholder as this project may not reach the solution exception if I fail to complete all the features the stakeholder expects. Furthermore, it could be difficult for the stakeholder to maintain the website in the long term since this project requires a server to host the website. This will increase the cost for the stakeholders since they

need to pay for a cloud service or purchase and maintain physical servers. This extra cost could make the project seem undesirable, but I plan to make the website with a robust framework, with thorough documentation.

On the contrary, I believe that this project can be created in the allotted time and be robust, a limitation would be that there will be no mobile support for the website since it is outside the scope of the project and will overcomplicate the project. This could be considered a downside for the stakeholder since the website will not be entirely accessible, however, the website still will be easily accessible due to it having low hardware specifications for desktops.

Finally, the website has many features that I would like to add in addition to the primary features like the topic channels and real-time chat capability. This can be considered a limitation as many features could be added to the website, which in turn will slow down development time. This is not a problem for this project since I plan to implement the key features first while implementing the secondary features like image sharing after. In the short term, the stakeholders could find the website difficult to modify since there will be many features, but in the long term, they will benefit from the many features to enhance the user experience on the website.

DESIGN

Programming Approach

I plan to use object-orientated programming as my programming approach, however here I will compare the disadvantages and advantages of using a procedural programming approach compared to an object-oriented programming approach.

- Procedural programming is an approach where functions (subroutines) are used to break down a problem into individual but modular blocks of code. Now an advantage of procedural programming is its modularity, which allows for individual blocks of code to be tested. This helps and improves bug finding and makes it easier for the code to be maintained and repaired. Furthermore, functions can be easier and simpler to understand since they are self-contained and are labelled so that other developers can understand the function of the subroutine immediately. However procedural programming can be memory intensive when recursion is used. This will slow down the program and present a performance issue for low-specification devices. Moreover, procedural programming can also create redundant code since functions cannot access the local variable within another function. This can present an issue when a function needs to communicate with one another in real time, creating unnecessary dependencies in the program. Another limitation of functions is that you cannot run these functions to modify each other in real time - creating ridge and heavily dependent code.

- On the other hand, object-oriented programming breaks down problems into objects. These objects are instances of classes and have their methods and attributes. Therefore, each object is its subroutine and while also having its' own public and private variables that are separate to each instance of the class. This leads to one of the advantages of object-orientated programming – encapsulation. Encapsulation is when private attributes are changed by public methods. This prevents these attributes from being changed in an unforeseen way. Preventing an invalid value from being assigned to an attribute. Furthermore, this makes the program more robust and less prone to logical errors since attributes will only be changed once the inputted values have been verified to be an accepted input. Moreover, this programming approach allows for inheritance between parent and child classes. Inheritance is when the child class is given access to the same methods and attributes as the parent class. This permits children's classes to have the same functionality as their parent class but also allows them to become specialised (having different methods and attributes unique to the child class). In addition, inheritance allows the fast deployment of changes throughout a hierarchy since a change in a parent class will update all instances of that class but also the children

of the class. This helps developers make all-encompassing changes to many objects, through only an update of the parent or base class. However, object-oriented programming adds an extra level of complexity since many objects and hierarchies can make it harder for developers to locate, debug and solve a problem in the code. This poses a challenge as problems are not as easily solved with this programming approach, leading to an increased development time.

However, overall, object-oriented programming is the best programming approach for me and the stakeholders since it is easier to deploy updates to the program but also due to the overall robustness of the approach. Furthermore, the robustness of the program is important since children will be using the application to communicate and maintain friendships through the website. Therefore, the website not functioning due to a bug will affect the children negatively. Therefore, object-orientated programming will permit me to navigate past this problem since private attributes are only changed when the user's input is verified using the object's public method.

Programming Language

The programming language of this project is especially important as each language has its limitations and strengths. Therefore, I plan to compare JavaScript and Python as my two potential programming languages for this project as they are suitable languages for me to use

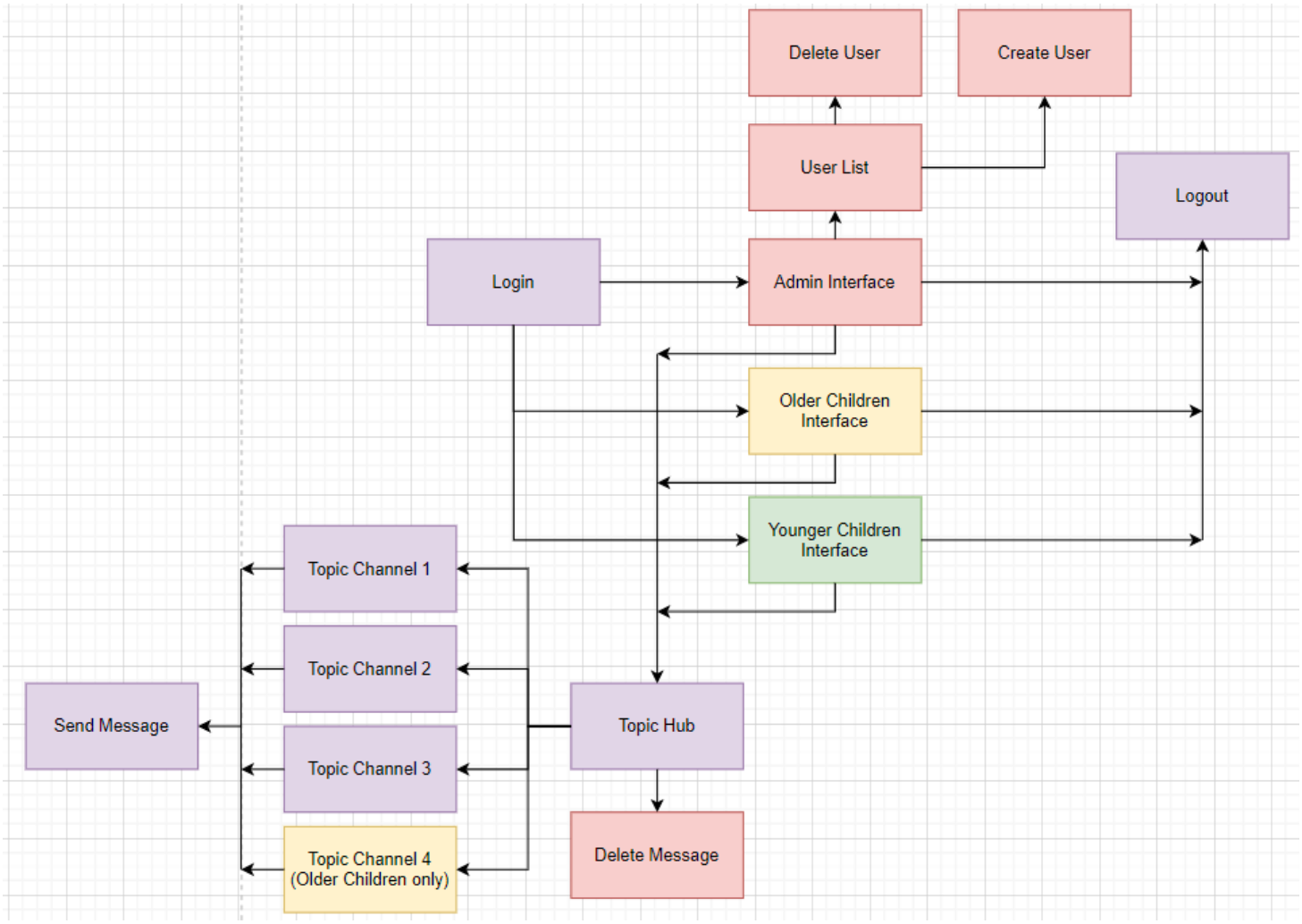
Python is the most widely used language therefore it can run on the widest range of devices, applications, and platforms. Moreover, python is easier to understand and simpler than JavaScript resulting in a quicker development cycle since bugs are easier to identify and solve. Another advantage of Python is its great support and community-made resources. At its despoil, python has a huge library that can be added to the program to add functionality but also increase code quality and save development time. However, python has a slow execution speed forcing low-specification users to experience and slower and less responsive application. Furthermore, due to Python's high memory consumption low-specification devices could struggle to run, due to memory limitations.

Now, JavaScript is more widely used for network applications (which my project is) due to its reliability, speed, and scalability. Moreover, JavaScript is reliable since it interacts with client-side and server-side behaviour permitting both to communicate flawlessly. Furthermore, JavaScript executes faster than Python due to its multithreading ability allowing the application to run fast even on lower-specification devices. In addition, JavaScript is an industry standard which means cross-platform functionality and integration with already pre-existing browsers. This broadens the accessibility of the website to a range of different browsers like Firefox and Brave. However, JavaScript is a very vulnerable programming language, which can pose a security risk for the stakeholders.

Overall, JavaScript is the best of the two languages due to it being an industry standard for web development. Moreover, vulnerabilities like security in JavaScript can be addressed using external plugins, its large ecosystem of tools and the use of HTTPS to encrypt packets. Ensuring data integrity and security. In addition, JavaScript more closely aligns with my success criteria of a complete backend and frontend capability (client-side, server-side) and my lower-specification hardware/software requirements. Finally, JavaScript is a versatile web development tool which will help me as the programmer to streamline the development and design of this project. Resulting in a quicker development time and more time allocated to debugging and test. Allowing me to give a more polished and complete finished product to my stakeholders.

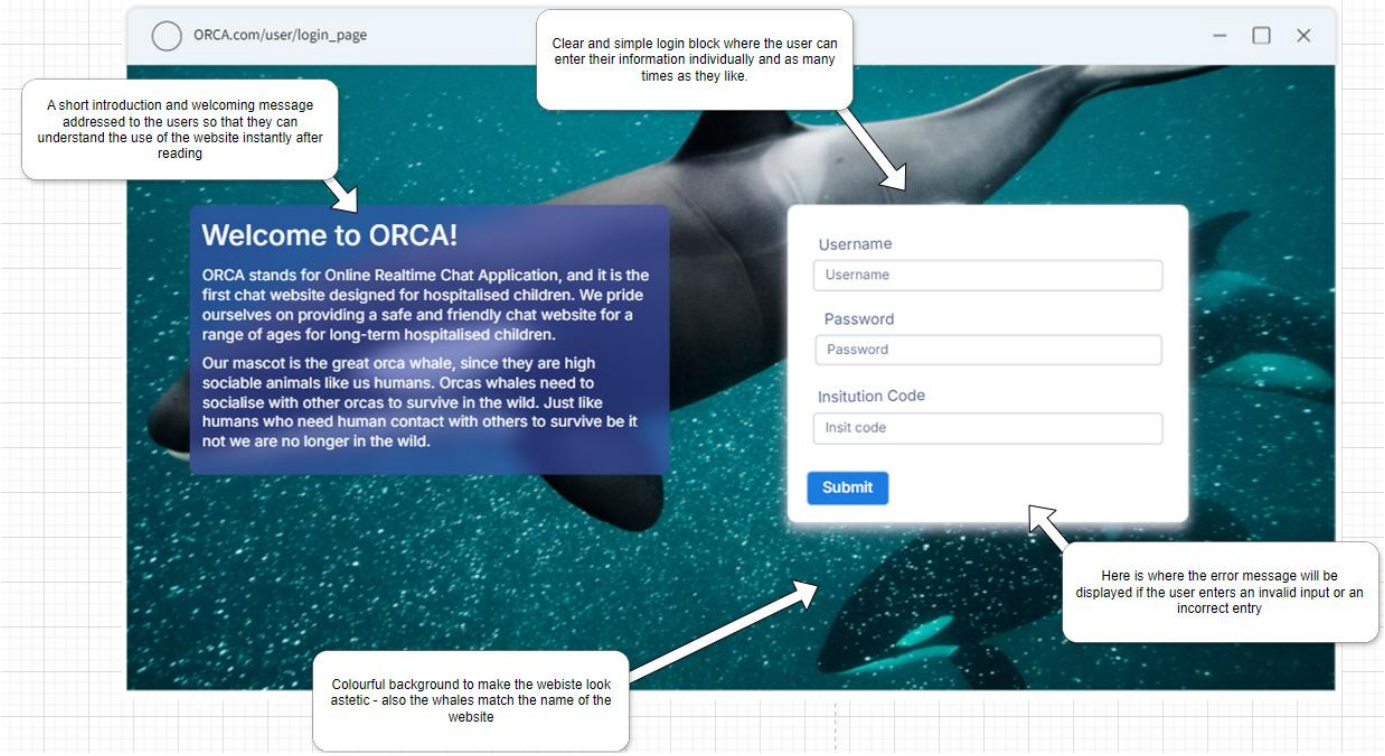
Abstracted Solution (Top Level Structure)

The website will be structured into 3 distinct levels: Admin, Older Children (+13), and Younger Children. The purpose of these access levels is to safeguard the children, while still allowing them to be able to interact with other children their age. Furthermore, the website must have a centralised hub where all other topic channels can be accessed. This will create a modular website, where new topic channels can be created or deleted without affecting other topic channels. Also, the different levels of access assist in the moderation of the website, since each level can be hidden or revealed to a particular topic channel. Moreover, allowing for each user to view their full access and all the various available topic channels accessible to them.



Here in the diagram, 4 distinct colours represent the different levels of access. Green is accessible to only young children, Red is admin only accessible, Yellow is only accessible to Admins and Older Children, and purple is accessible to all. The admin can delete messages and delete users which will aid in the moderation of the website. Moreover, to further strengthen the moderation of the website, all users can only be created by the admin. The purpose of this is to prevent the creation of unauthorised accounts and help ensure that there will be only one account per user.

Abstracted Solution (Login Authentication)

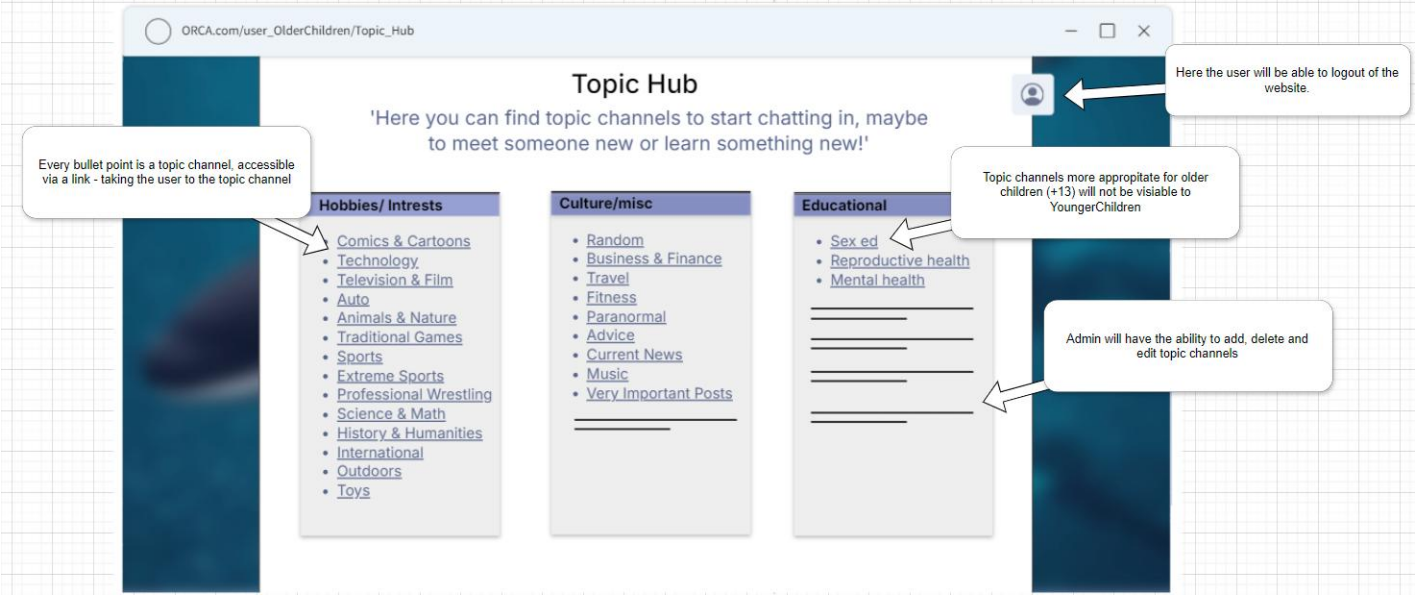


The user will be sent to this login page when they enter the URL or join via an embedded web link. The user has 3 fields to fill in, their username, password and inst code, with a login button to submit the entries. The login page will allow the user to be able to enter their password, username and int code multiple times, each time with an outputted error message displaying to the user what incorrect entry they have entered.

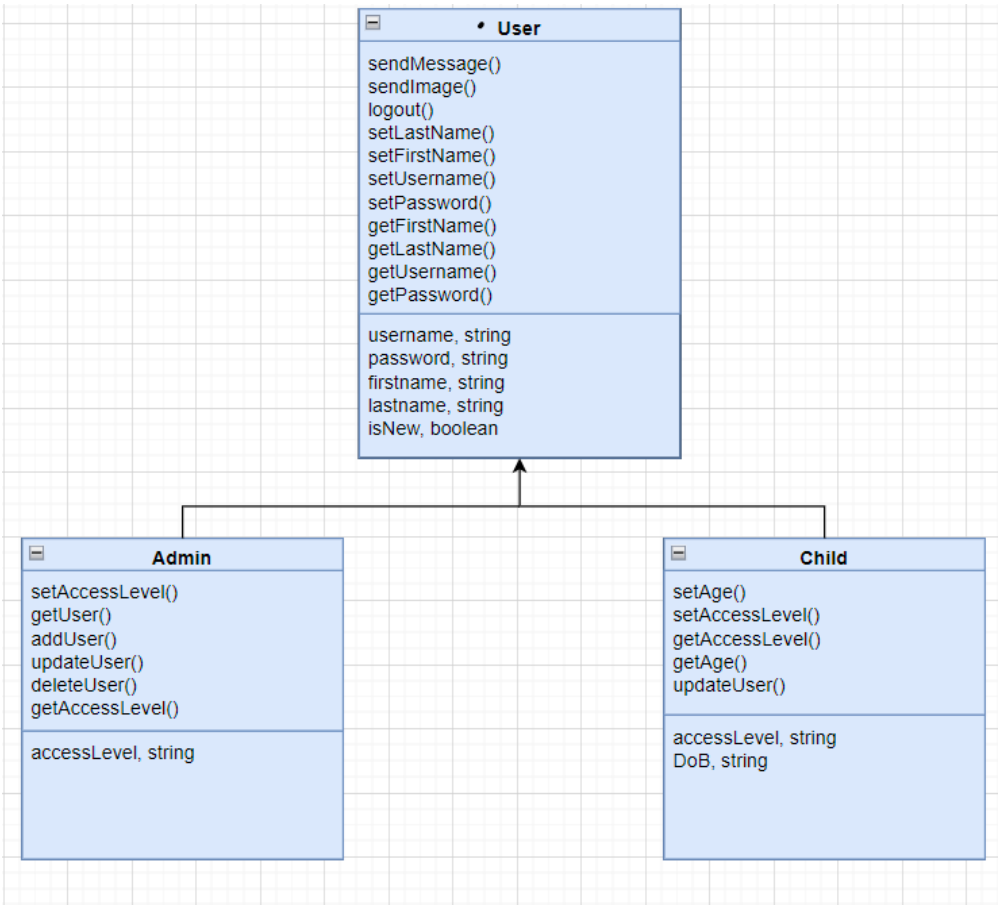
This is the logic of the login page; it utilizes two encrypted databases which are used to verify the data entered by the user. Each breach of the flow chart is a contained loop, therefore for each incorrect entry, the user will need to input all the entries again. Furthermore, as shown above, error messages will be displayed to notify the user of any incorrect entries. Finally, the access level of the user is pulled from the user data table where the corresponding interface is displayed for the user.

Abstracted Solution (Topic Hub)

The topic hub is the central access point to all the topic channels. The user can click on the links to take them to a topic channel. To make the site inclusive to all ages of children there are multiple levels of access, where some topic channels are hidden. For example, in the GUI below, the sex ed topic channel would be hidden for YoungerChildren users. Moreover, topic channels can be modified by admins, giving administrators full control over the channels, to help safeguard them.



Abstract Solution (Class Diagram)



User				
Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
username	attribute	string	“RedRider_123!”	This attribute stores the username of the user, this value will be private so that the getUsernane() and setUsername() functions can be used to validate the changes to the attributes. Furthermore, the username will need validation because, the username must be at least 4 characters long but no longer than 24 characters, it also must contain at least 2.
password	attribute	string	“Red_Apples12!”	This attribute stores the password of the user, this value will be private so that the getPassword() and setPassword() functions can be used to validate the changes to the attributes. Furthermore, the password will need validation because, the password must be at least 12 characters long but no longer than 24 characters, it also must contain at least 2 numbers and one special character.
first name	attribute	string	“John”	This contains the first name of the user, this attribute will be retrieved and updated using the getFirstName() and setFirstName() functions since the attribute is private. Also, the value of this attribute will be encrypted therefore it will only be readable through these functions.
lastname	attribute	string	“Doe”	This contains the last name of the user, this attribute will be retrieved and updated using the getLastName() and setLastName() functions since the attribute is private. Also, the value of this attribute will be encrypted therefore it will only be readable through these functions.
isNew	attribute	boolean	False	This determines whether the user is new to the website or a returning user. This attribute will be private and only accessible by calling the updateUser(), where there value of the attributes will be returned once the rest of the process of the function has taken place.
sendMessage()	function	N/A	N/A	This function will allow the user to send messages, using the website. The inner working of the functions is explained below.
sendImage()	function	N/A	N/A	This function will allow the user to send images, using the website. The inner working of the functions is explained below.
logout()	function	N/A	N/A	This function will disconnect the user from the server and bring them back to the home page.
setLastName()	function	N/A	N/A	This function will set the last name of the user. However, once the value is set, it is encrypted

User				
				so that the real last name of the users will not be accessible and readable. Finally, this encrypted value is set to the lastname attribute.
setFirstName()	function	N/A	N/A	This function will set the first name of the user. However, once the value is set, it is encrypted so that the real first name of the users will not be accessible and readable. Finally, this encrypted value is set to the firstname attribute.
setUsername()	function	N/A	N/A	This function will return the username of the user.
setPassword()	function	N/A	N/A	This function will set the password of the user. However, once the value is set, it is encrypted so that the password of the users will not be accessible and readable. Finally, this encrypted value is set to the lastname attribute.
getFirstName()	function	N/A	N/A	This function will return the first name of the user. The function will decrypt the attribute firstname before returning the value. This will keep the first name of the users safe and inaccessible unless called by an account verified permissions. How the value is decrypted I explained below in the database section of this project.
getLastName()	function	N/A	N/A	This function will return the last name of the user. The function will decrypt the attribute lastname before returning the value. This will keep the last name of the users safe and inaccessible unless called by an account verified permissions. How the value is decrypted I explained below in the database section of this project.
getUsername()	function	N/A	N/A	This function will return the username of the user.
getPassword	function	N/A	N/A	This function will return the password of the user. The function will decrypt the attribute password before returning the value. This will keep the passwords of the users safe and inaccessible unless called by account-verified permissions. Overall, this decreases the risk of accounts being compromised. How the value is decrypted I explained below in the database section of this project.

Admin				
Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
accessLevel()	function	N/A	N/A	This will check the ID of the user with the user data since the first 5 IDs are given administrator access level. This will make it so that there can only be 5 admins at any time, and these accounts cannot be changed/modified since they will be hard-coded in the program.
getUser()	function	N/A	N/A	This will return the ID of the user from the username of the user as the parameter.
addUser()	function	N/A	N/A	This will update the user data database, adding in a new account in the next available space. The parameters of this function will be username and password. This will allow the administrator to set the password and username of the user before they join the website. This will help with keeping track of the accounts and their passwords as well as maintaining that the usernames of users follow the guidelines of the website.
updateUser()	function	N/A	N/A	This will allow the admin to modify the record of the user in the database. The first parameter is the ID of the user, then the second parameter is which record will be changed. Finally, the third parameter will be the new value that will updated in the record.
deleteUser()	function	N/A	N/A	This will delete a user from the user database while keeping the ID intact. This function will not affect any other records in the database but the single record being deleted. Therefore the ID of the user can be reused.
accessLevel	attribute	string	“Administrator”	This attribute will contain the access level of the user.

Child				
Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
accessLevel	attribute	string	“OlderChildren”	This attribute will contain the access level of the user.
DoB	attribute	string	“08/10/2006”	This attribute contains the date of birth of the user.
setAge()	function	N/A	N/A	This function will set the DoB of the user. This function can only be called once.
accessLevel()	function	N/A	N/A	This will assign the value of access level based on the age of the user. +13 will assign the value to “OlderChildren” anything less will be “YoungerChildren”.
getAccessLevel()	function	N/A	N/A	This will return the access level of the user.

getAge()	function	N/A	N/A	This will return the age of the user, by calculating the age from the DoB relative to the current date.
updateUser()	function	N/A	N/A	This function will check all the various flags of the user object. It will call, getLastName(), getUsername() and getAge() and check if the attributes are not empty. Finally, it will set the isNew attribute to False, and then return the isNew attribute. The purpose of this function is to check whether the last name, first name and DoB of the user have been changed. As in this program, the user can only set their last name, first name and DoB once. This is to prevent the user from changing their name or age to imitate another person or to masquerade as younger or older. Since the access level is dependent on the age of the user - YoungerChildern(-13) or Olderchildern(+13)

Class Method – sendMessage()

```
FUNCTION OnClickSendButton
    // Retrieve the value from the input field
    message ← GetValueOf(messageInput)

    // Check if the message is not empty
    IF message IS NOT EMPTY THEN
        // Emit the 'chat message' event with the message
        EmitToSocket('chat message', message)

        // Clear the input field
        SetValueOf(messageInput, '')
    END IF
END FUNCTION
```

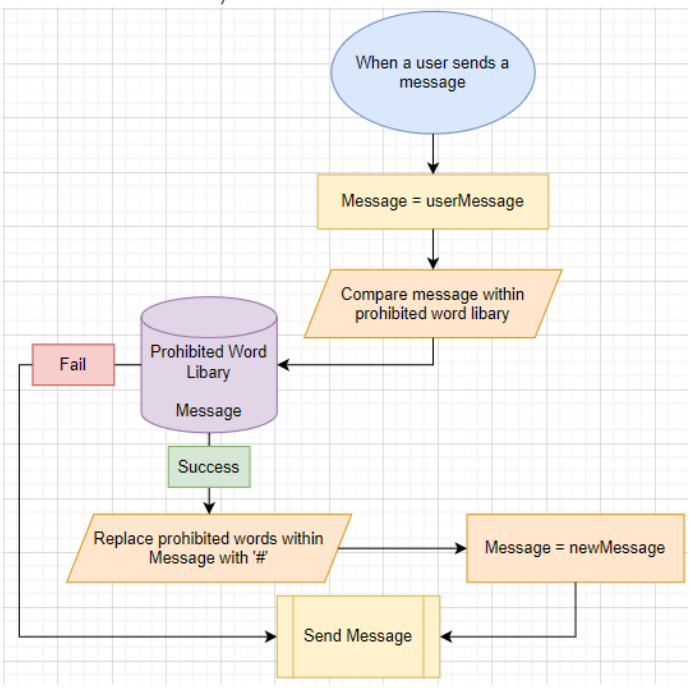
Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
message	variable	string	“Hello my name is Daniel”	This is a variable that is assigned to the input of the user. This input is the message that the user would have typed.
EmitToSocket()	function	N/A	N/A	This will send the message to the server if the message is not empty. Furthermore, the first parameter details the type of event that is firing when this function runs so that the server can know how to handle the data sent to it.
SetValueOf()	function	N/A	N/A	This will clear the content of the HTML label so that after the message has been sent the user’s message bar is cleared.

Class Method – sendImage()

```
FUNCTION HandleImageUpload(file)
    // Create a new FileReader object
    reader ← CreateFileReader()

    // Define the onloadend event handler
    reader.onloadend ← FUNCTION
        // Get the Base64 encoded image
        base64Image ← GetResultOf(reader)

        // Emit the image to the server
        EmitToSocket('image', base64Image)

        // Clear the input field after sending
        SetValueOf(imageInput, '')
    END FUNCTION

    // Convert the image file to Base64
    ReadAsDataURL(reader, file)
END FUNCTION
```

Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
reader	variable	object	N/A	This is the variable that is assigned to the value of the encoded image.
reader.onloadend	function	N/A	N/A	This is the event handler for the reader. It will run a function when the event is triggered (The event is when the file has finished being read).
Base64Image	variable	string	iVBORw0KGgoAAAANSUhEUgAAAUA...	This stores the value that the function GetResultOf() returned, which is the Base64-encoded format of the image.
CreateFileReader()	function	N/A	N/A	This creates an object of the FileReader API, this API is what I will use to be able to make the image readable.
GetResultOf()	function	N/A	N/A	This will return the image file but now base64-encoded from the reader. Base64-encoded is the network format by which images are transmitted.
EmitToSocket()	function	N/A	N/A	This will send the image to the. Furthermore, the first parameter details the type of event that is firing when this function runs so that the server can know how to handle the data sent to it.
ReadAsDataURL()	function	N/A	N/A	This will encode the image into a Bas64 data URL, this function once complete will then cause the reader. onloadend to fire.
SetValueOf()	function	N/A	N/A	This will clear the content of the HTML label so that after the image has been sent the user's message bar is cleared.

Abstracted Solution (Chat Moderation)



To moderate the website, I will be using a chat filter that will check every message sent by the users before sending the message to the server. First, the message is passed through into the 'Prohibited Word Library' where the message will be compared against the contents of the library. If there is a prohibited word found within the message, then the prohibited word is censored using '#'. For example, if the dog was a prohibited word, then Hotdog would become Hot###. This is simple and effective in preventing users from using inappropriate language and safeguarding the children. Furthermore, this would put less demand on the server since the filter will pass through the message on the client side before it reaches the server.

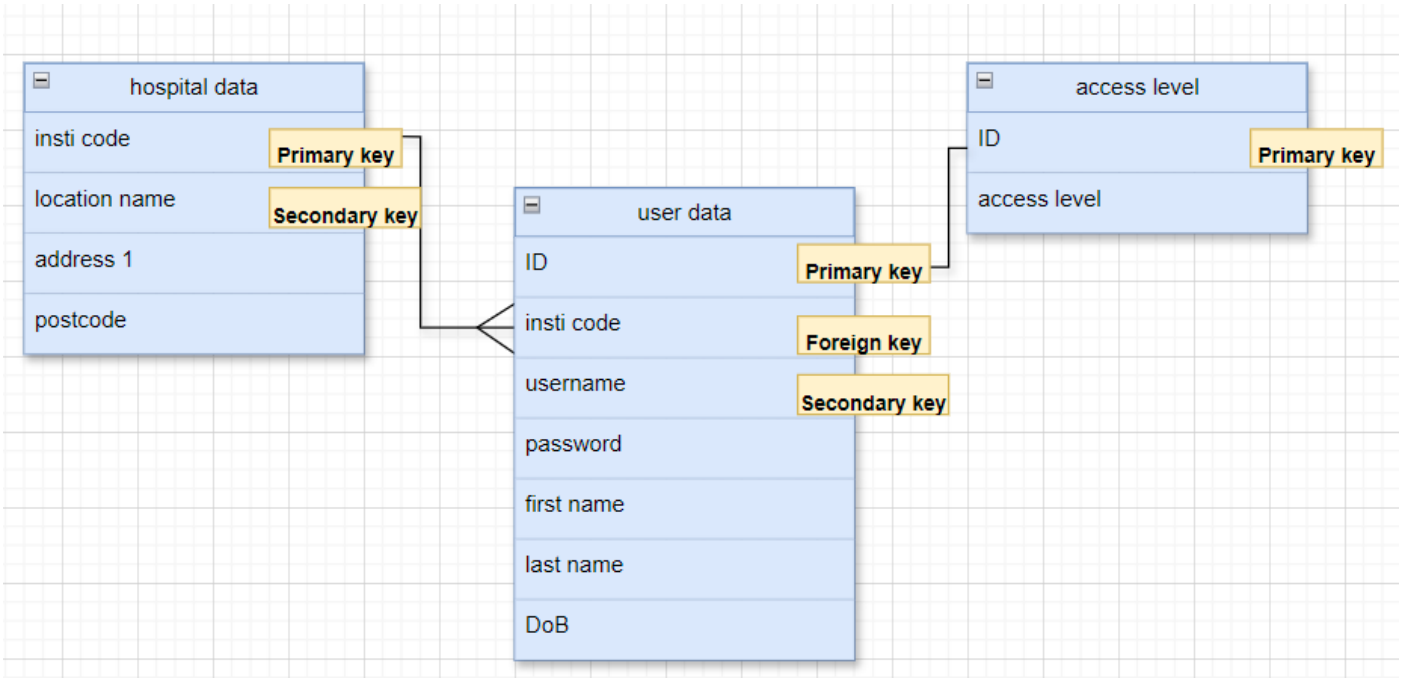
The send message subroutine is an abstracted version of the communication between the client and the server. Below I show how this abstraction would run using pseudocode, to facilitate communications between users on the server.

```
// Function to censor banned words
FUNCTION censorMessage(message)
    SET censoredMessage TO message // Start with the original message
    FOR EACH word IN bannedWords DO // Loop through each banned word
        SET regex TO new RegExp(word, 'gi') // Create a regex for the banned word ('gi' for global and case-insensitive - it will flag even if the word is capitalized)
        SET censoredMessage TO censoredMessage.replace(regex, '#'.repeat(word.length)) // Replace banned word with '#' characters
    END FOR
    RETURN censoredMessage // Return the modified (censored) message
END FUNCTION

CALL io.on("connection", FUNCTION(socket)
    OUTPUT "A user connected" // Log when a user connects

    CALL socket.on("disconnect", FUNCTION()
        OUTPUT "User disconnected" // Log when a user disconnects
    END FUNCTION)

    CALL socket.on("chat message", FUNCTION(msg)
        SET censoredMsg TO censorMessage(msg) // Censor the message
        OUTPUT "message: " + censoredMsg // Log the censored message
        CALL io.emit("chat message", censoredMsg) // Send the censored message to all connected clients
    END FUNCTION)
END FUNCTION)
```

Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
censoredMessage	variable	string	Hot###	This variable holds the censorsedMessage, outputted by the censoredMessage() function. The censoredMessage is not dependant on whether there is a banned word or not. Even if there are no banned words in the sentence, the message will be unaltered as the variable censoredMessage
bannedWords	array	string	[‘crap’, ‘cheese’]	This is an array of banned words, that can be modified to call an external flat file of banned words so that it can be easily modified by administrators. Moreover, this array can be built into the server, which is an increased level of security as the banned words array will not be accessible by the users.
regex	variable	string	‘crap’	This variable contains all the matches within the message that are also within the banned words array. This variable is how we replace the banned words with ‘#’ since we first must identify every case where the various banned words occur. Furthermore, the function RegExp() will match every occurrence of the various banned words within the message. It will then output all the matches. Moreover, the second parameter of the function is how its modes are determined – ‘g’ means that it will look for every case of said banned word within the message, therefore finding every occurrence of the word. The ‘i’ will make it so that even if the banned words within the message are capitalised in any orientation, the function will still flag its existence.
socket.on()	function	N/A	N/A	This the the client's side instance of the socket library. The method .on() listens and waits for events from the server. Likewise, when the event ‘chat message’ is emitted by the server to all the clients, the clients will run this function. The function that runs is the censoredMessage() function that takes in the message sent by the client as the parameter. The output of the functions is a censored version of the message if it contains censored words.
censoredMessage()	function	N/A	N/A	This function contains how the messages are censored. First, it takes in the parameters of the client’s message. Next, it assigns the string of messages to the variable censoredMessage. Next, the function loops through all the banned words within the banned words array. Here all the banned words will be compared with censoredMessage. Then we find every match within censoredMessage where there is a banned word. We store the matches in the variable regex. After that, we then replace all the matches

				found within censoredMessage by using regex to identify the matches. The replaced matches are replaced with '#' equivalent to their length. Finally, we repeat this for every word within the banned word array till we have exhausted the list. Once this whole process has been completed we then return the censored message that will now abide by the rules of the website.
io.on()	function	N/A	N/A	This is a server-side, instance of socket.io that will listen for when clients emit to the server. In this case, when the client connects to the server a function runs. Here the function will output to the command log that a client has connected to the server.
io.emit()	function	N/A	N/A	This function is a method of the socket.io instance. When called all the clients connected to the server are sent the same method. The 'chat message' string declares what type of event will occur when the function is called. In this case, the censored message will be sent to all clients. Likewise, all messages sent by the client are verified by the server that they do not contain banned words before sending the message to the rest of the clients.

Abstracted Solution (Relational Database)



This abstract solution matches the success criteria of moderation and safety since this is what the administrators will use to help moderate the server. The user data is very important for the administrators to have access to because they must be able to interact with the user accounts. Furthermore, administrators can view user data to make informed decisions to solve problems like which accounts are problematic in the community. By this, the administrator can find the name of the user and what hospital they are at and speak with the person in person. This can add accountability to the website since you may be anonymous to other users, but administrators will know who you are.

I will use three tables to store all the data for the website. All these tables will be accessible for admin users only, giving them a clear overview of users' data. Moreover, the database is relational and normalised to the third degree, increasing the functionality and integrity of the database. Functionality such as quicker searching (assisting the admins to find specific users). Referential integrity, such as data being verified before entering the database and consistent across each table, with records being both atomic in each but also contiguous.

User data					
Identifier Name	Data structure	Data Type	Example Data	Validation? (Y/N)	Why is this validation necessary?
ID	Primary key	integer	922	Y	To insert a new user, the ID of the user must be unique. There a createUser() function will make sure that the assigned ID is contiguous with the current IDs.
in code	Foreign key	string	Kp7	Y	The initial code is made up of the first letter of the location name, the first letter of the postcode and then a random number between 0-10. However, the insti code is checked against the current insti code to check if there is a duplicate. If there is then the number will be generated again. This is to make sure that the insti code is unique but also useful as it will contain information that an admin can understand immediately.
username	Attribute	string	RedRobin12	Y	When the username is entered using the setUsername(), it will check if the username is no more than 24 characters with at least 1 capital number and 1. Finally, the username will be checked for duplicates. These requirements will help make each username unique and help each person to be identified more easily.
password	Attribute	string	Healthier_45!	Y	The password will be entered using the setPassword() which will check if the password, contains a special character(listed in the function), also the password must be at least 12 characters long, but no longer than 24 characters.
first name	Attribute	string	Daniel	N	N/A
last name	Attribute	string	Okafor	N	N/A
DoB	Attribute	string	"08/10/2006"	N	N/A

Hospital data					
Identifier Name	Data structure	Data Type	Example Data	Validation? (Y/N)	Why is this validation necessary?

Hospital data					
in code	Primary Key	string	Kp7	Y	The insti code is made up of the first letter of the location name, the first letter of the postcode and then a random number between 0-10. However, the insti code is checked against the current insti code to check if there is a duplicate. If there is then the number will be generated again. This is to make sure that the insti code is unique but also useful as it will contain information that an admin can understand immediately.
location name	Secondary Key	string	Queen's Hospital	N	N/A
address 1	Attribute	string	Romford, England	N	N/A
postcode	Attribute	string	RM7 0AG	N	N/A

Access level					
Identifier Name	Data structure	Data Type	Example Data	Validation? (Y/N)	Why is this validation necessary?
ID	Primary key	integer	922	Y	To insert a new user, the ID of the user must be unique. There a createUser() function will make sure that the assigned ID is contiguous with the current IDs.
isAdmin	Attribute	Boolean	True	Y	A developer can't set the access level to admin. There is a preset array of IDs that are admin accounts.
isYoungerChildern	Attribute	Boolean	False	Y	Depending on the age of the user, if they are above 13 years old, they will be assigned this access level.
isOlderChildern	Attribute	Boolean	False	Y	Depending on the age of the user, if they are below 13 years old, they will be assigned this access level.

A potential problem with administrators having access to user data is that there is a chance that administrators could mishandle information or use the information maliciously. That's why administrators must be trusted adults, and there must be multiple administrators so that they can hold each other accountable. Moreover, a complaints section on the website would also help to prevent the misuse of administrative powers; however, I don't plan to implement this. However, I will be encrypting the information that is entered into the database by using a decryption key. I plan on using MySQL to store the database but also decrypt and encrypt the database.

```sql
-- Insert encrypted data for first name, last name, username, and password
INSERT INTO users (first_name_encrypted, last_name_encrypted, password_encrypted)
VALUES (
    AES_ENCRYPT('John', 'my_secret_key'),          -- Encrypt first name
    AES_ENCRYPT('Doe', 'my_secret_key'),           -- Encrypt last name
    AES_ENCRYPT('password123', 'my_secret_key')    -- Encrypt password
);
```

The data inserted into the table will be encrypted to ensure, data integrity and security for bad actors gaining access to the database. Furthermore, this safeguards the children as their names are encrypted, keeping their identities private. Moreover, the username of the accounts will not be encrypted, as the username will be used to identify and search for target records.

Identifier Name	Data structure	Data Type	Example Data	What is the purpose?
first_name_encrypted	variable	binary	0x53C64A0C2B... (32-digit hexadecimal)	This is the first name of the user and will be encrypted since the users of the website are children 18 and under therefore to safeguard them I will encrypt their name as well.
last_name_encrypted	variable	binary	0x62A92F0803... (32-digit hexadecimal)	This is the last name of the user and will be encrypted since the users of the website are children 18 and under; therefore to safeguard them, I will encrypt their name as well.
password_encrypted	variable	binary	0x9F8A00C8DE... (32-digit hexadecimal)	This is the password of the user, it will be encrypted to prevent their accounts from being compromised.
AES_ENCRYPT()	function	N/A	N/A	This is an inbuilt function of MySQL, and it works by taking in a value and then using a key of binary digits it encrypt the data using a complex algorithm. Finally it returns an encrypted version of your value.
my_secret_key	constant	string	'ThisIsAVeryLongEncryptionKey256BitsLong!'	The key is a long string, the key mustn't be hardcoded into the program but must be requested by an external key manager. To request the key you will have to verify that your account is an administrator.

```sql
-- Retrieve and decrypt first name, last name and password
SELECT
    AES_DECRYPT(first_name_encrypted, 'my_secret_key') AS first_name,
    AES_DECRYPT(last_name_encrypted, 'my_secret_key') AS last_name,
    AES_DECRYPT(password_encrypted, 'my_secret_key') AS password
FROM users;
```

To retrieve the information from the encrypted keys, use must use – AES_DECRYT(), which is a function that will take the encrypted value and decrypt the value when the correct key is given as its second parameter. Moreover, all this will be accessible on the administrator panel so that only authorised users can access the unencrypted data.

```
1 -- Update encrypted data for first name, last name and password
2 UPDATE users
3 SET
4     first_name_encrypted = AES_ENCRYPT('Jane', 'my_secret_key'),
5     last_name_encrypted = AES_ENCRYPT('Smith', 'my_secret_key'),
6     password_encrypted = AES_ENCRYPT('newpassword456', 'my_secret_key')
7 WHERE id = 1;
```

Likewise, to update the data you will need to first encrypt the information then before setting the new data to the old data.

Abstract Solution (Test Plan)

The testing strategies that I plan to use for my test plan will be:

Alpha testing – I plan on entering erroneous data on the client side of the website (Entering invalid data) and then sending this data to the server. I expect to understand how the server will behave in case of encountering invalid data. Moreover, this should help me to iterate the design of the server and client relationship so that the website can be robust in handling a wide variety of different data – erroneous data and non-erroneous data. In addition, I plan to test the chat filter of the website to ensure that users' messages are being moderated.

White box testing – With this strategy, I will test the features and individual modules of the program to gauge the functionality of these programs and also ensure smooth communication between the server and the client. Furthermore, this plan will have me testing communication between the server and client by using erroneous data from the server side of the website. In addition, I will test how the server can handle multiple users and how this will affect the website.

Black box testing – Here, I will have multiple users on the website who will test-run the website. I will have these users find ways to break the program by using techniques like entering erroneous data, stress testing (overloading the server with requests) the website and concurrency testing (Where users enter data in simultaneously).

Relating to my success criteria, I will need the program to be user-friendly, the backend and frontend will need to be robust, there needs to be safeguarding and moderation, and the website needs to have real-time chat functionality. My test plan will cover these criteria by the strategies shown above and the test plan shown below.

Iterative Test Plan

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
1.	Ensuring the layout of the website scales properly with differing amounts of text.	Entering dummy text into the text boxes and labels of differing sizes. By typing in 'Lorem' into text boxes it will generate dummy text, by typing 'lorem' multiple times we can get more dummy text.	Boundary	The website should be able to handle varying amounts of text and scale accordingly within predefined limits.
2.	Ensure that the subpage of the website corresponds with their links.	To ensure that the subpages are corresponding to the links, I will input the URL of the subpages of the website into the address bar like – '/hub', '/tchannel1', '/tchannel2' etc.	Normal	When I enter the URLs into the address bar, they should take me to their corresponding pages.
3.	Ensure that the user can log in to the website.	I will enter valid data into the respective text box. For example: Username - "johndoe " Password - "Password123" Insti Code – "Kr7" But I will also enter invalid data like: Username - "sdfnjpwe;ogubmis;" Password - "adfn@As'dasd'wqd" Insti Code – "sdfnuisjdfpmnsdk;"	Invalid/ Normal	*The valid data should take the user to the hub pages, but invalid data should clear the text box and then output an error message: "Invalid input"*
4.	Ensure that the user can access the website on a desktop with differing screen sizes/resolutions.	By using a virtual machine, I can simulate different desktop screen sizes. Here, I will use a range of commonly used desktop screen sizes/resolutions. Like: 1920×1080, 1366×768, 1280×1024, etc.	Normal/ Boundary	The website should scale accordingly to the different desktop sizes.
5.	Ensure that the user can send messages.	I will input a variety of different messages into the text boxes of the topic channels. For example: Normal - "Hello, how are you?" Boundary - sending an empty message. But also, sending a message with special characters like "&<>" ensures that special characters can be sent as well.	Normal/ Boundary	The server should be able to receive the messages sent by the user.
6.	Ensure recording of message history and chat logs	Users should be able to see previously sent messages on the topic boards. By entering messages: "Hi, this is John." "Hello, this is Jane." From two different accounts, then log out of one of them so we can test this feature.	Normal	The server should save previous messages even after the user leaves the website. Once the user returns to the topic channel, I should be able to see: "Hi, this is John." "Hello, this is Jane."

7.	Ensuring that words are moderated/ filtered.	Entering banned words into the topic channels: “crap”, “kys”, etc I can test this feature	Normal	When the words are sent to the server, they should be filtered through the chat filter to check for any banned words. If found, these words should be replaced with “#”.
8.	Ensure that multiple users can use/ send messages on a topic channel.	By using three users, I will input: User 1 – “Hello, my name is Daniel”. User 2 – “Hello, my name is John”. User 3 – “Hello, my name is James”. All at the same time.	Normal	When the server receives all these messages, the server should output them one at a time. They should be outputted in order of the time of the server received the messages.
9.	Ensure that messages that are sent while the user is disconnected are sent once the user reconnects	By disconnecting user 1, the messages “hello” and “I think that I’m disconnected”. Then reconnecting the user back to the server.	Normal	The messages should be sent once the user reconnects to the server. We should see user 2 receive the messages once user 1 reconnects.

Post Development Test Plan			
Test No.	Test Purpose	Test Data	Expected Outcome
1.	Ensure that there is real-time messaging on the website.	Entering the messages: “Hi, how are you?” “I’m good, thanks! How about you” -Send an empty message “?@?@@/” I will then use a third user to spectate the topic channel for the messages to ensure that they are being received in real time.	The user should be able to view the messages as they are entering once the server outputs the received messages.
2.	Ensure that users can access different topic channels that are appropriate to their age.	By using a user with the access level of ‘olderChildren’ and a user with the access level of ‘YoungerChildren’ to check if topic channels for older children (13+) only are accessible to them and admins. Also, checking to make sure that the older children cannot access younger children-only topic channels	Older children trying to access younger children topic channels should be met with a pop up telling them that they cannot access the topic channel. This should happen for when younger children try to access older children only topic channels as well.
3.	Ensure that a user can interact with the website.	Clicking on buttons to test if the website outputs to the user. For example, users should be able to click on links to take them to subpages on the website. Race Cars – This should take the user to the topic channel (subpage) on race cars.	The user should be able to interact with the website when clicking on buttons. Also, when the user clicks on the link they should be taken to the corresponding subpage (topic channel).

4.	Ensure that can't use inappropriate language.	Entering banned words which are deemed inappropriate (They are stored in an external library), I can test to see that inappropriate language is censored, safeguarding the children who use the website. For example, entering: 'You look like crap'. It will be sent to the server where it should be censored.	Once the message is received by the server, the server will compare it to the banned words. Checking if the message contains banned words, censoring them if found. So, in the example, the message: 'You look like crap' Will become: 'You look like ####' This is how it will be outputted to the rest of the users.
5.	Ensure that Admins can delete, create, and alter users.	Using the admin panel, I will test user creation, deletion and alteration. For example, I will enter in (in creation mode): Krp, RedRider, Password123!, Daniel, Okafor, 12/8/2007 (corresponding with the user table). This will test the user creation. Entering in: 21, Password, newpassword123! When in altering mode, it should alter the user with ID 21. Entering: "21" When in deletion mode, it should delete user 21 from the table.	When using the admin panel, the admin should be able to use the different modes to delete, create, and alter users. In the example (when in creation mode), entering in: Krp, RedRider, Password123!, Daniel, Okafor, 12/8/2007 It should be entered into the next free space in the table. In the example (when in altering mode) entering: 21, Password, newpassword123! Will change the password of the user with the ID 21 in the table to "newpassword123!". In the example (when in deletion mode), entering in: "21". Should delete the user with the ID 21 from the table.

DEVELOPMENT

In this section of my project, I will cover my four success criteria:

Moderation and user safety – Ensuring security and safety on the website is especially important for my stakeholders since they are responsible for the care of the children and young people who will use the website.

Real-time chat – Creating a website with real-time functionality will be the most important feature of the project since it is the main solution of my stakeholders.

Website functionality and user interface – This is not as complex as real-time chat capabilities, but it will pose a challenge as it can be very time-consuming to create a user-friendly website with topic channel capabilities. Furthermore, I must create a user-friendly website since young people of all ages will be using the website. However, I plan to use pre-made assets and designs to help create a good user interface to minimise time spent on graphical improvements.

Networking and backend – These are a complex feature of the project since it will be a new problem for me to solve. Fortunately, many applications like node.js can help me to manage most traffic to the website and network for the website. Node.js especially, allows me to be able to create multi-level user access and uses JavaScript as its programming language which makes it easier for me to transfer the skills I already have.

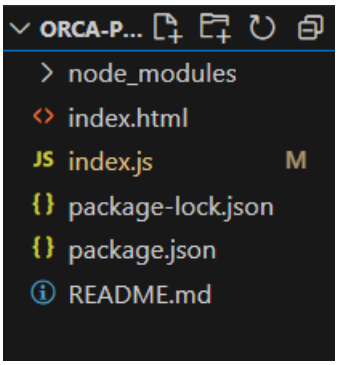
Development (Real-Time Chat Functionality)

My success criteria of real time chat functionality for the server is important because the users will need to be able to communicate with one another. The message delivery system must be responsive so that the website can imitate real life discussion. This is critical for my stakeholder's solution since this website is to create a sense of community among the user, which should help to meet my user's emotional/social needs.

To fulfil this success criteria, I created a server to host my website and developed the server-side interactions so that messages can be send to user in real-time.

```
1  const express = require('express'); // Importing the Express framework
2  const { createServer } = require('node:http'); // Importing the createServer function from the http module
3  const { join } = require('node:path'); // Importing join function from the path module
4  const { Server } = require('socket.io'); // Importing the Server function from socket.io module
5
6  const app = express(); // Creates and instance of Express
7  const server = createServer(app); // Creates a HTTP server
8  // Express (app) will handel all incoming HTTP requests
9  const io = new Server(server);
10 // Creates an instance of socket.io
11
12 // On request of the server, the server will responds with the HTML file (index.html)
13 app.get('/', (req, res) => {
14     res.sendFile(join(__dirname, 'index.html'));
15 })
```

By importing in the Express framework, I then create an instance of Express – This will act as the backend of our server. I also need to import some modules from Node.js to handle web initialisation and connectivity. After I've imported these frameworks, I then import the socket.io library to create the middleware of the server – It will handle connecting users to the server, listening to web sockets and sending messages between the server and the clients. On line 13, the function will run once a client requests the website; the server will send the local HTML file back to the client – The HTML file is the front end of the website (what the user will see and interact with)



The nodes are stored locally as well as the client for now, for easy access for the JavaScript file. Here is the file arrangement currently. The node_modules folder stores all the libraries and frameworks used in this project while the index.html file is the client side of the website (What the user will see). The index.js file is the server side of the website housing all the server program and functionality.

On the client side, there is a form which is used by the users to send their messages. Once the user presses send, the value they sent (the message) will be sent to the server. This is done by using the function on line 34, which runs once the form has been submitted. It will check first if the message that they are sending isn't empty then if it's found to not be empty the message is sent to the server.

```
21      <form id="form" action="">
22        <input id="input" autocomplete="off" /><button>Send</button>
23      </form>
24      <script src="/socket.io/socket.io.js"></script>
25      <script>
26          const socket = io(); // Reference to the server (Client-side)
27
28          const form = document.getElementById('form');
29          const input = document.getElementById('input');
30          const message = document.getElementById('messages'); // Container for the recieved messages
31
32          // Once the submit button is pressed, the client checks if the message is not empty.
33          // Then the client sends this message to the client
34          form.addEventListener('submit', (e) => {
35              e.preventDefault();
36              if (input.value) {
37                  socket.emit('chat message', input.value);
38                  input.value = '';
39              }
40          })
```

By using the server.emit() function we can send the message to the server with the event name 'chat message' so that the server will know how to handle the response.

This is the form where the user can send their message.

Next, I created the server-side interaction that will listen and wait for a user to send a message, once a user connected to the server. The function on line 25 handles the sending all the messages to all the users once the 'chat message' event has been received from a client.

```
17  // When there is a connection to the server this event will fire
18  io.on('connection', (socket) => {
19      console.log('A user connected');
20
21      socket.on('disconnect', () => {
22          console.log('User disconnected');
23      })
24
25      socket.on('chat message', (msg) => {
26          io.emit('chat message', msg);
27      })// Sends the chat message to all users
28  })
29
30  // Here we make the server listen for connections to port 3000
31  server.listen(3000, () => {
32      console.log('server running at http://localhost:3000');
33    });
```

Once the message has been sent to all the users connected to the website, the users will be able to see the message be displayed onto their screens. The function on line 45, on the client side listens for response the from the server. If the event the server sends to all the clients is 'chat message' the client will get the value sent by the response and display it in a ordered list on the clients' screen.

```
42          socket.on('chat message', (msg) => {
43              // Creates a list element that will contain the message
44              const iteam = document.createElement('li');
45              iteam.textContent = msg;
46              message.appendChild(iteam);
47              window.scrollTo(0, document.body.scrollHeight); // Moves down the pages to fit the message
48          })
49          // Once the message is recieved, it will be displayed to the client
50      </script>
51    </body>
52  </html>
```

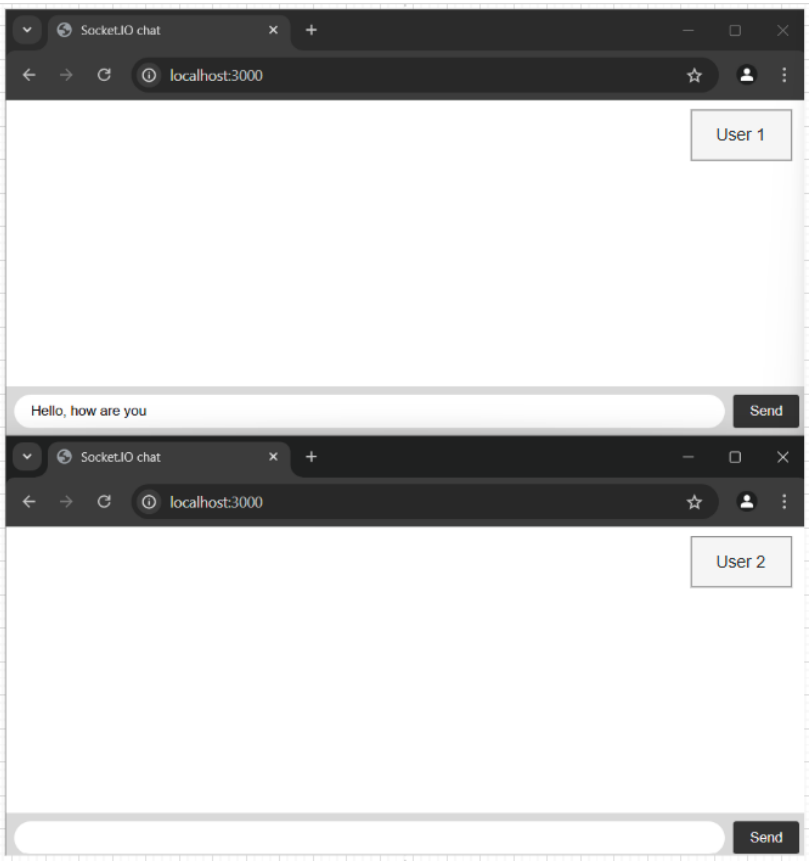
Here is what the client will see once the message is received from the server.

Iterative Test (Real-Time Chat Functionality)

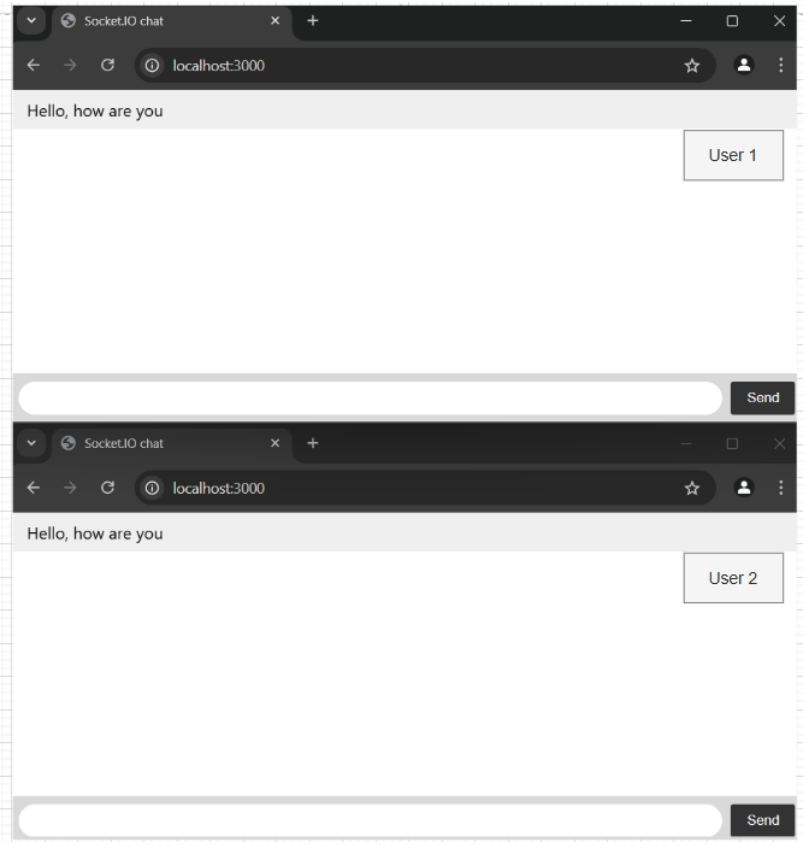
Here, I will test if users can send messages in real time and if users can send boundary messages like symbols. Also, I will test if the server flags invalid messages. This will fulfil part of real time chat functionality success criteria.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
1	Ensure that the user can send messages.	I will input a variety of different messages into the text boxes of the topic channels. For example: Normal - "Hello, how are you?" Boundary - sending an empty message.	Normal	The server should be able to receive the messages sent by the user.

Before message sent



After the message sent

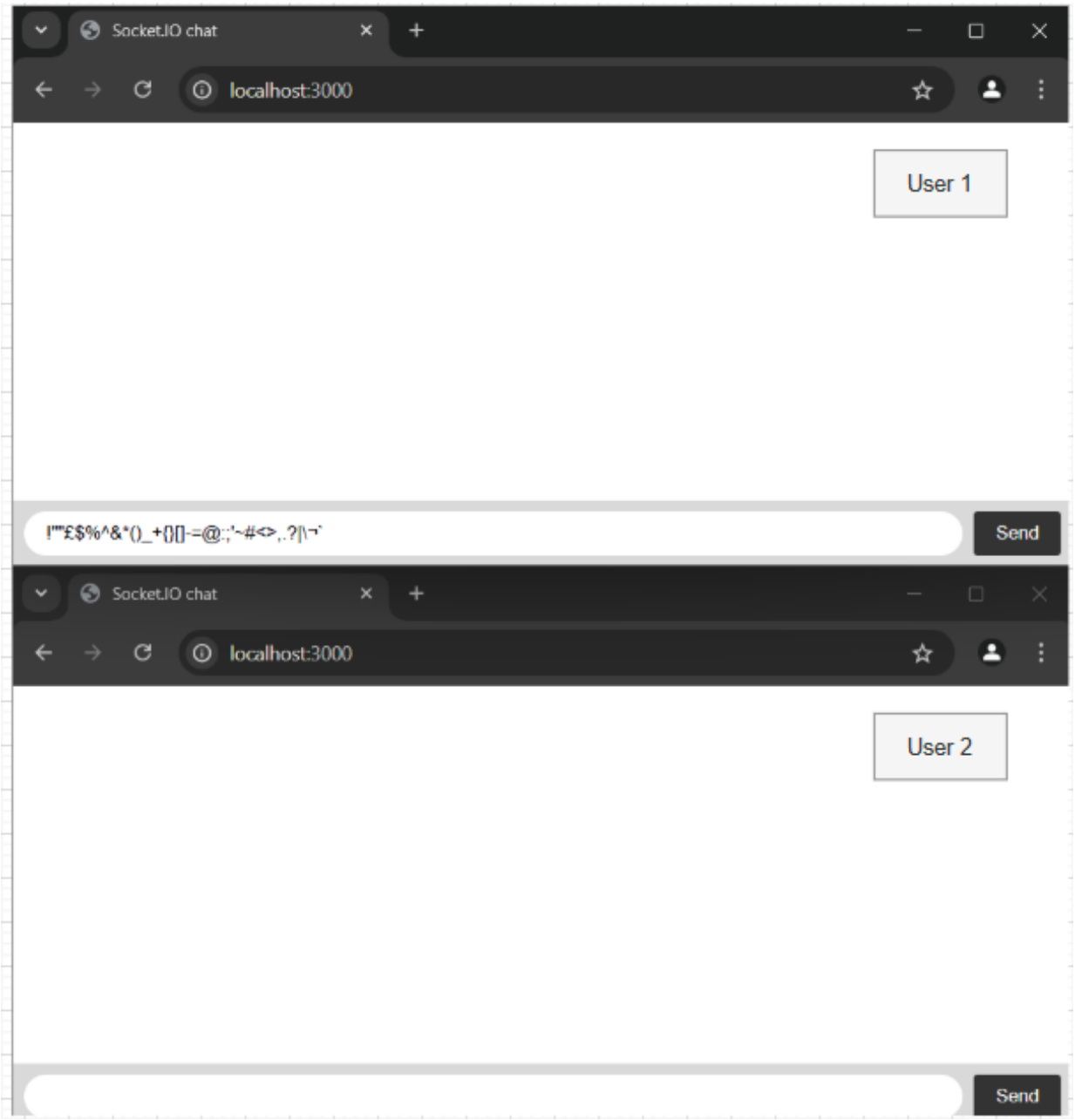


This test of normal data was successful, as both users were able to receive the message sent by User 1. Furthermore, the message sent by User 1 was within the word limit, showing that normal data is treated by the server correctly.

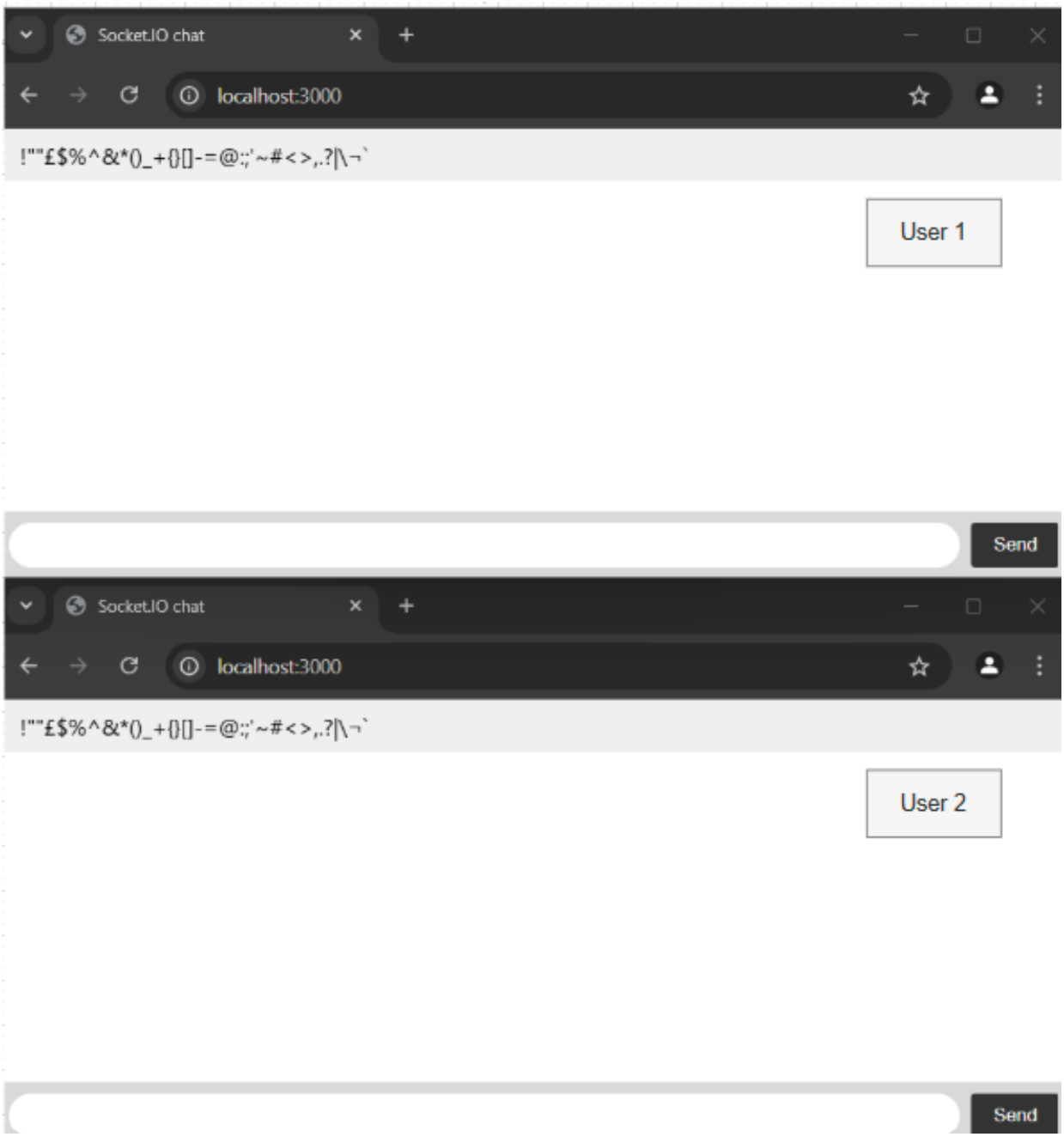
Next, I will test boundary data – special characters like '&<>'.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
2	Ensure that the user can send messages.	I will send a message with special characters like "&<>", ensuring that special characters can be sent as well.	Boundary	The server should be able to receive the messages sent by the user.

Before message sent



After the message sent

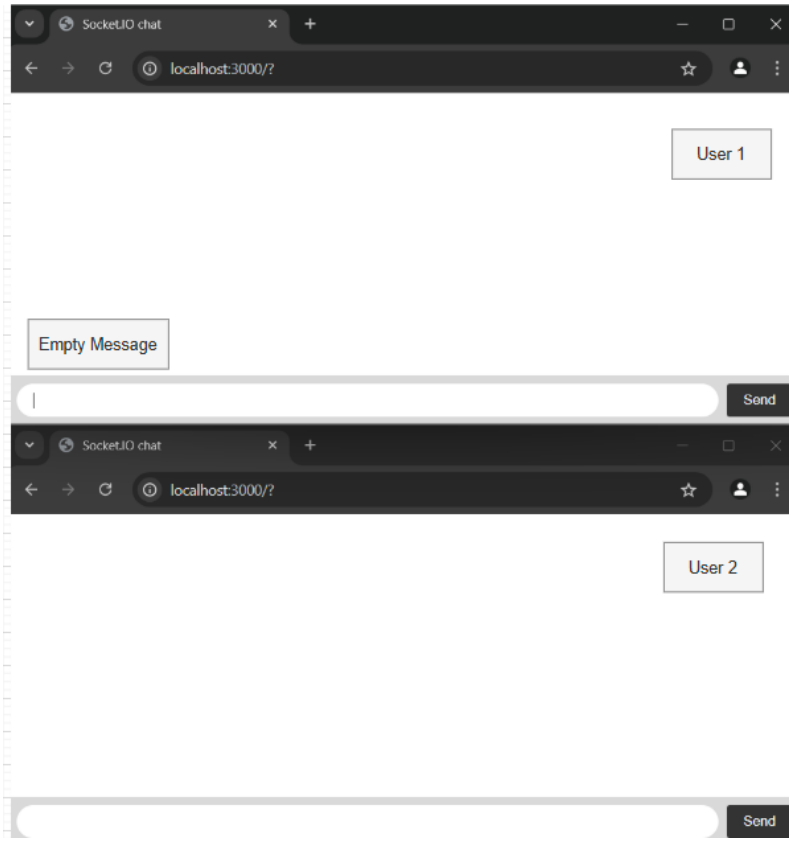


This test of boundary data was successful, displaying that special characters are accepts as long as the characters are withing the Unicode character set.

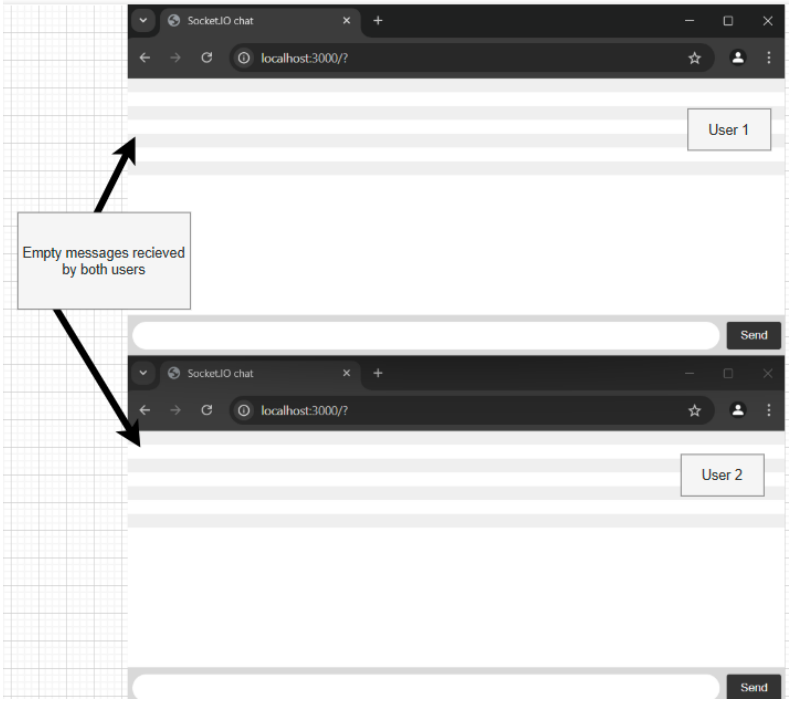
I will test how to the program will respond to invalid data – empty data. By sending multiple empty messages it will become more visible as to how the program will respond to this invalid data.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
3	Ensure that the user can't send empty messages.	I will enter no message into the input box.	Invalid	The server should not be able to receive the messages sent by the user, but the user should be notified of their invalid input.

Before messages sent



After messages sent



This test was unsuccessful, as the empty messages were received by the users. Moreover, empty messages populated the message board. This is not the outcome that I wanted as empty messages could pollute the message

board, making it crowded and cluttered. This will make it harder for many users to have discussions over the website because of the clutter from empty messages. This outcome of the test does not fulfil my stakeholder's requirement of the website being user-friendly and functional, as children will be using this website.

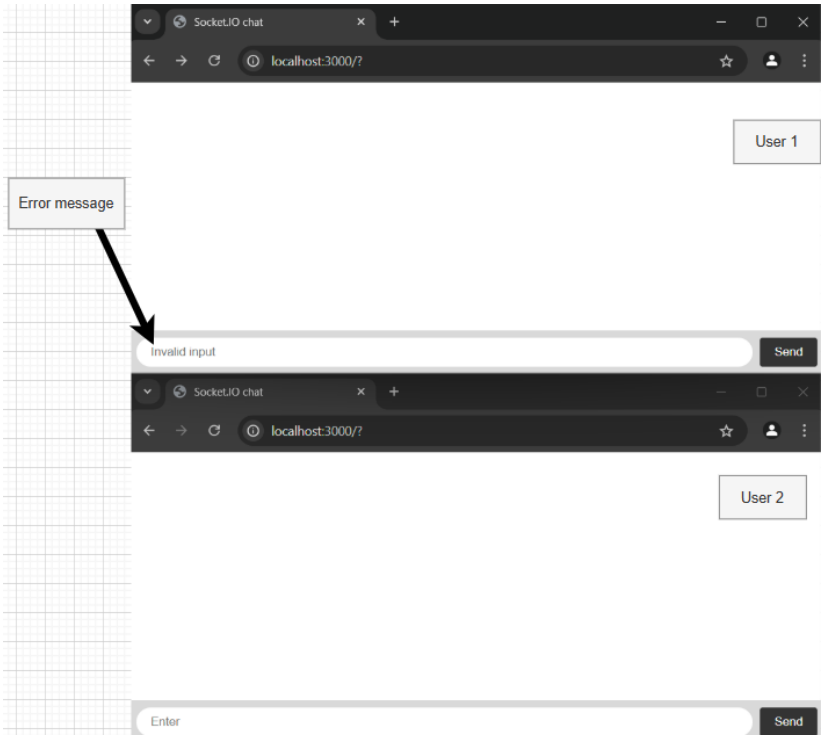
Remedial action

To correct this outcome, I will check if the message the user wants to send is empty on the client side before sending it to the server. If empty, output an error message to the user.

```
// Then the client sends this message to the client
form.addEventListener('submit', (e) => {
    e.preventDefault();
    if (input.value) { // Once the submit button is pressed, the client checks if the message is not empty.
        socket.emit('chat message', input.value);
        input.value = '';
        input.placeholder = 'Enter'
    }
    else {
      input.placeholder = 'Invalid input';
    }
})
```

Here, I check if the user has entered an empty message, then I change the placeholder text to "Invalid input" so that the user can know they have inputted an invalid input. Furthermore, the empty message will not be sent to the server where all the clients will receive the message. This will solve the issue of clutter from the empty messages appearing on the message board.

After changes

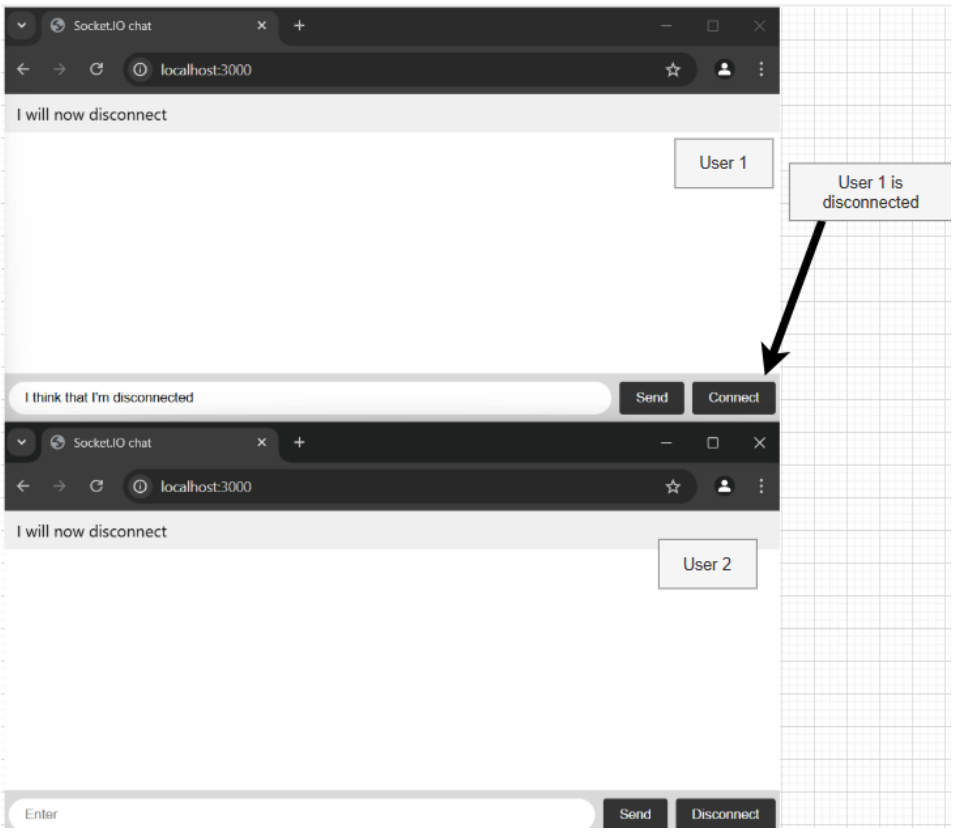


Here, you can see that no message is sent, and the user is warned discretely of their invalid input.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
4	Ensure that messages that are sent while the user is disconnected are sent once the user reconnects	By disconnecting user 1, the messages “I think that I’m disconnected”. Then reconnecting the user back to the server.	Normal	The messages should be sent once the user reconnects to the server. We should see user 2 receive the messages once user 1 reconnects.

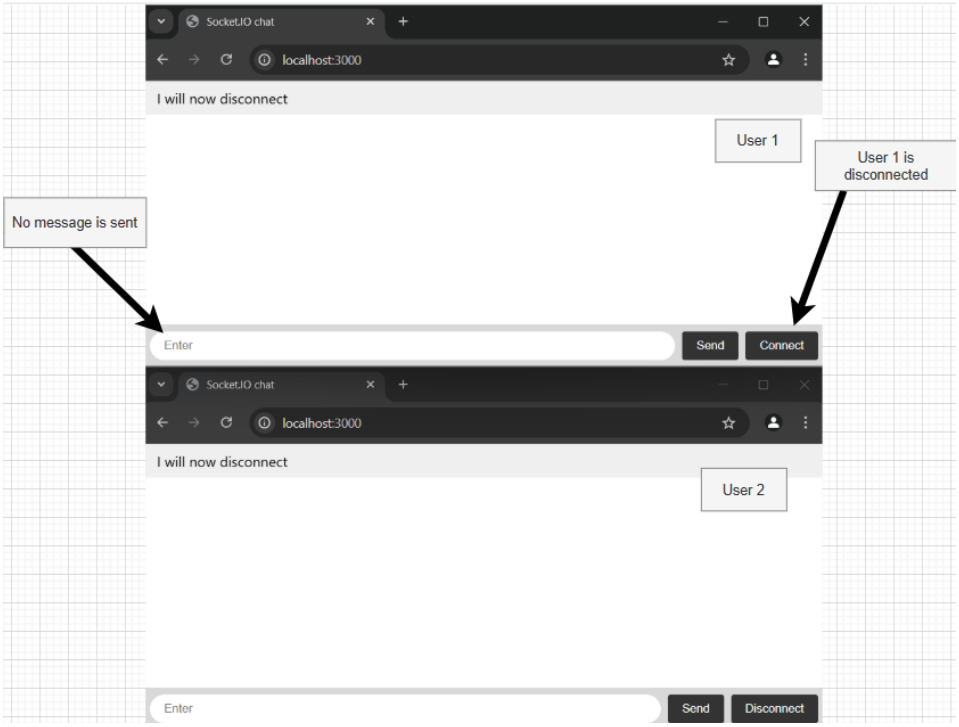
Here, I will test how the server will respond to connectivity issues, ensuring that real-time communications can be maintained even after disconnects. This test will fulfil a part of the real-time functionality chat success criteria.

After User 1 disconnects



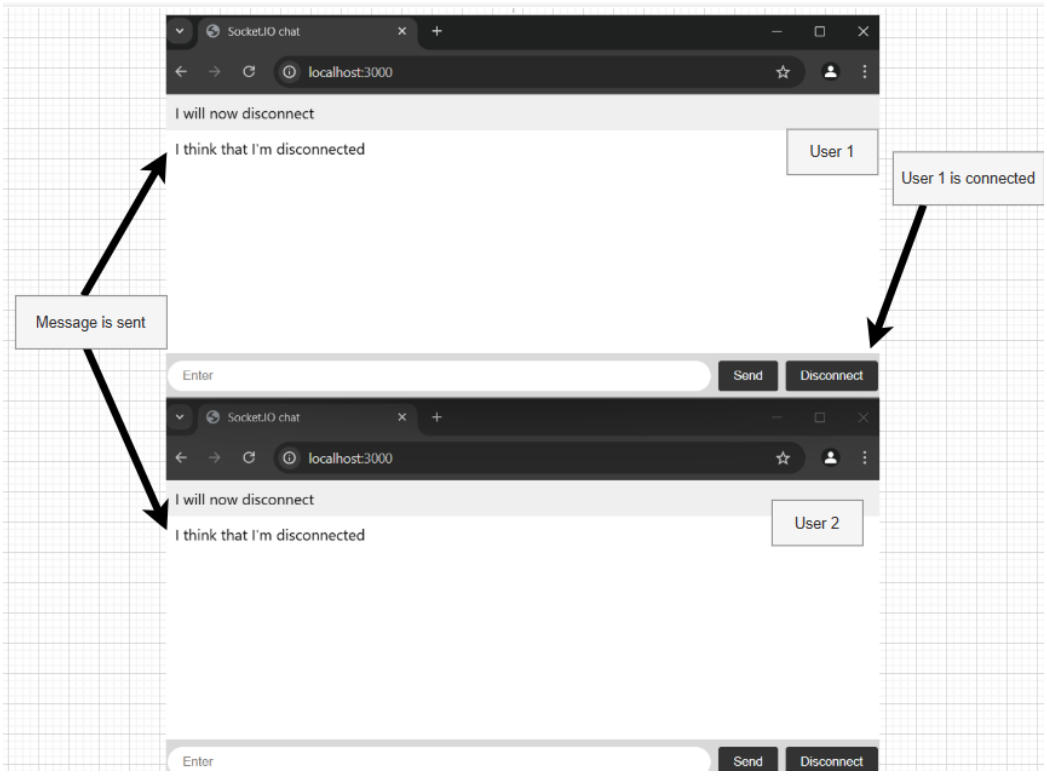
The button beside the send button is a testing feature and will not be included in the final solution.

After the message is sent while disconnected



Since user 1 is disconnected from the server, user 2 does not receive any message sent from user 1.

After User 1 reconnects



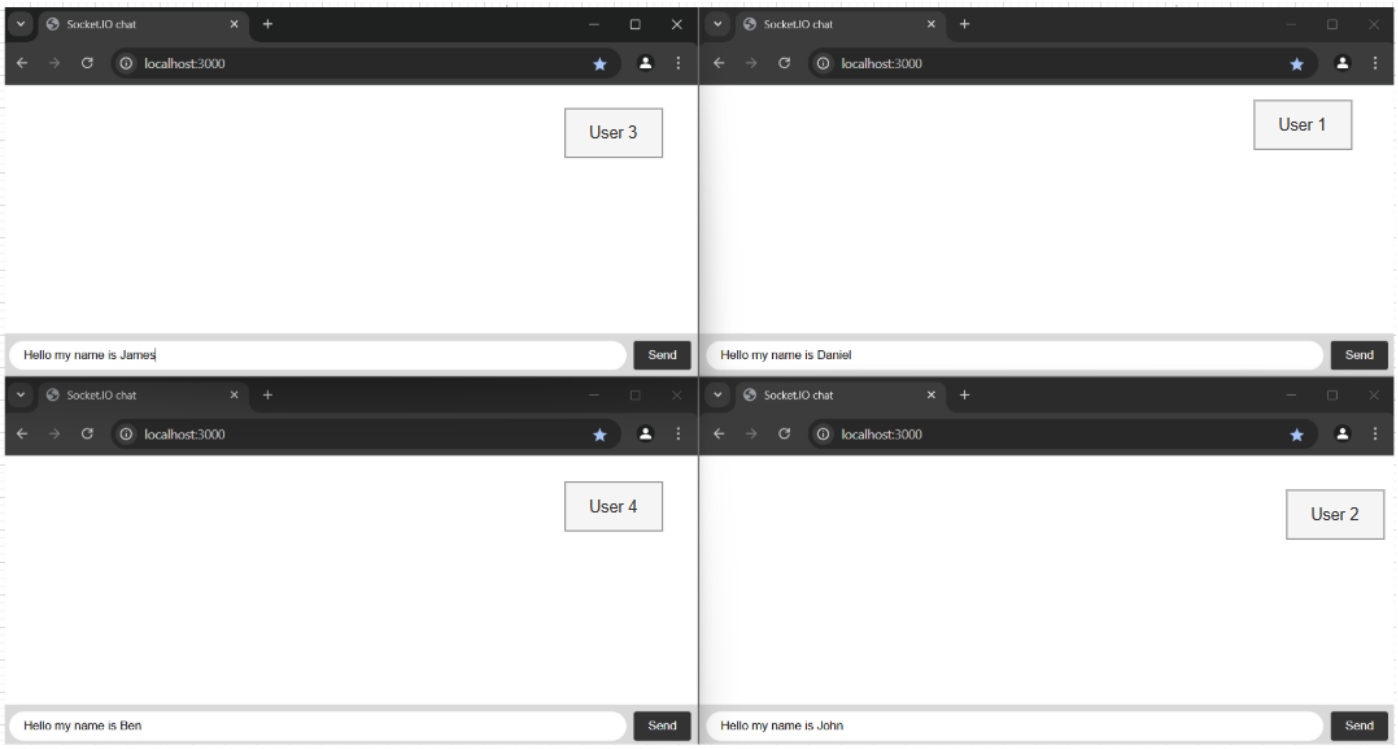
The message is then received by user 2 once user 1 reconnects to the server.

This test is relevant to my stakeholder's solution since the hospital cannot ensure a stable Internet connection indefinitely. For the quality and heath of the solution, the website must be robust in handling disruptive events like users disconnecting as it may be a reoccurring disruption. To have the server send the messages once the user reconnects to the server decease disruption cause by the disconnect improving overall functionality and maintainability.

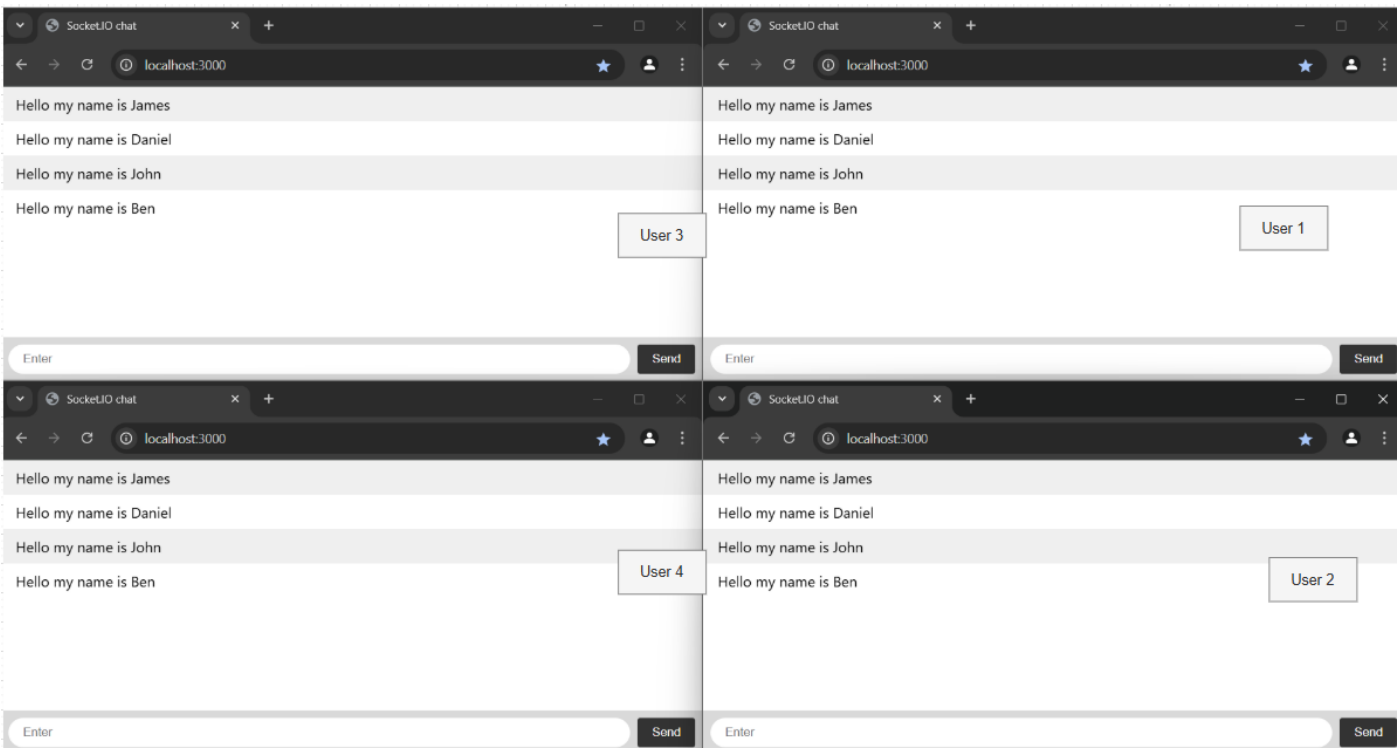
Test No.	Test Purpose	Test Data	Data type	Expected Outcome
5	Ensure that multiple users can use/ send messages on a topic channel.	By using three users, I will input: User 1 – "Hello, my name is Daniel". User 2 – "Hello, my name is John". User 3 – "Hello, my name is James". User 4 – "Hello, my name is Ben" All at the same time.	Normal	When the server receives all these messages, the server should output them one at a time. They should be outputted in order of the time the server receives the messages.

This test will be used to see if many users can communicate in real time, fulling a part of the real time chat functionality success criteria.

Before messages sent



After messages sent



Here, you can see that all the messages are received by the server. The order of the messages sent is related to the time that the messages are received by the server. In this test, User 2's messages were received first by the server, followed by User 1, User 3, and User 4. This test is important for my stakeholder's solution because many users will need to be able to chat simultaneously in real-time so that the website can imitate the feeling of a live discussion.

Chunk Review (Real-Time Chat Functionality)

Overall, implementing real-time chat functionality to the website has been an overall success. It has been a success because users can send messages to one another through the website and receive messages almost instantly. This fulfils my success criteria of having real-time chat functionality as users can now interact with one another as if it were face to face. By testing how the website will handle disconnects, allows for real time chat functionality to be maintained – Improving overall maintenance and robustness. Furthermore, by discreetly prompting the users when their message is invalid, the user can correct their error seamlessly. For my stakeholder, the criterion of real-time chat functionality is important since the website should imitate as if the users were face to face.

Development (Networking & Backend)

The success criteria of Networking and Backend can be broken down into two criteria, Networking and Backend. For the networking of the website, I will integrate network functionality like saving the state of the server so that every client connected to the server can view the same state. By syncing all the states of the clients, all users will be able to see the same messages and respond accordingly. Furthermore, in this part, I will deal with network interruptions and how the server will respond to ensure the smooth running of the website.

For the Backend of the project, most of it has been done, however, I plan on adding an SQL table to store the saved state of the server (save the messages sent over the website). Furthermore, I plan on creating multiple rooms to execute my solution of having different topic channels. Also, I will implement horizontal scaling to be able to support thousands of concurrent clients connected to the server. This aligns with my success criteria of having multiple users be able to connect to the server since there could be upwards of hundreds of children in a hospital.

Development (Networking)

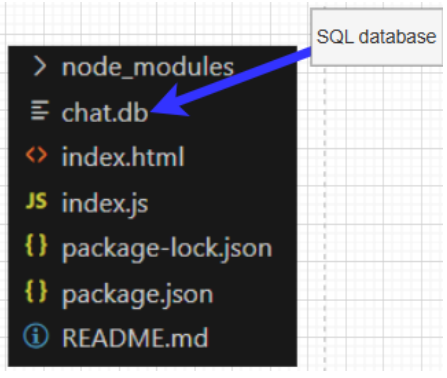
First, I import SQLite to my program, so that I can create and manipulate databases during the runtime of the program.

```
5   const sqlite3 = require('sqlite3'); // Importing SQlite which is a database engine (Database manager)
6   const { open } = require('sqlite'); // Importing the open function from SQLite to open the database
```

Next, I created a main subroutine to organise the main functionality of the server and allow for asynchronous events. Also, this creates one entry point into the main part of the program, which will improve maintainability and debugging.

```
8   // This function is the main part of our program, it has one entry point and is asynchronous
9   async function main() {
10
11      const db = await open({
12          filename: 'chat.db', // It will create a new database with this name is none is found
13          driver: sqlite3.Database // Tells SQLite which drives to use when interacting with the database
14      });
15      // This function will wait till the database chat.db is open
16
```

In this first part of the function, I open the database ‘chat.db’ on line 11 using the open() function from the SQLite library. If there is no database already in the project file, the function will create one. In addition, the function open() is asynchronous, therefore, we must wait for the function to complete. However, the keyword await means that the program will wait for the open() function to complete before running other parts of the function. This is because the main() function is asynchronous and will run other parts of the program while waiting, but this is not what we want since other parts of the program require the database. Therefore, by waiting for the database to load in, we prevent any runtime errors that may arise.



Here, it the “chat.db” SQL database, it´s stored in the project file and will store all the chat messages sent over the server. We can manipulate the data in the SQL database by using the SQLite function exec(). This allow for SQL commands to be executed from the program file.

```
    await db.exec(`
        CREATE TABLE IF NOT EXISTS messages (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            client_offset TEXT UNIQUE,
            content TEXT
        );
    `);
    // This function will execute the SQL commands inside the function
    /* The SQL will create the fields:
    id - the primary key that's an integer that will autoincrement
    client_offset - is a key that is unique for each client (used for searching messages for certain users)
    content - Here the message sent by the client is stored
    */
```

Here in this part of the function, the SQL commands will create the table messages if the table does not exist in the database (chat.db). If there is no message table, the SQL commands will create a message table with the following fields:

id – Primary key, and will autoincrement as records are added to the table

client offset – a key that is unique for each client (used for searching messages for certain users)

Content - Here, the message sent by the client is stored

The use of the database will come into view in later sections of the program but for now it will be vital in storing message; used for syncing messages of clients and recovering clients to the current state of the server if they disconnect.

```
    const app = express(); // Creates and instance of Express
    const server = createServer(app); // Creates a HTTP server
    // Express (app) will handel all incoming HTTP requests
    const io = new Server(server, {
        connectionStateRecovery: {} // Will save the state of the server if a user disconnects
    });
    // Creates an instance of socket.io

    // On request of the server, the server will responds with the HTML file (index.html)
    app.get('/', (req, res) => {''
        res.sendFile(join(__dirname, 'index.html'));
    })
```

In this part of the program, we create an instance of Express, which will handle the HTTP requests from clients. This will be pivotal in the handshake protocol used to send data between the server and clients. Next, we create an instance of a socket server, this object will listen with the socket 3000 for the client requests. Also, the attribute connectionStateRecovery will store the state of the server, it will be used to save the state of the server if a client disconnects. To help the client recover and sync back up with the state of the server.

```
// When there is a connection to the server this event will fire
io.on('connection', async (socket) => {
    // these functions are asynchronous since we are fetching from an API

    // This function will run once a 'chat message' is received by the server
    socket.on('chat message', async (msg) => {
        let result;
        try {
            // store the message in the database once a chat message is received
            result = await db.run('INSERT INTO messages (content) VALUES (?)', msg);
        } catch (e) {
            // should return an error
            return;
        }

        io.emit('chat message', msg, result.lastID);
        // Sends the chat message to all users and the id of the message
    })
```

Here in this part of the program, the function on() is asynchronous because in a later part of the function we will be inserting into the SQL database. Once the 'chat message is received by the server, the function will attempt to store the message into the database. It will return an error if the insertion is unsuccessful.

Next, the chat message is sent to all the clients connected to the server. The ID of the last message inserted into the database will also be sent over to the clients. This is because the clients will need to be updated on the current position of the server within the database. This means that if a client is not at the same position in the database as the server, then the client will need to be recovered to the current state of the server. Overall, this allows for the server to keep all the clients in-sync with all messages are sent and received. How the server updates the client if it's not in sync with the server will be covered in a later part of this document.

```
// This creates a reference of the server on the client side
const socket = io({
  auth: {
  serverOffset: 0
  }
});
```

Now, on the client side, we create a reference of the server. This object has the attribute auth, which contains the serverOffset. The significance of this attribute is to authenticate whether the server and the client are in sync.

```
socket.on('chat message', function(msg, serverOffset) { // the message and the state of the server sent from the server
    // Creates a list element that will contain the message
    const iteam = document.createElement('li');
    iteam.textContent = msg;
    message.appendChild(iteam);
    window.scrollTo(0, document.body.scrollHeight); // Moves down the pages to fit the message
    socket.auth.serverOffset = serverOffset;
    // This will update the state of the client to be synced with the state of the server
})
// Once the message is recieved, it will be displayed to the client
```

Continuing with the client side, once the event 'chat message' is received by the client, the variables msg and serverOffset are sent over by the server. What's important here to understand is that serverOffset is the current position of the server in real time. The serverOffset sent over by the server will be used to update the client's reference to the server. This is done on line 57.

```
64        // this event will run if the user reconnects to the server unsuccessfully
65        if (!socket.recovered) {
66            try {
67                // This will send all the messages to the client that they have missed
68                await db.each('SELECT id, content FROM messages WHERE id > ?',
69                    [socket.handshake.auth.serverOffset || 0],
70                    (_err, row) => {
71                      socket.emit('chat message', row.content, row.id);
72                    }
73                )
74            } catch (e) {
75                // Errors should be displayed
76            }
77        }
```

Moving back to the server side, if a client tries to reconnect to the server and is unsuccessful, this function (event) will run. The program will try to update the client on all the messages that have been sent from the time it disconnected. It does this by fetching the ID and content of each record in the message table if the ID is less than the current position of the client's serverOffset. This means it will check where the client is on the table messages (the position is stored in serverOffset on the client side; [.hanshake] allows the server to access this value on the client's side) and then will check the position of the server's serverOffset. Next, it will send the messages to the client that it missed until the client is up to date with the server (in-sync). If this process fails, it will output an error message.

This next section of code will deal with clients retries, which are the amount of time the client will retry to send a message to the server. In my program, the client will send a message to the server three times, and after each time, it will wait for an acknowledgement from the server. This acknowledgement is the call back function on line 57. This notifies the client that it does not need to keep retrying to send the message to the server because the server has received the message. This is because error 19 means that there is an SQL error, which means that the message that the server wants to put in the server has already been inserted. This is a great identifier of whether the message has been received by the server successfully or not. Finally, if the error does not run, then the message was not in the database and therefore not received by the server. In this case, the server will send the message out to all the clients and then acknowledge the client that it has now received the message.

```
// This function will run once a 'chat message' is received by the server
socket.on('chat message', async (msg, clientOffset, callback) => {
    let result;
    try {
      // store the message in the database once a chat message is received
      result = await db.run('INSERT INTO messages (content, client_offset) VALUES (?, ?)', msg, clientOffset);
    } catch (e) {
      if (e.error === 19) {
          callback(); // Will notifiy the client when there is a duplicate message
      } else {
          pass // nothing let the client retry
      }
      return;
    }

    io.emit('chat message', msg, result.lastID);
    // Sends the chat message to all users and the id of the message
    callback(); // acknowledge the event
})
```

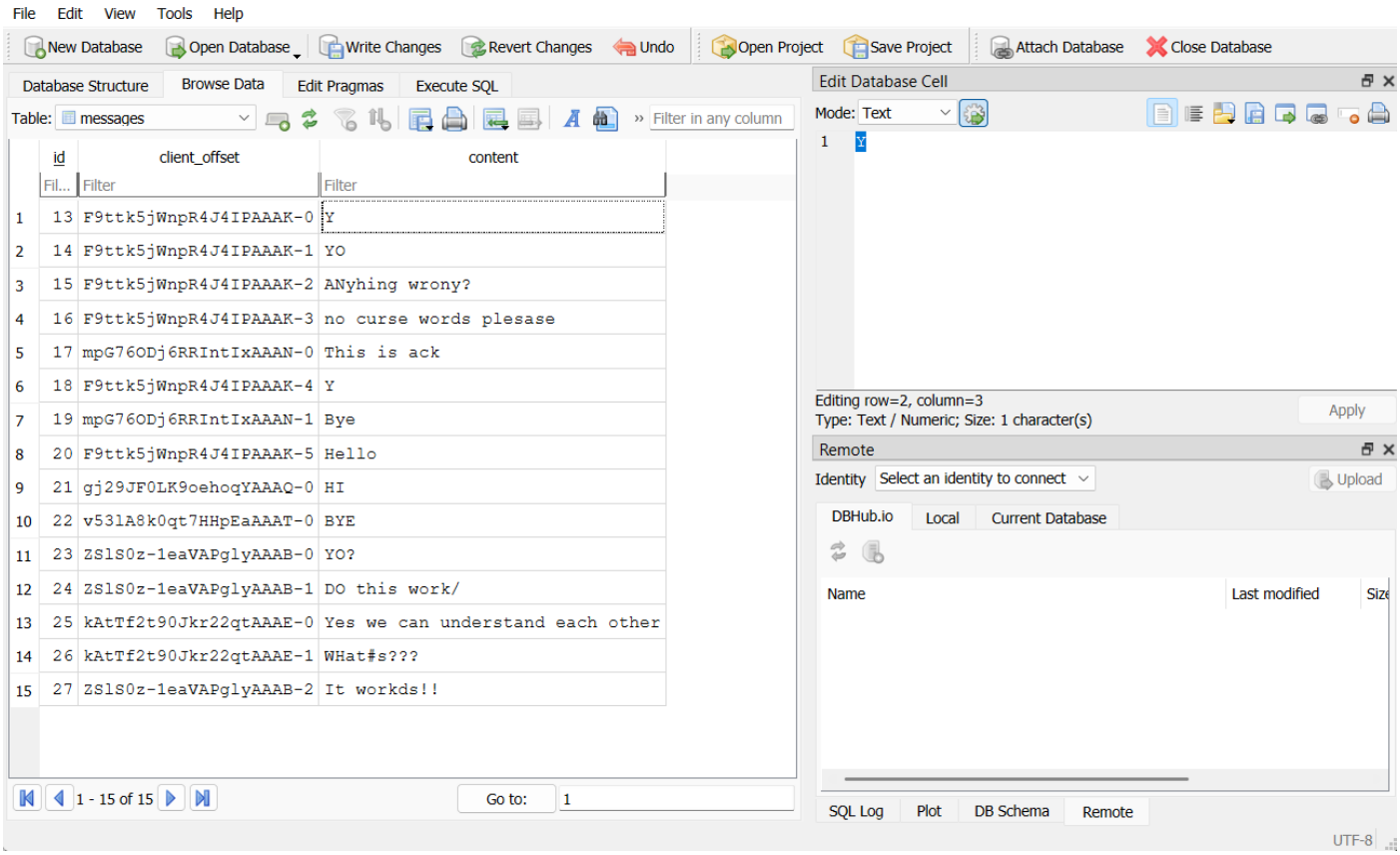
The code above is the server-side code that will handle client requests. However, on the client side the logic that enable retries is below. First, we define how long the client will wait for an acknowledgement (10000ms), then we define how many times the client will retry sending to the server (3 times).

```
let socket = io({
  auth: {
  serverOffset: 0
  },
  // How long the client will wait for an acknowlegdement from the server
  ackTimeout: 10000,
  // The amount of times the client will try to send messages to the server
  retries: 3,
});
```

The logic of this code below, is to send over a unique identifier to the server – clientOffset. This identifier will be used to store the messages of the clients in the chat database. The counter is added to the clientOffset and is incremented, this is so that a client can send multiple messages. Finally, the socket.id is the unique identifier of each client connect to the socket.

```
form.addEventListener('submit', (e) => {
    e.preventDefault();
    if (input.value) {
      // Creates an unique identifier for each message
      let clientOffset = `${socket.id}-${counter++}`;
      socket.emit('chat message', input.value, clientOffset);
      input.value = '';
      input.placeholder = 'Enter'
    }
    else {
      input.placeholder = 'Invalid input';
    }
})
```

Now, I will show the chat database, which stores the messages.



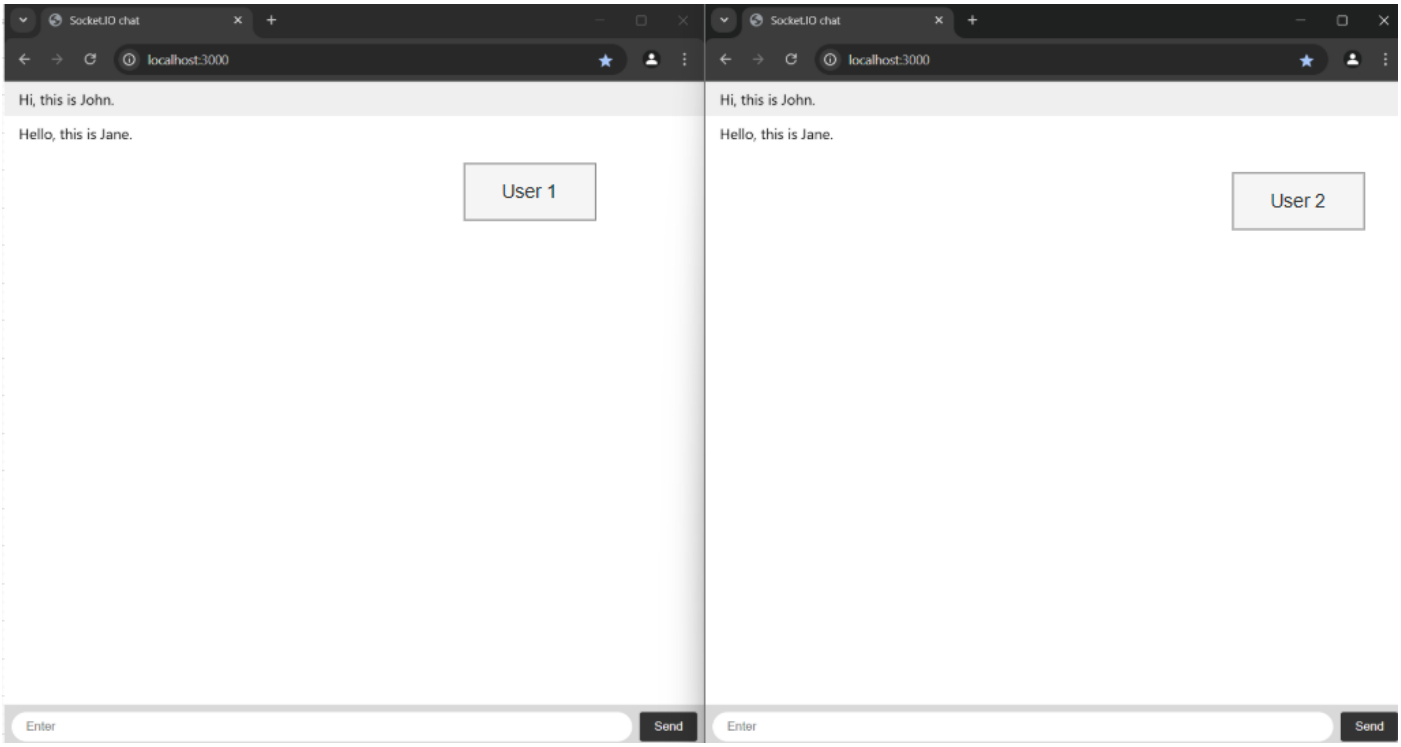
In this database browser, we can see that the client_offset field is unique, this is because we don't want duplicate messages. Furthermore, there are similar records in the client_offset field, this is because the characters before the dash is the unique id of each client. This is useful because we can search the field for messages sent be specific clients.

Iterative Tests (Networking)

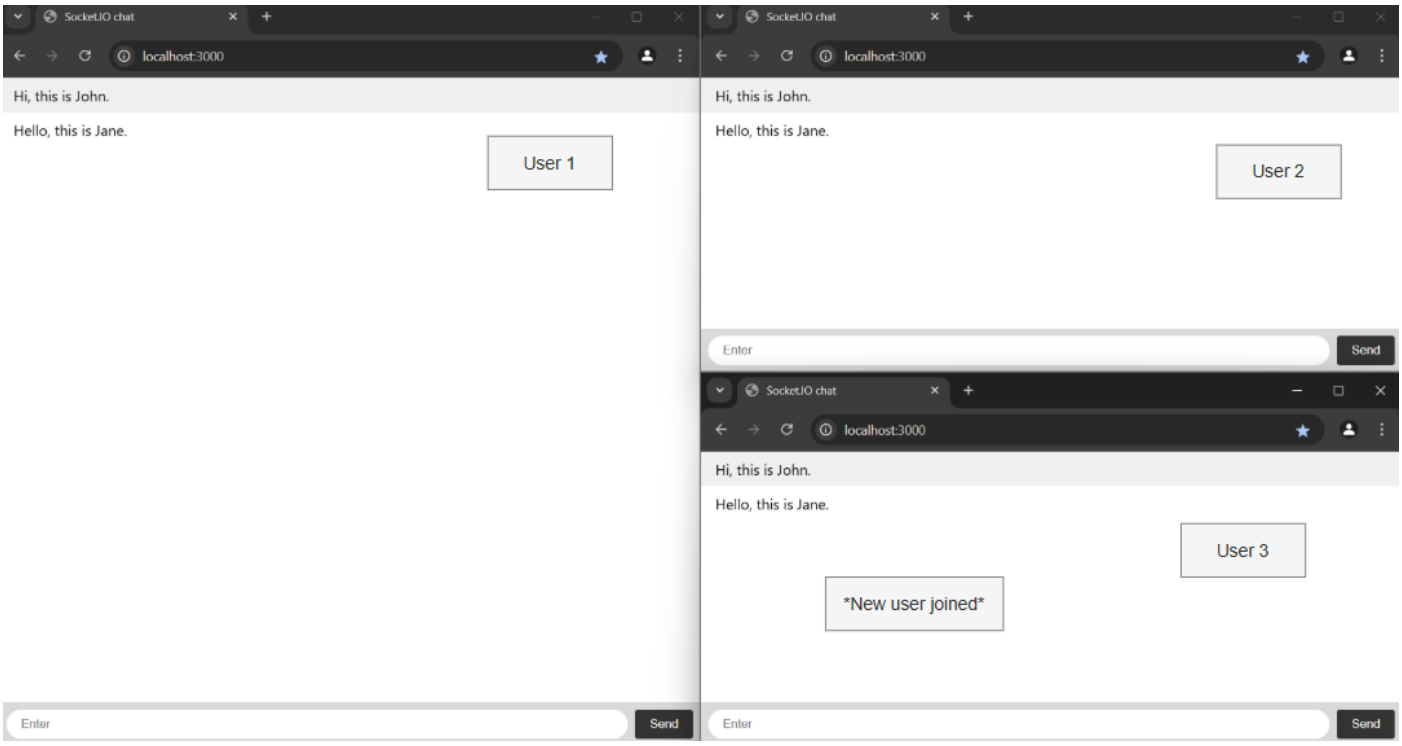
Test No.	Test Purpose	Test Data	Data type	Expected Outcome
6	Ensure recording of message history and chat logs	Users should be able to see previously sent messages on the topic boards. By entering messages: "Hi, this is John." "Hello, this is Jane." From two different accounts, then join on a new account.	Normal	The server should save previous messages even after the user leaves the website. Once the new user joins the topic channel, I should be able to see: "Hi, this is John." "Hello, this is Jane."

In this test I will connect a new user to the server, if the test is successful the new user should be able to see the previous messages sent by user 1 and user 2.

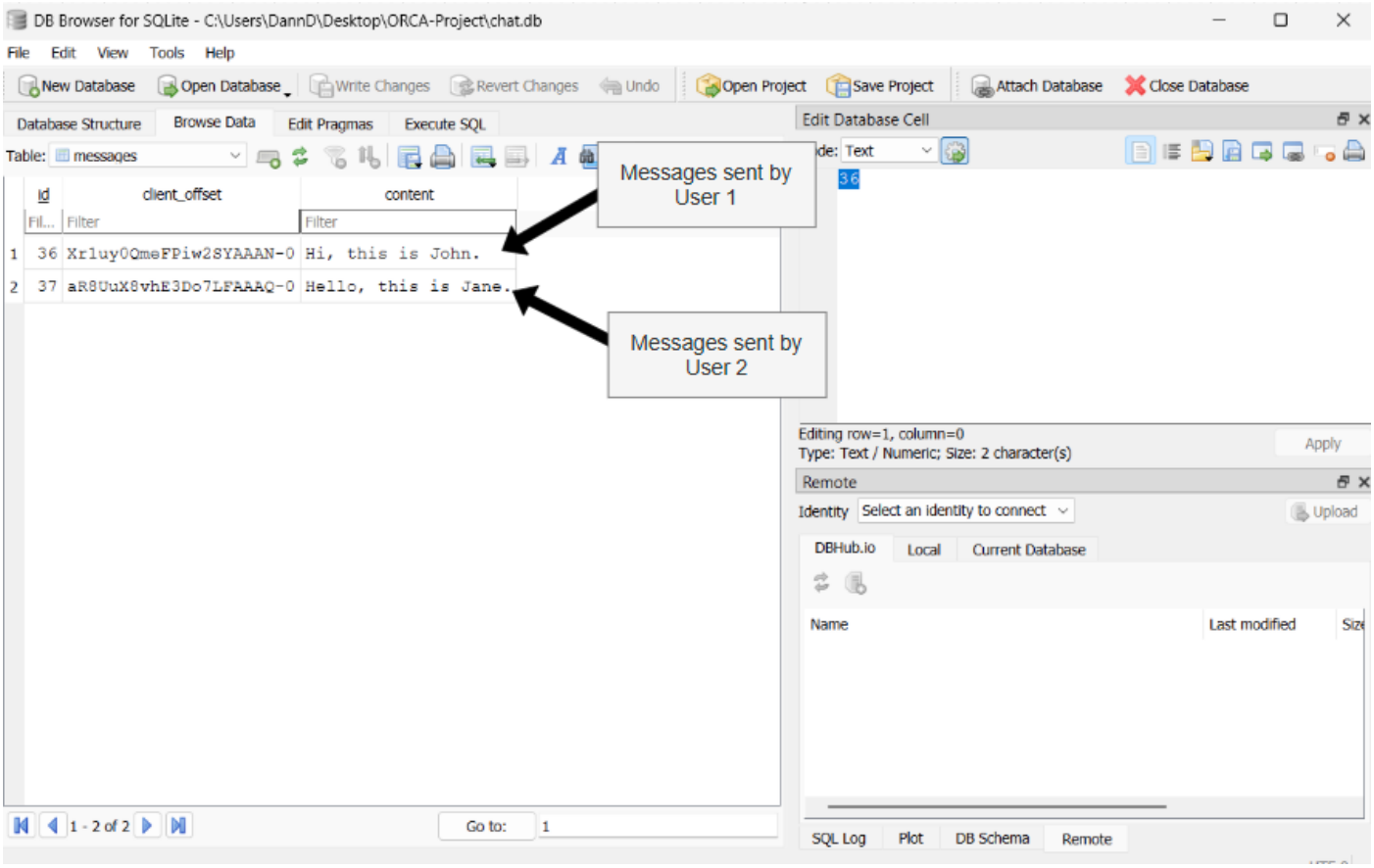
Before a new user joined



After a new user joined



Here, you can see that the test was a success since the new users that joined the server were able to see the previous messages sent by user 1 and user 2. Furthermore, this proves that messages are retrieved and stored into the chat database successfully, as if we take a look into the database, we should be able to see the two messages.

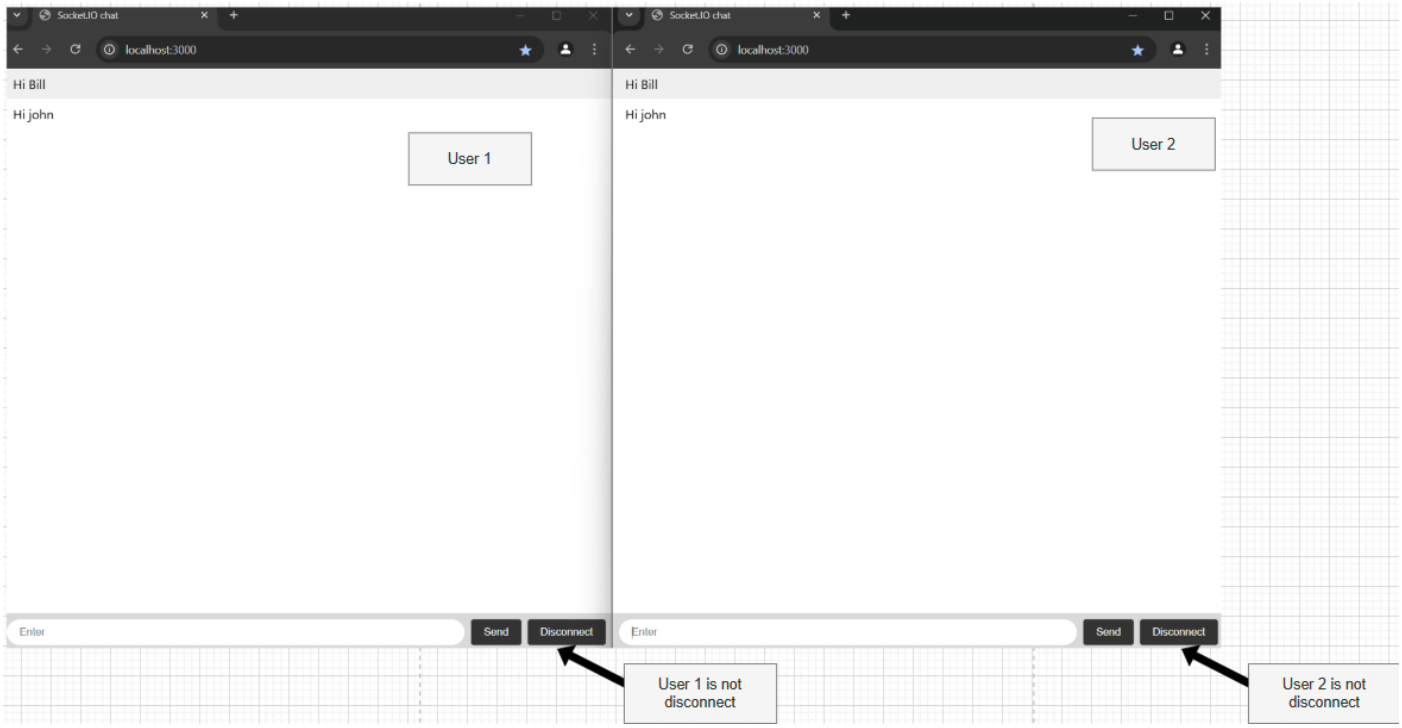


From this screenshot, I have proved that messages are stored and retried successfully.

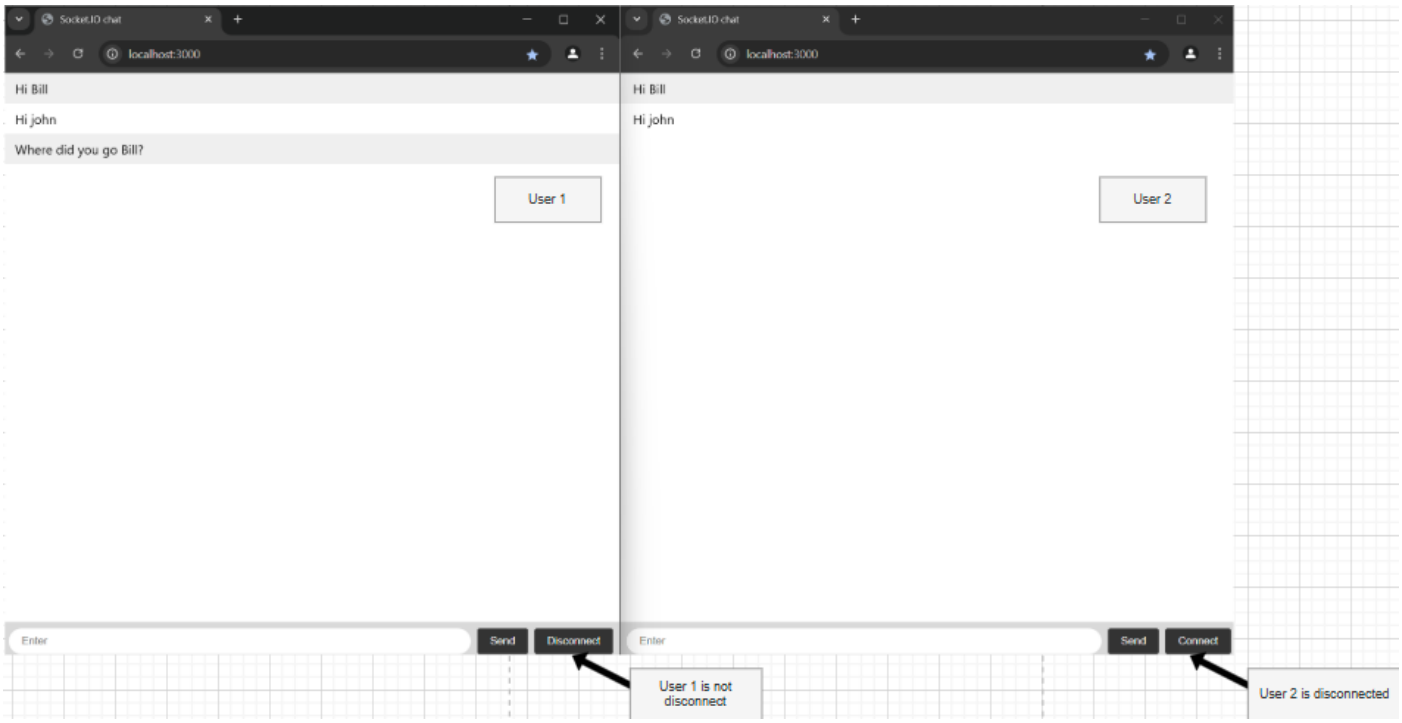
Test No.	Test Purpose	Test Data	Data type	Expected Outcome
7	Ensure that when a user connects back to the server, they are updates on any messages sent during disconnect.	User 1 will send - “Hi Bill” Then, user 2 will be sent “Hi John”. To ensure that the two users can receive messages before disconnecting user 2. Once user 2 is disconnect user 1 will send – “Where did you go John?”. Then user 2 will reconnect to the server.	Normal	The server should save previous messages even after the user disconnects from the website. Once the user 2 reconnects the topic channel, user 2 should have all the messages sent by user 1 while user 2 was disconnected displayed. – “Where did you go Bill?”

This test will ensure that users can reconnect to the server.

Before User 2 disconnects

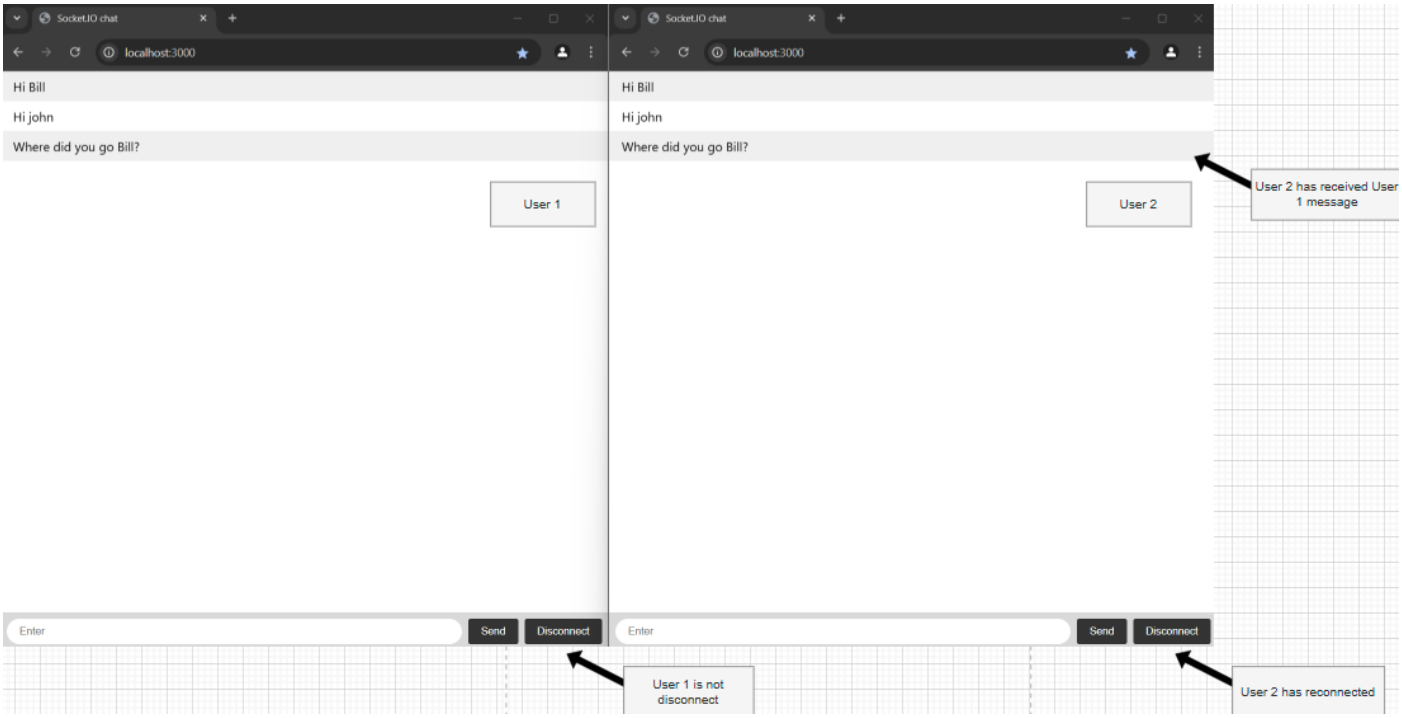


After User 2 disconnects



User 1 is still connected to the server here.

After User 2 reconnects



Once user 2 reconnects to the server, they are updated with all the messages sent by user 1, who was not disconnected from the server.

Development (Backend)

In this section, I first implemented horizontal scaling, which will allow hundreds of clients to connect to the server on different sockets.

```
7  const { availableParallelism } = require('node:os'); // This function will allow for parallel proccessing on the CPU's cores
8  const cluster = require('node:cluster'); // This module allows us to run multiple processes (workers) in parallel.
9  const { createAdapter, setupPrimary } = require('@socket.io/cluster-adapter');
10 // This will import a function the will be used to connect worker (servers) together
11
12 if (cluster.isPrimary) { // if this is the primary process (the first time this code has run)
13     const numCPUs = availableParallelism(); // Returns the number of cores of the CPU
14     // create one worker per available core
15     for (let i = 0; i < numCPUs; i++) {
16       cluster.fork({
17         PORT: 3000 + i
18       });
19     }
20
21     // set up the adapter on the primary thread (Connects all workers together)
22     return setupPrimary();
23   }
```

To implement horizontal scaling, I will need to import cluster and adapter functionality from node.js and socket.io, respectively – this is done on lines 7- 9. To create the framework that will handle the multiple sockets, I will need to create worker servers that will each handle https requests and messages sent by clients on a specific socket. In line 12, this process is started with a condition; this condition means that the block of code will run if the cluster is the primary cluster (The group of processes will manage the workers). Next, the function availableParallelism() will return the number of cores the host system has available. The for loop on line 15 will then create a new worker for each core available and assign the workers to different sockets. Finally, on line 22, the function setupPrimary() will connect all the workers; this means that all workers should be in sync with one another. In summary, this means that

a client can connect on sockets 3000 – 3005 (if the host server has 5 cores) and still be in sync with all other clients connected to different sockets.

Next, I implemented the topic channel system into the website – the functionality that will allow clients to communicate over multiple pages. I created this functionality by adding new events to the client and server side of the website.

```
91      socket.on('join channel', async (channel_ID, username, callback) => {
92
93          socket.join(channel_ID); // client joins the channel
94          if (!channels[channel_ID]) {
95              channels[channel_ID] = { name: `Room ${channel_ID}`, users: {} };
96              // If the channel doesn't exist then a new one is created
97          }
98          channels[channel_ID].users[socket.id] = username;
99          // client's username store in corresponding channel
100
101         // The username is sent to all the client in the channel - (updating the clients on who joined)
102         io.to(channel_ID).emit('user-joined', username, Object.keys(channels[channel_ID].users));
103         console.log(`${username} joined channel ${channel_ID}`);
104         callback(`Joined channel ${channel_ID}`);
105
106         // Fetch past messages for this channel
107         try {
108             const messages = await db.all('SELECT id, content FROM messages WHERE channel = ? ORDER BY id ASC', channel_ID);
109             messages.forEach((row) => {
110                 socket.emit('chat message', row.content, row.id);
111             });
112         } catch (e) {
113             console.error('Error fetching messages:', e);
114         }
115     });
```

First, on the server side, I added the 'join channel' event; this event will listen for when the client joins a channel. Once a client joins a channel, they will need to be updated with the past messages sent in the channel. Therefore, when the client joins a new channel, they will also be updated on past messages. Then, the client's unique socket ID is stored in their corresponding channel in the channels array. This will allow the server to know which server each client is in which server; therefore, the server knows to whom to send the messages.

To send the message to the corresponding channel, the server will use the .to() function from socket.io. This function allows specific clients to be sent messages if they are in a specified room (in my project, it's called a channel).

```
130             // Will send messages to all client in the corresponding channel
131             io.to(channel_ID).emit('chat message', msg, result.lastID);
132             callback(); // acknowledge the event
133         });
```

The messages are fetched from the chat.db. The database has been modified to include the channel_ID of each message sent. This will be useful when searching and querying, specifically when the server needs to fetch messages for a certain channel.

```
async function createDatabase() {
    const db = await open({
        filename: 'chat.db', // It will create a new database with this name is none is found
        driver: sqlite3.Database // Tells SQLite which drives to use when interacting with the database
    });
    // This function will wait till the database chat.db is open

    await db.exec(`
        CREATE TABLE IF NOT EXISTS messages (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            client_offset TEXT UNIQUE,
            content TEXT,
            channel TEXT
        );
    `);
    // This function will execute the SQL commands inside the function
    /* The SQL will create the fields:
    id - the primary key that's an integer that will autoincrement
    client_offset - is a key that is unique for each client (used for searching messages for certain users)
    content - Here the message sent by the client is stored
    channel - This is the channel that the message was sent in
    */

    return db;
}
```

This is a preview of the new chat.db.

DB Browser for SQLite - C:\Users\DannD\Desktop\ORCA-Project\chat.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Undo | Open Pro

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: messages | Filter in any column

	id	client_offset	content	channel
	Fil...	Filter	Filter	Filter
1	1	iVKUsW7fZ4n3BS_lAAAE-1740415620670	Hi	general
2	2	iVKUsW7fZ4n3BS_lAAAE-1740415626181	This workss!!!	general
3	3	RiXzLdy8IhsBqRboAAAH-1740415638548	My name is james	general
4	4	iVKUsW7fZ4n3BS_lAAAE-1740415649087	What's good jamies	general
5	5	RiXzLdy8IhsBqRboAAAH-1740415656382	YO	general
6	6	RiXzLdy8IhsBqRboAAAH-1740415671353	yyy	tech
7	7	iVKUsW7fZ4n3BS_lAAAE-1740415674373	>>>	tech

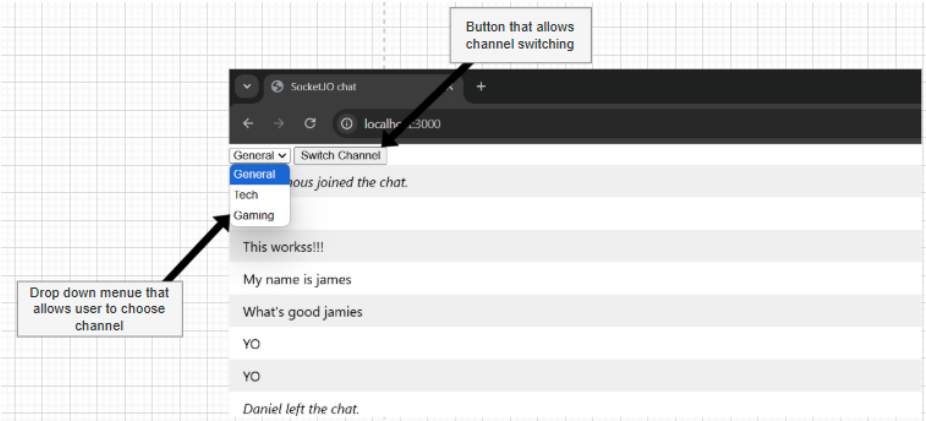
To complete the channel functionality, the client side must also be able to respond and listen for events from the server related to channel switching.

```
let channel_ID = "general"; // the default channel
let username = prompt("Enter your username:") || "Anonymous";
// This will ask the client for their name, if they enter nothing their name is set to Anonymous


// Join a channel
socket.emit('join channel', channel_ID, username, (response) => {
    console.log(response);
});
```

When a user joins the server, they are first prompted to choose a username. If they enter no username, their username will be set to anonymous. Once the user has a username, they join the default channel, which is 'general' – this will allow the user to be able to switch servers later without breaking the code.

```
    <select id="channelSelect">
      <option value="general">General</option>
      <option value="tech">Tech</option>
      <option value="gaming">Gaming</option>
    </select>
    <button onclick="switchChannel()">Switch Channel</button>
    <ul id="messages"></ul>
    <form id="form">
      <input id="input" autocomplete="off" placeholder="Enter" />
      <button>Send</button>
    </form>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/script.js"></script>

    <script>
      // When the client switches channels this event will run
      function switchChannel() {
        let newChannel = document.getElementById("channelSelect").value;
        socket.emit('join channel', newChannel, username, (response) => { // the client joins the new channel
        console.log(response);
        document.getElementById("messages").innerHTML = ""; // Clear messages when switching
      });
      channel_ID = newChannel;
      }

    </script>
  </body>
</html>
```

Here is the HTML, which will be what the user will see. Here, I've created a button that, when pressed, will switch the channel of the user.



Finally, when a user switches channels, they need to be disconnected from their previous channel. This is implemented by the client telling the server that it has left the channel (The block of code below).

```
// Once a client disconncets a message is sent to the clients
// Notifying them that someone has left
socket.on('user-left', (username) => {
    const item = document.createElement('li');
    item.textContent = `${username} left the chat.`;
    item.style.fontStyle = "italic";
    message.appendChild(item);
});

```

Also, the server must disconnect the client from the server physically and remove them from the channels array.

```
        // this will remove the client from the channel once the client disconnects
        socket.on('disconnect', () => {
            console.log('User disconnected');
            for (let channel_ID in channels) {
                if (channels[channel_ID].users[socket.id]) {
                    let username = channels[channel_ID].users[socket.id];
                    delete channels[channel_ID].users[socket.id];
                    io.to(channel_ID).emit('user-left', username);
                }
            }
        });
    });
```

Next, I started creating the authentication and login system by importing all of the necessary APIs. The first of which was ‘express-session’, allowing the server to hold in local memory the client’s credentials while the client is connected to the server. Essentially, it will be used to authenticate a user by using their cookies to hold in local memory a ‘token’ of verification. Next, the passport method will be the method used to authenticate the client. The bcrypt function is a hashing process which will be used to securely store the user’s password into a database. Finally, I load the database – userDatabase from the db.js file.

```
const authRoutes = require('./routes/auth'); // Imports the authenication routes
require('dotenv').config(); // Will store sensitive information outside the code securely
const session = require('express-session'); // Manages user sessions (keeps the users logged in)
const passport = require('passport'); // This will handle authentication
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcryptjs'); // Will hash our client's password
const userDatabase = require('./db'); // Loads the database (USERDATA)
```

The database is created in a different JavaScript file and then imported to the index.js file (our server).

```
JS db.js > ...
1   const Database = require('better-sqlite3');
2   const db = new Database('chat_auth.db');
3
4   // Create Users table if not exists
5   db.prepare(`
6     CREATE TABLE IF NOT EXISTS users (
7       id INTEGER PRIMARY KEY AUTOINCREMENT,
8       username TEXT UNIQUE NOT NULL,
9       password TEXT NOT NULL
10    )
11  `).run();
12
13  // exports database so that index.js can access it
14  module.exports = db;
15
```

To authenticate a client, I must first create a passport instance, which will run once the client tries to log in. In this function, I check the login details of the client with the data stored in the user database. This is because if the hashed password and username match a record in the database, then the client must be that user. In addition, I hash the password received by the server. This is to maintain user confidentiality and security as only the user knows their password. Overall, this function will authenticate the client as one of the users in the database if not an error message is returned.

```
122 async function checkLoginCredentials(database) {
123     // The passport will recieve the username and password from the client when the user tries to login
124     passport.use(new LocalStrategy(
125         { usernameField: 'username' }, // Use username instead of default username
126         (username, password, callback) => {
127             const user = database.prepare("SELECT * FROM users WHERE username = ?").get(username);
128             // Here it searches the database for the username to get the record
129
130             if (!user) { // If username is not in the database an error message is output
131                 callback(null, false, { message: "User not found" });
132                 return
133             }
134
135             // This hashes the password and compares it with the hashed password in the database
136             bcrypt.compare(password, user.password, (err, isMatch) => {
137                 if (err) {
138                     return callback(err);
139                 }
140                 if (!isMatch){ // If the hashes don't match then an error message is output
141                     return callback(null, false, { message: "Incorrect password" });
142                 }
143                 return callback(null, user);
144             });
145         }
146     ));
147 }
```

To finish client authentication, the server needs to keep the client authenticated, especially when the client is navigating to different pages on the website. This is done using the function serializeUser() to save the session ID to the user's browser's cookies. Then the function deserializeUser() is used to get the ID of the client, which will be used to get the full details of the user when the callback is called.

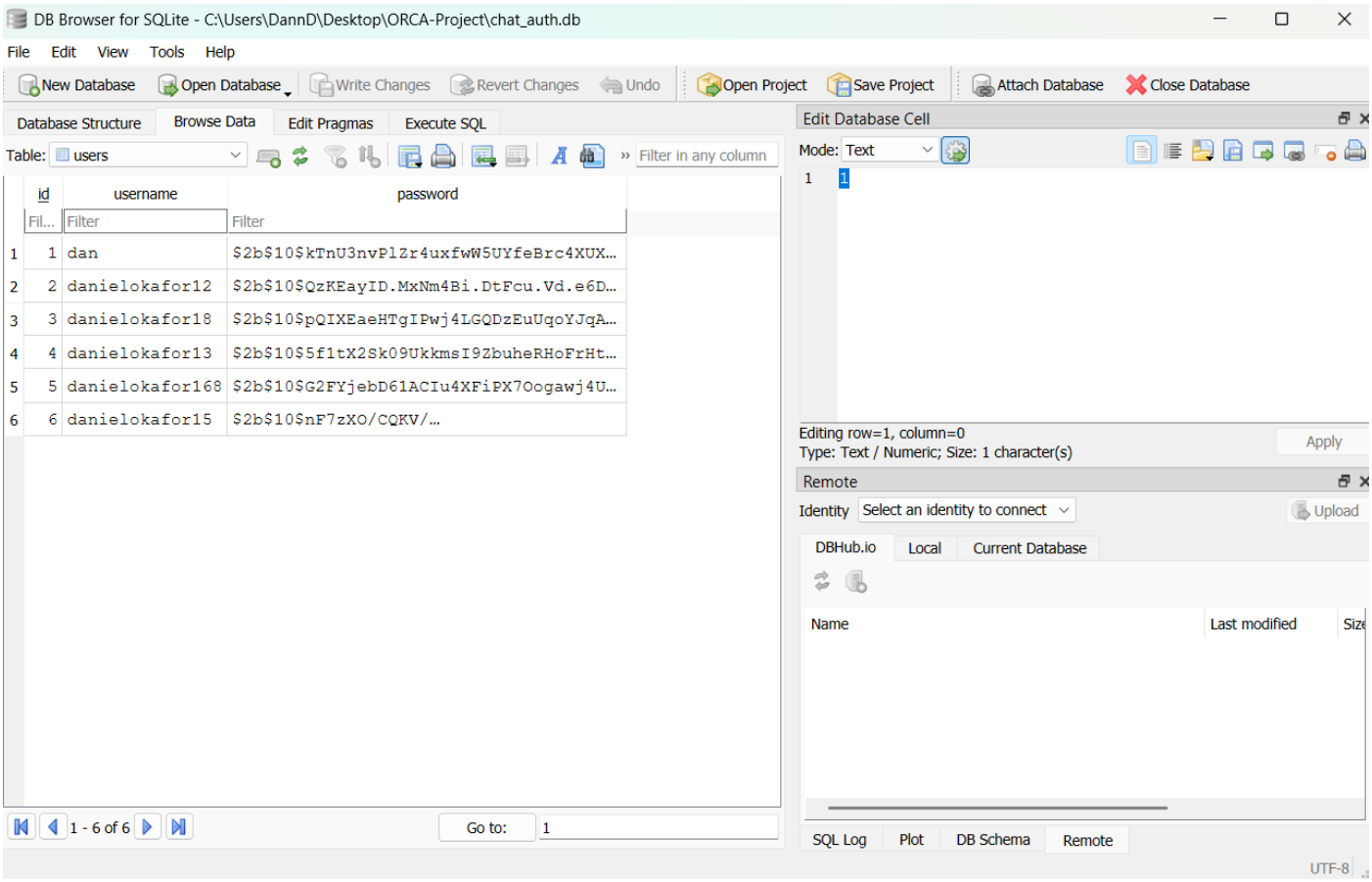
```
149  async function handleClientSession(database) {
150      // The sever will now remeber the use by saving their ID in the session
151      passport.serializeUser((user, callback) => callback(null, user.id));
152      // If they decide to change pages the ID is used to get their full details
153      passport.deserializeUser((id, callback) => {
154      const user = database.prepare("SELECT * FROM users WHERE id = ?").get(id);
155          callback(null, user);
156  });
157  }
158
```

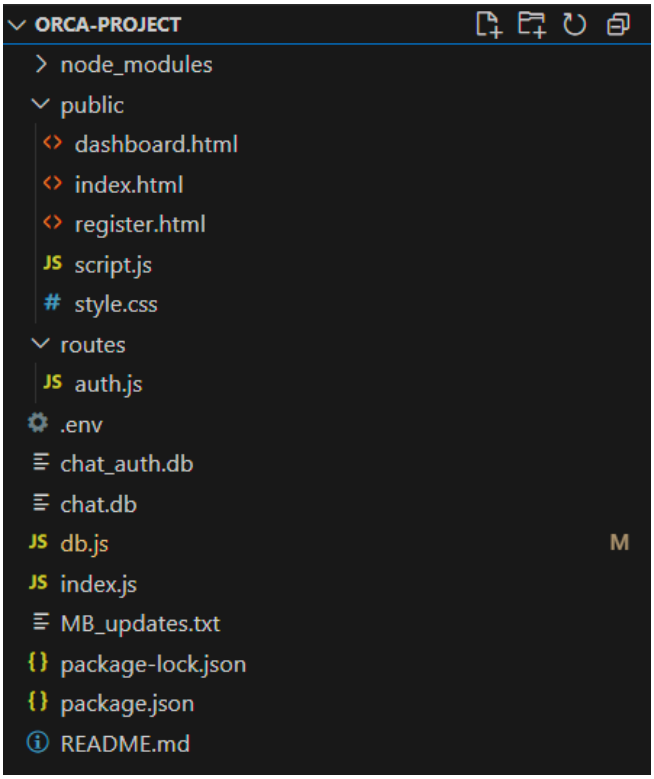
To finish off authentication, I used this function to authenticate users when they join the server.

```
159  async function authenticateClient(database) {
160
161      await checkLoginCredentials(database)
162
163      await handleClientSession(database);
164
165  }
```

```
171  // This function is the main part of our program, it has one entry point and is asynchronous
172  async function main() {
173
174      const { app, server, io } = setUpServer();
175
176      const messageDatabase = await createDatabase();
177
178      handleFileConnection(app);
179
180      await authenticateClient(userDatabase);
181
```

Here is the SQL database (userDatabase) used to store the sensitive user information:





Next, I need to create the login and register pages, but I also need to establish a secure connection between these pages to send sensitive data like the user's unhashed password. This secure communication will take place in the auth.js file. However, I first need to connect the files; I do this in the index.js file (the server).

In the index.js file, I have the function handleFileConnection (), which is used to link and configure all necessary files on the website. The part important to this current section of the project is the 'load routes' section of the function and the

configureSession() function, with the first establishment the secure routes (file paths) in which sensitive data can be sent over.

```
91  function handleFileConnection(app) {
92
93      // Serve static files (Give all public files to the client)
94      app.use(express.static(path.join(__dirname)));
95      app.use(express.static(path.join(__dirname, 'public')));
96
97      // On request of the server, the server will responds with the HTML file (index.html) as default
98      app.get('/', (req, res) => {
99          res.sendFile(join(__dirname, 'public', 'index.html'));
100     });
101
102     // Middleware
103     app.use(express.json());
104     // Allows the server to understand JASON data
105     app.use(express.urlencoded({ extended: false }));
106     // Allows the server to read HTML form data
107
108     configureSession(app);
109
110     // Load routes
111     app.use(authRoutes);
112     app.use('/auth', require('./routes/auth'));
113
114     // Middleware
115     app.use(express.urlencoded({ extended: false }));
116     // Allows the server to read HTML form data
117     app.use(express.json());
118     // Allows the server to understand JASON data
119 }
```

The configureSession() function is used to help keep the client authenticated. Here, I use a key to protect the data from being accessed by nonauthorised users. Next, I initialise the passport and link it to the session so that once a client is authenticated by the passport, the session ID can be stored in the cookies of the user's browser to keep them authenticated. In the final solution, the key will be changed to not compromise the security of the website.

```
75  function configureSession(app) {
76      // The session helps keep the user logged in
77      app.use(session({
78          secret: process.env.SECRET_KEY || 'your_secret_key', // secret key to protect data
79          resave: false,
80          saveUninitialized: false
81          // Prevents unnecessary session saving
82      }));
83
84      // Initialize Passport
85      app.use(passport.initialize());
86      // Start the node.js passport
87      app.use(passport.session());
88      // Links passport to session so logged in users are remembered
89  }
```

In the auth.js file, we create express routers, which will act as event handlers for pre-defined events.

```
const express = require('express');
const bcrypt = require('bcryptjs');
const passport = require('passport');
const db = require('../db');

const router = express.Router();
```

In this file, there are two specific events that it will wait for once the client calls it (In the case of express routers, the event happens when a client requests a specific subdomain). The first is when the client wants to log into the server. As you can see, once the /login subdomain is requested, this router function will run. The first operation of the function is to authenticate the user by using the passport.authenticate() function to check if the credentials sent by the client are valid. If the credentials are valid, then the client is redirected to the dashboard. The dashboard is now where chatting takes place (as shown in the previous part of this project – real-time chat functionality).

```
// Login Route
router.post('/login', (req, res, next) => {
    passport.authenticate('local', (err, user, info) => {
        if (err) {
            console.error("Authentication error:", err.message);
            return res.status(500).json({ message: "Internal server error" });
        }
        if (!user) {
            return res.status(400).json({ message: "Invalid username or password." });
        }

        req.logIn(user, (err) => {
            if (err) {
                console.error("Login error:", err.message);
                return res.status(500).json({ message: "Internal server error" });
            }
            res.redirect("/dashboard.html");
            return
        });
    })(req, res, next);
});
```

Now, on the client side, the index.html file is the file that is calling this event.

```
    <script>
        document.getElementById("login-form").addEventListener("submit", async (e) => {
            e.preventDefault(); // Prevent default form submission

            // Get username and password input values
            const username = document.getElementById("username").value;
            const password = document.getElementById("password").value;

            try {
                const response = await fetch("/auth/login", {
                    method: "POST",
                    headers: { "Content-Type": "application/json" },
                    body: JSON.stringify({ username, password })
                });

                if (!response.ok) {
                    const data = await response.json();
                    showErrorMessage(data.message); // Display error message from server
                    return;
                }

                // Redirect user to dashboard on successful login
                window.location.href = "/dashboard.html";
            } catch (error) {
                console.error("Error during login:", error.message);
                showErrorMessage("An error occurred. Please try again.");
            }
        });

```

The username and password are retrieved from the login form once the form is submitted. Then, the file will request the /auth/login subdomain, which the server-side logic (auth.js) is listening to. The server-side logic will then return a message, which will be displayed for the user to see when the login is unsuccessful. Otherwise, the file will redirect the client, assuming that the login was successful.

Finally, here is the logic for the error handling.

```
        function showErrorMessage(message) {
            const errorContainer = document.getElementById("error-message");
            errorContainer.textContent = message;
            errorContainer.style.display = "block"; // Show error message
        }
    </script>

</body>
</html>
```

Which then edits the division shown below.

```
7       <h2>Login</h2>
8       <form id="login-form">
9           <input id="username" name="username" placeholder="username" required />
10          <input id="password" type="password" name="password" placeholder="Password" required />
11          <button type="submit">Login</button>
12      </form>
13      <a href="register.html">Register</a>
14
15      <div id="error-message" style="display: none; color: red;"></div>
```

The next important section of the backend is the registration; it follows the same structure as the login section. First, the user is taken to the registration page via a link on the login page. Once they are on the registration page, the user will be able to create an account by filling out the input boxes. Once they fill out the inputs and register, the client will request the express route /auth/register, which will run the code below.

```
25  // Register Route
26  router.post('/register', async (req, res) => {
27      try {
28          const { username, password } = req.body;
29
30          // Hash password and insert user
31          const hashedPassword = await bcrypt.hash(password, 10);
32          db.prepare("INSERT INTO users (username, password) VALUES (?, ?)").run(username, hashedPassword);
33
34          return res.status(200).json({ message: "User registered successfully!" });
35      } catch (err) {
36          console.error("Error during registration:", err.message);
37          return res.status(500).json({ message: "Server error. Please try again later." });
38      }
39  });
40
```

This program will receive the password and the username of the user, it will hash the password for security and then insert the hashed password into the user database. Finally, the program returns a message; this could be an error, like on line 37, or it can be an acknowledgement of success as done on line 34. The program sends the message to the user by using http status responses: 200 being a good request – successful registration, 500 being a bad request – an error has occurred. The client knows what to do given each request due to the logic on the client side to handle the https request.

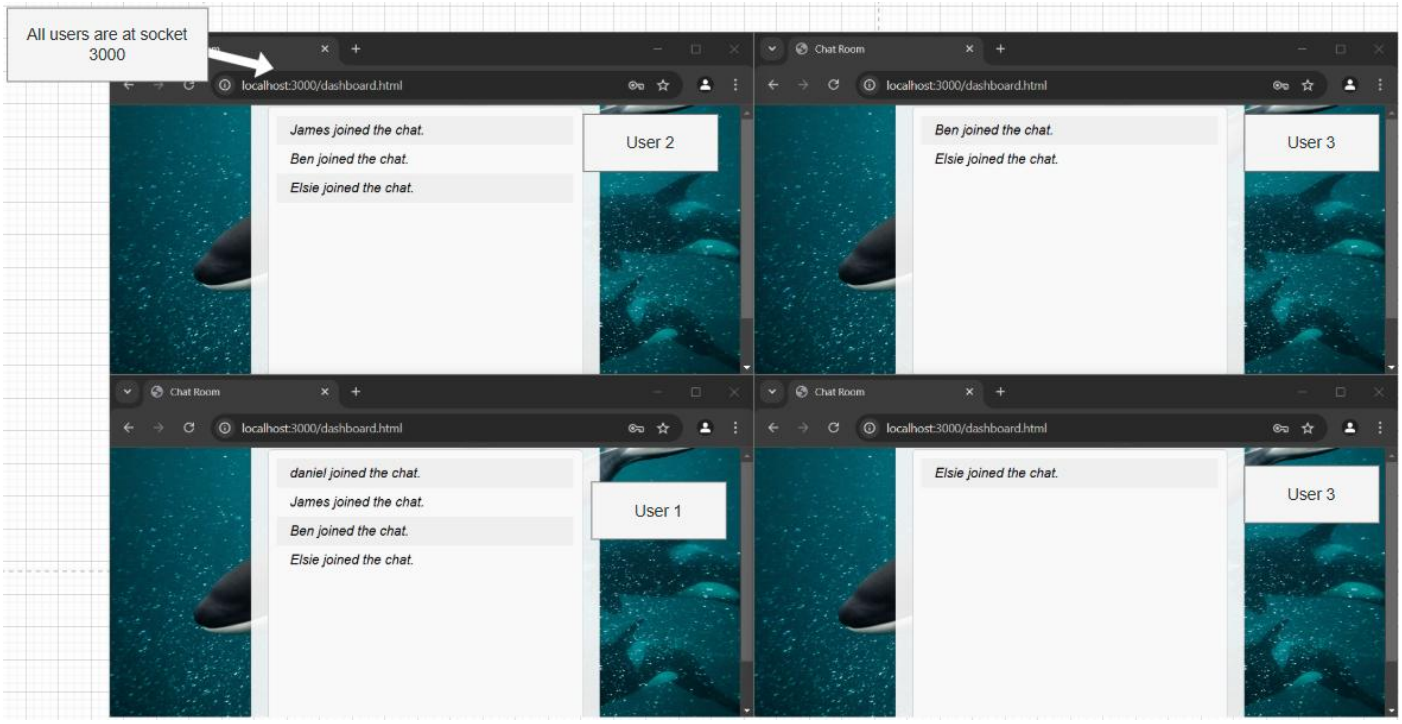
```
41      <script>
42          const form = document.getElementById("register-form");
43
44          form.addEventListener("submit", async (e) => {
45              e.preventDefault(); // Prevent default form submission
46
47              const username = document.getElementById("username").value.trim();
48              const password = document.getElementById("password").value.trim();
49              let confirmPassword = document.getElementById("confirm-password").value;
50
51              // Password match validation
52              if (password !== confirmPassword) {
53                  showErrorMessage("Passwords do not match.");
54                  return;
55              }
56
57              try {
58                  const response = await fetch("/register", {
59                      method: "POST",
60                      headers: { "Content-Type": "application/json" },
61                      body: JSON.stringify({ username, password })
62                  });
63
64                  const data = await response.json();
65
66                  if (!response.ok) {
67                      showErrorMessage(data.message); // Show error message from server
68                  } else {
69                      window.location.href = "/index.html"; // Redirect on success
70                  }
71              } catch (error) {
72                  showErrorMessage("An error occurred. Please try again.");
73              }
74          });
```

Here is the client-side logic. Once the payload of the username and password is sent to the server, the client will receive a http response. Depending on whether the response is 'ok' or not, it will display the error message which was sent over by the server. Finally, if the response is 'ok', the client is then redirected to the login page (index.html) where they can now log into the website having registered.

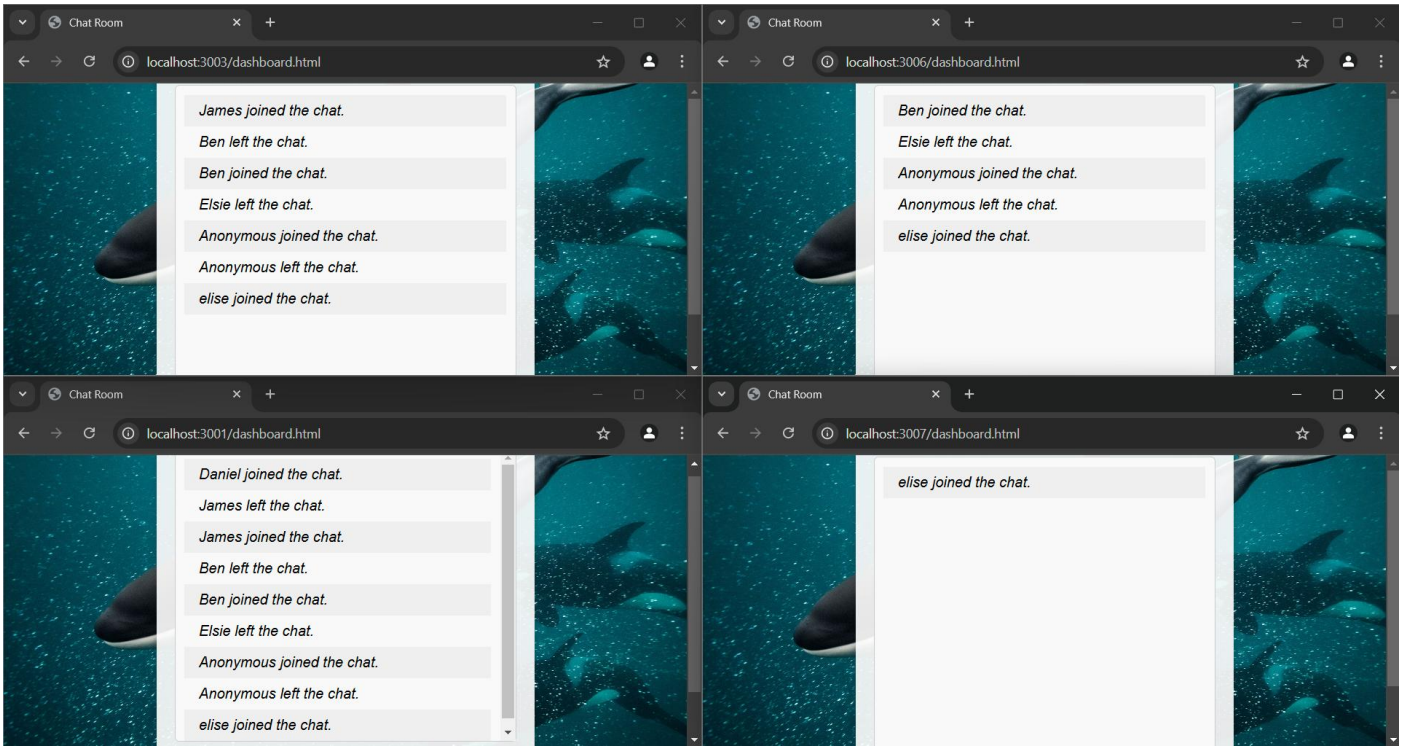
Iterative Tests (Backend)

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
8	Ensured that the user can join the server on different sockets	I will create four instances of the website using four different accounts. I will join the server of four different sockets. localhost:3001, localhost:3003, localhost:3006 localhost:3007	Normal	The server should be available on multiple sockets.

Before switching sockets



After switching sockets



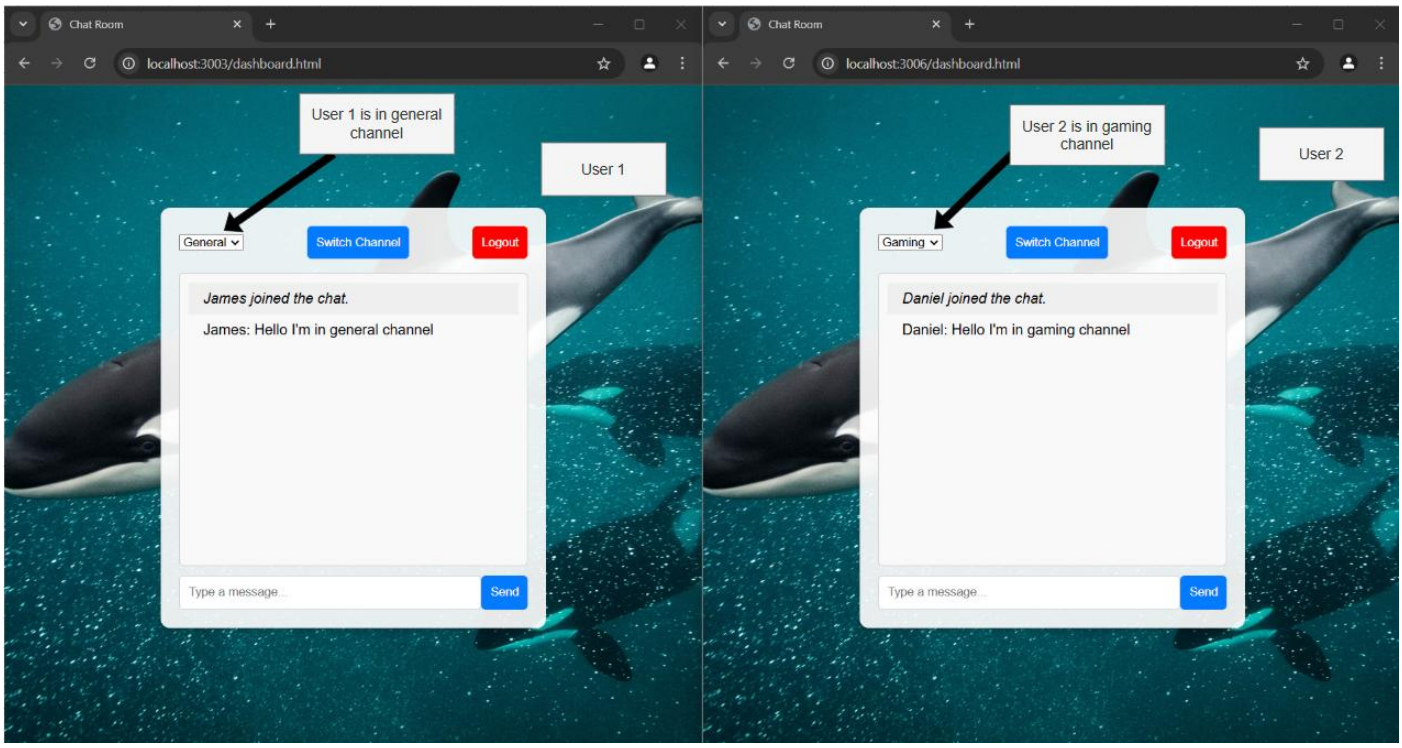
As you can see, all the users left and rejoined the new sockets. The users were still able to access the server even at different sockets. The number of sockets available is based on the number of cores available on the server. In this test, sockets 3000 – 30007 were available and had servers running on them. Furthermore, all servers are synced with one

another as shown above – messages are still received by each client.

```
PROBLEMS  OUTPUT  COMMENTS  TERMINAL  DEBUG CONSOLE  PORTS

PS C:\Users\DannD\Desktop\ORCA-Project> node index
Server running at http://localhost:3003
Server running at http://localhost:3004
Server running at http://localhost:3007
Server running at http://localhost:3000
Server running at http://localhost:3001
Server running at http://localhost:3002
Server running at http://localhost:3005
Server running at http://localhost:3006
```

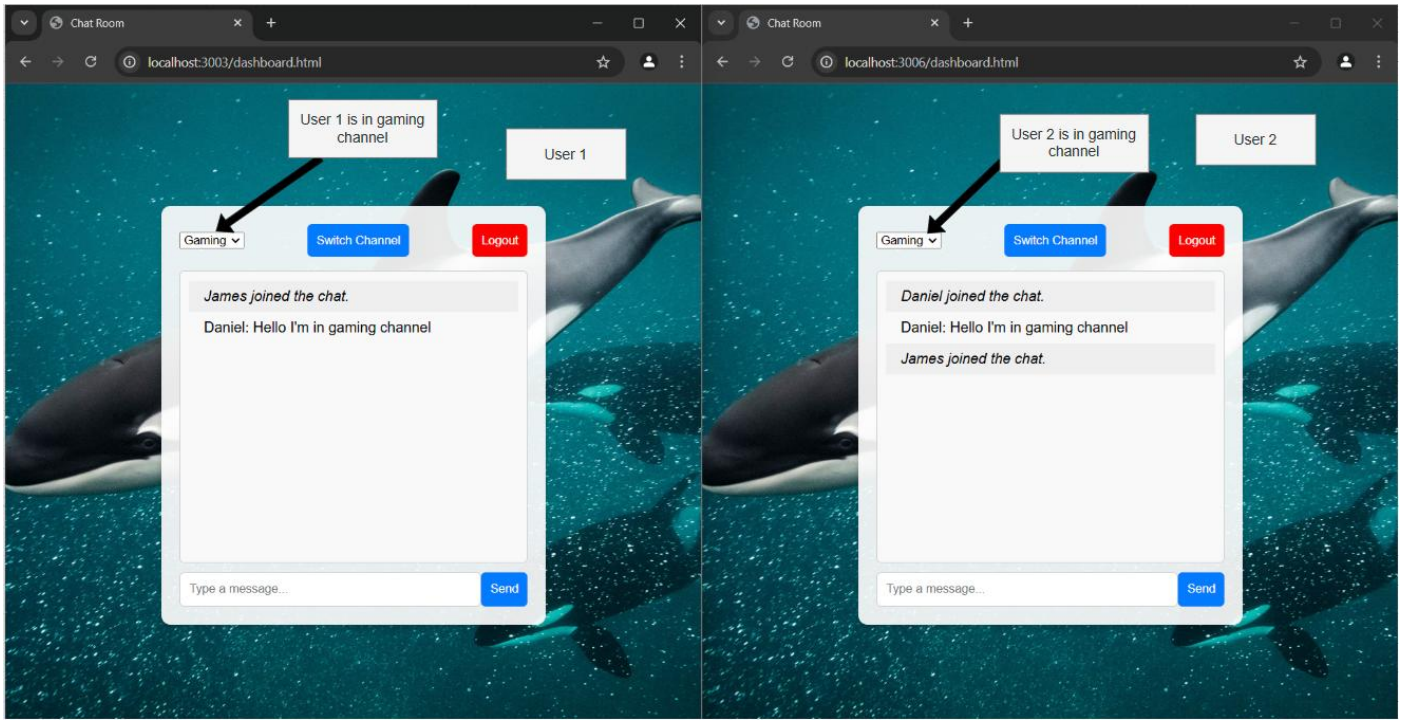
Test No.	Test Purpose	Test Data	Data type	Expected Outcome
9	Ensure that users can switch channels.	Using two users I will have then in two different channels; I will have one send a message in their channel. Then, I will show the of user joining their channel.	Normal	The users should be able to send message on multiple channels, the messages sent of different channels should only be received by those in that channel. However, once a user joins the channel they should receive all the past messages sent.

After sending messages



As you can see above, user 1 has sent a message in the general channel and user 2 has sent a message in gaming channel. Both the user can not see each other's message because they are in different channels.

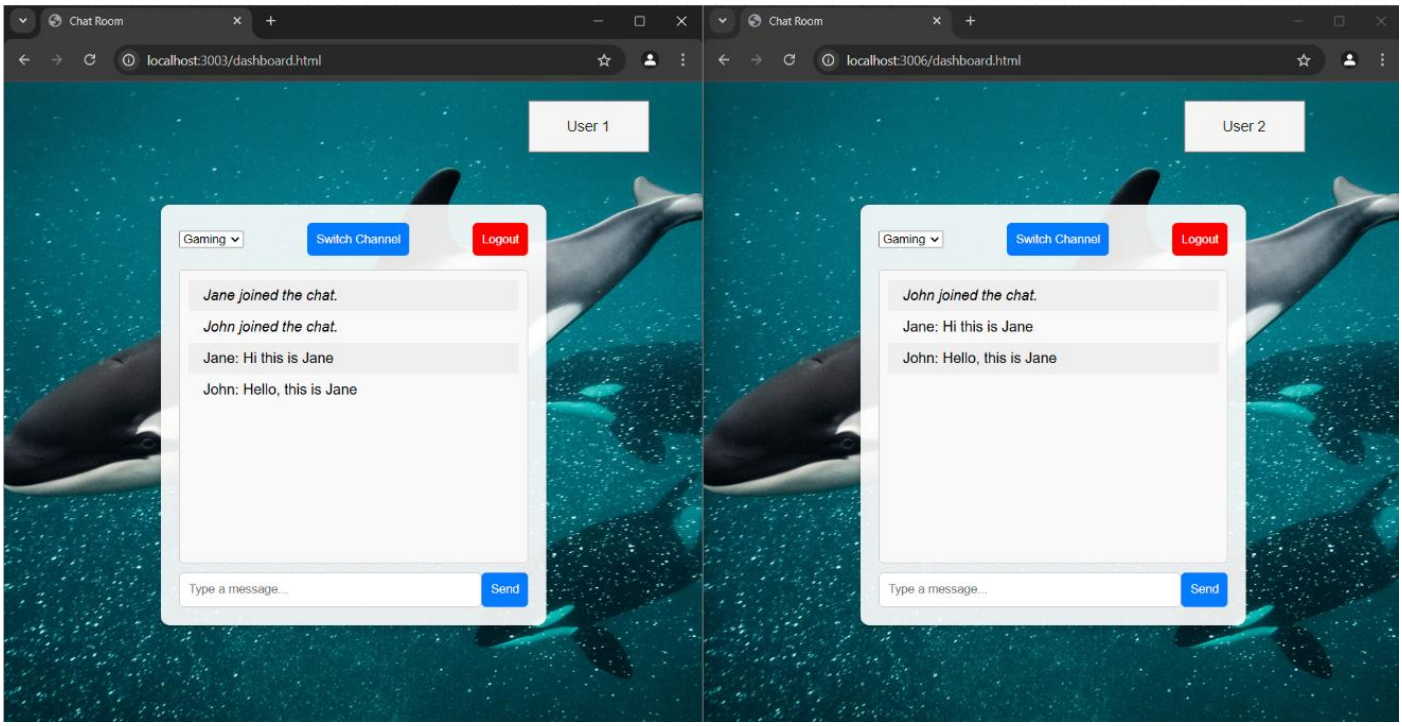
After switching channels



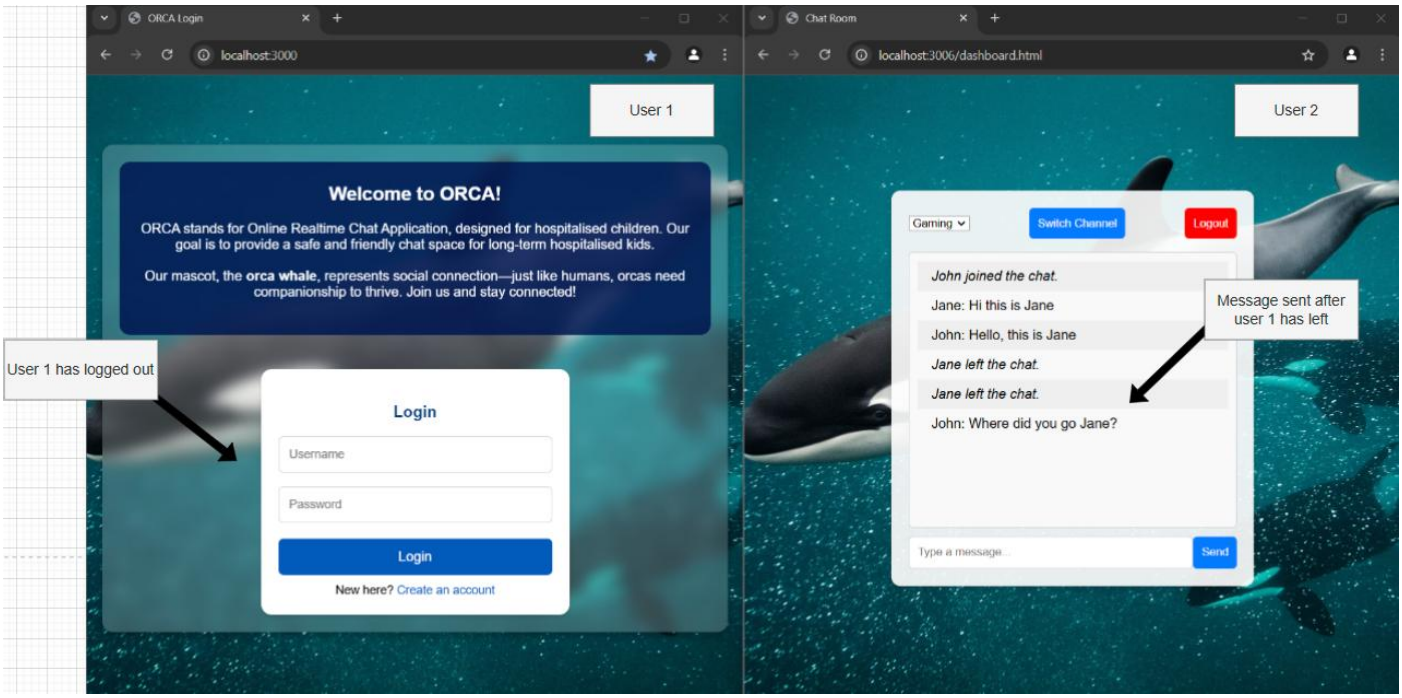
In this image here, we can see that user 1 (James) has joined the gaming channel and now he can see user's 2 (Daniel) message.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
10	Ensure recording of message history and chat logs	Users should be able to see previously sent messages on the topic boards. By entering messages: "Hi, this is John." "Hello, this is Jane." From two different accounts, then log out of one of them so we can test this feature.	Normal	The server should save previous messages even after the user leaves the website. Once the user returns to the topic channel, I should be able to see: "Hi, this is John." "Hello, this is Jane."

Before user 1 logout

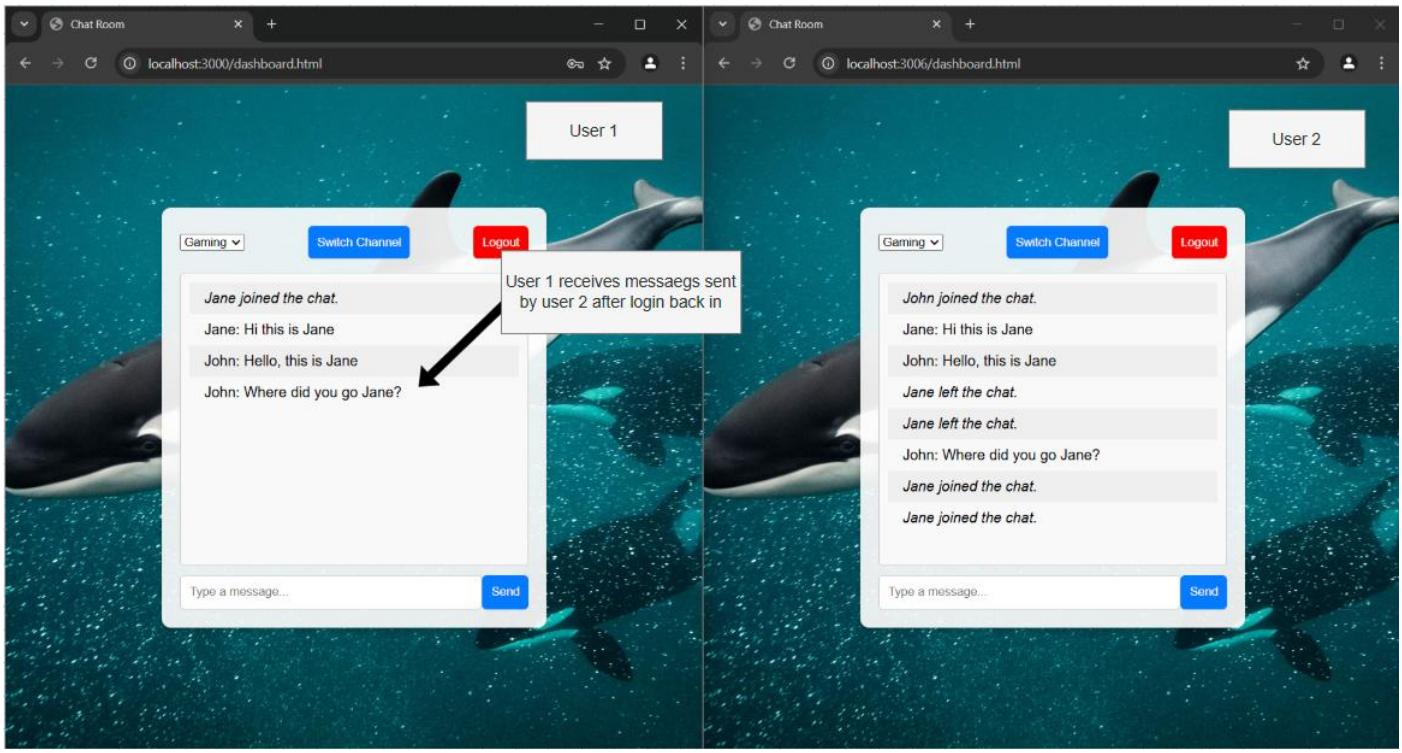


After user 1 logout



Here after logging out, user 2 send the message ‘Where did you go Jane?’. This will be useful as a reference for when user 1 rejoins the server, telling us whether messages have been loaded correctly after a user logs out.

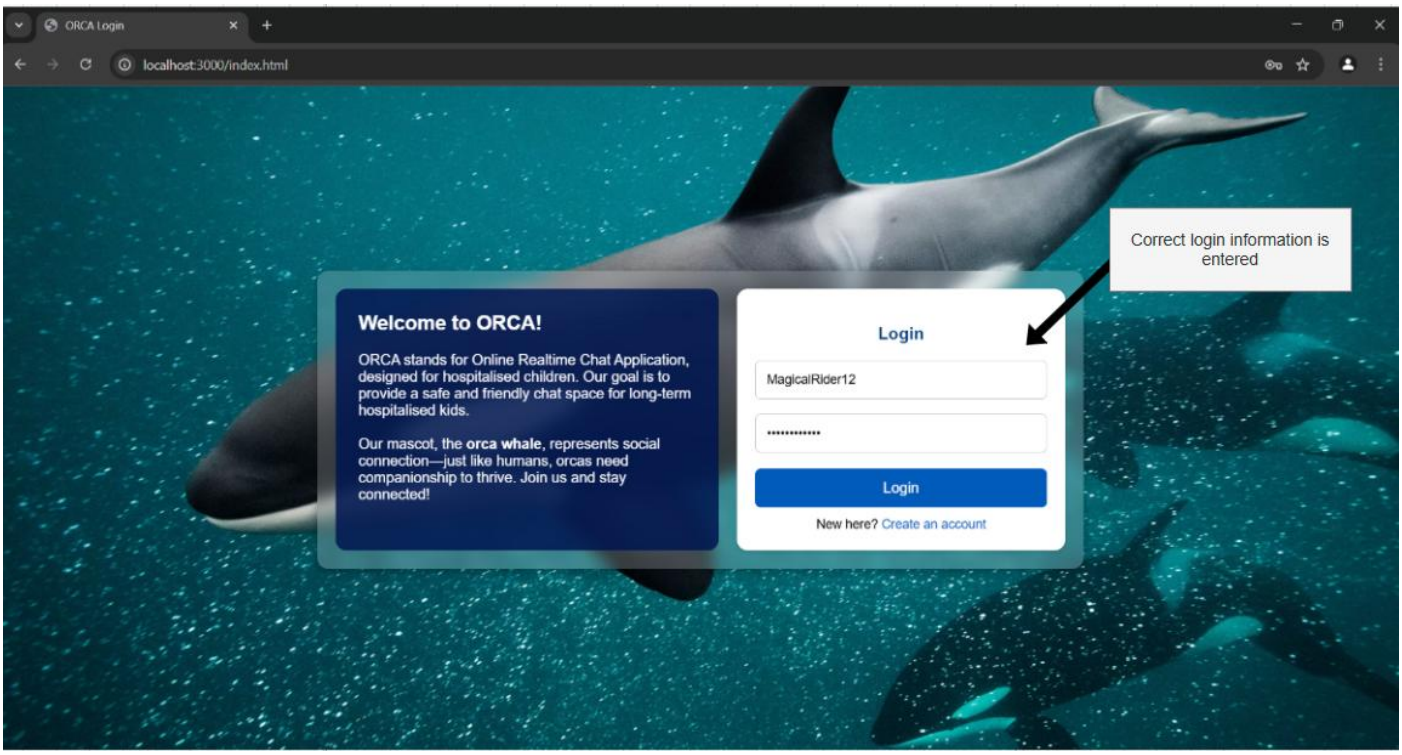
After user 1 rejoins



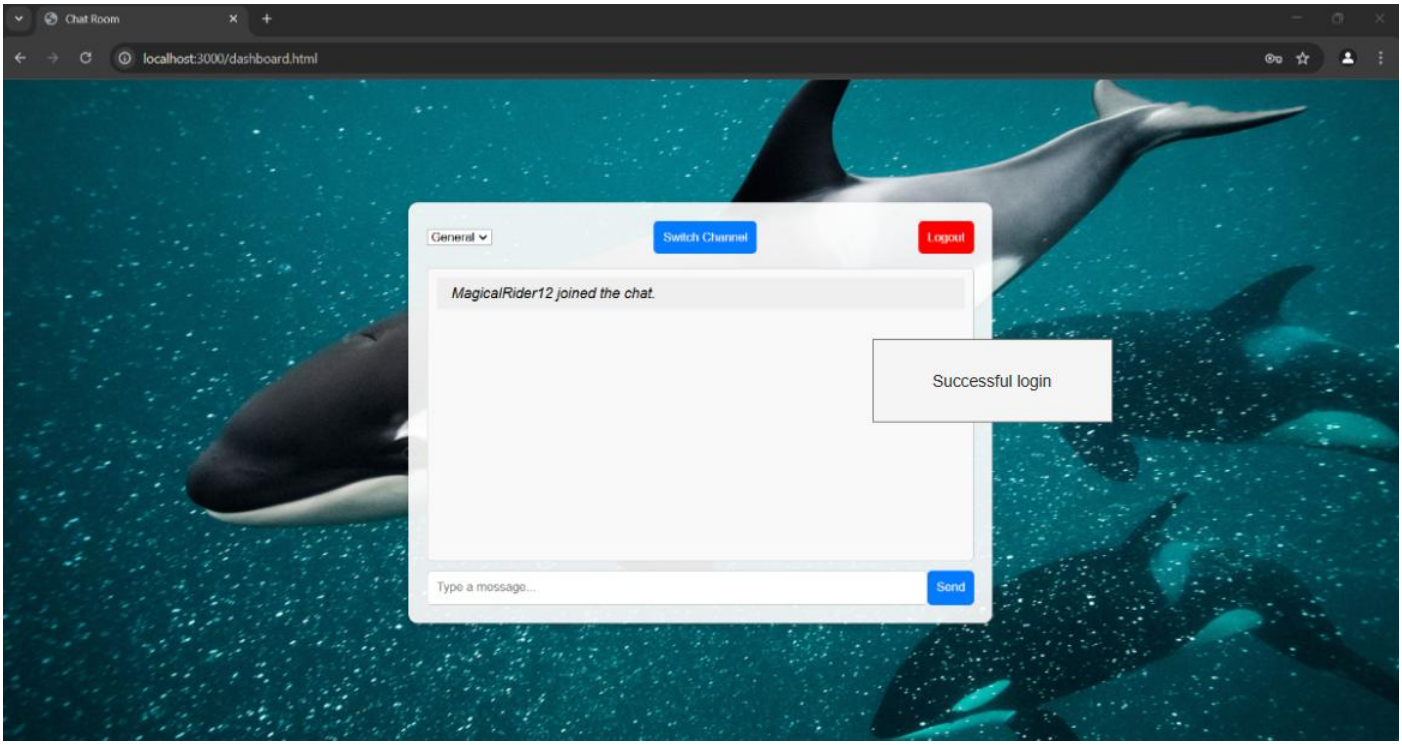
Once the user rejoins the sever, they are updated on the messages sent where they were not login. This proved that the test was a success because user 1 received our reference message.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
11	Ensure that users can log in to the website with a valid account.	I will give the user a valid account Username: MagicalRider12 Password: Password1234	Normal	The user should be able to access the chat panel after entering the correct credentials.

Before logging

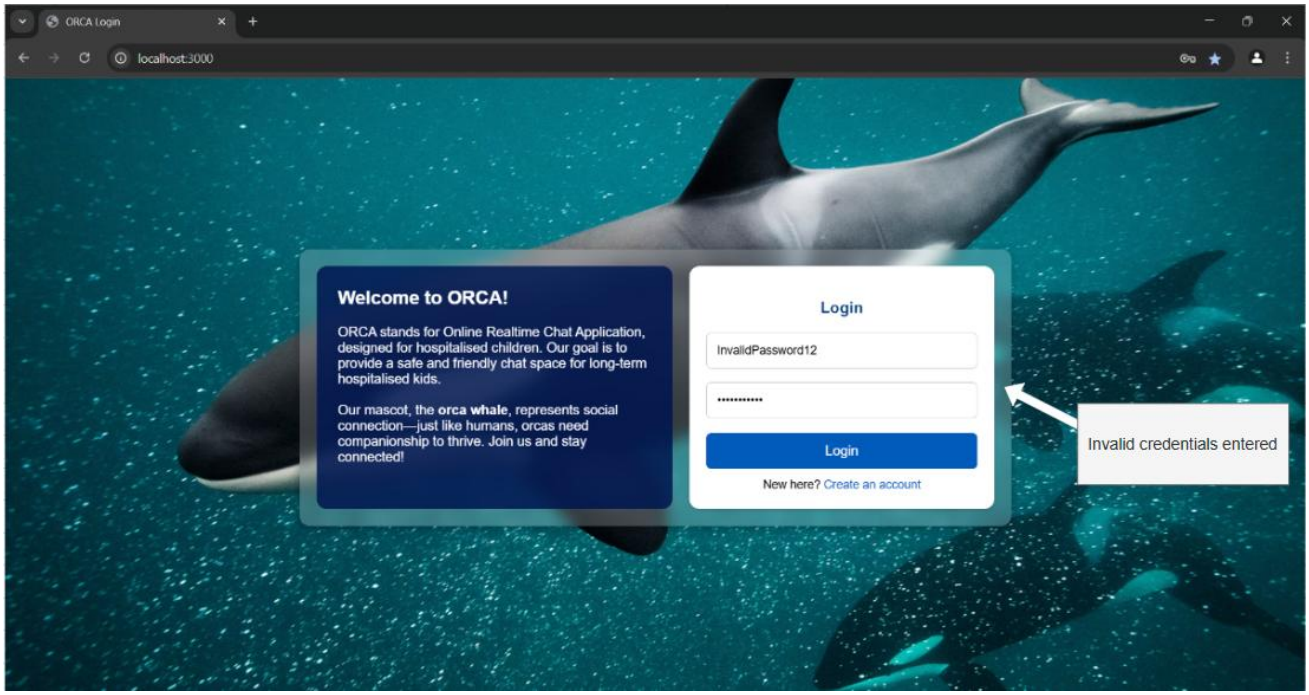


After logging

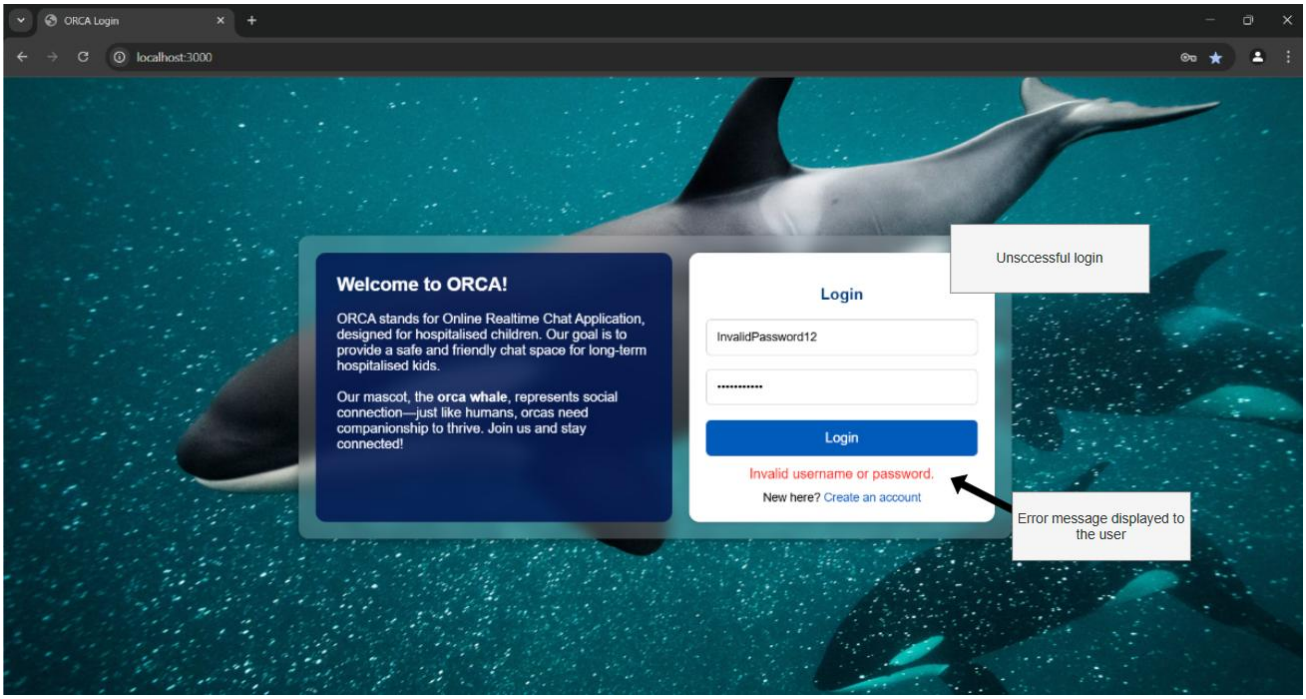


Test No.	Test Purpose	Test Data	Data type	Expected Outcome
12	Ensure that the user cannot log in with invalid account information.	I will give the user an invalid account Username: InvalidAccount12 Password: Password1234	Invalid	Invalid credentials should warn the user with an error message. Also, the user should not be redirected to the chat panel.

Before entering invalid credentials



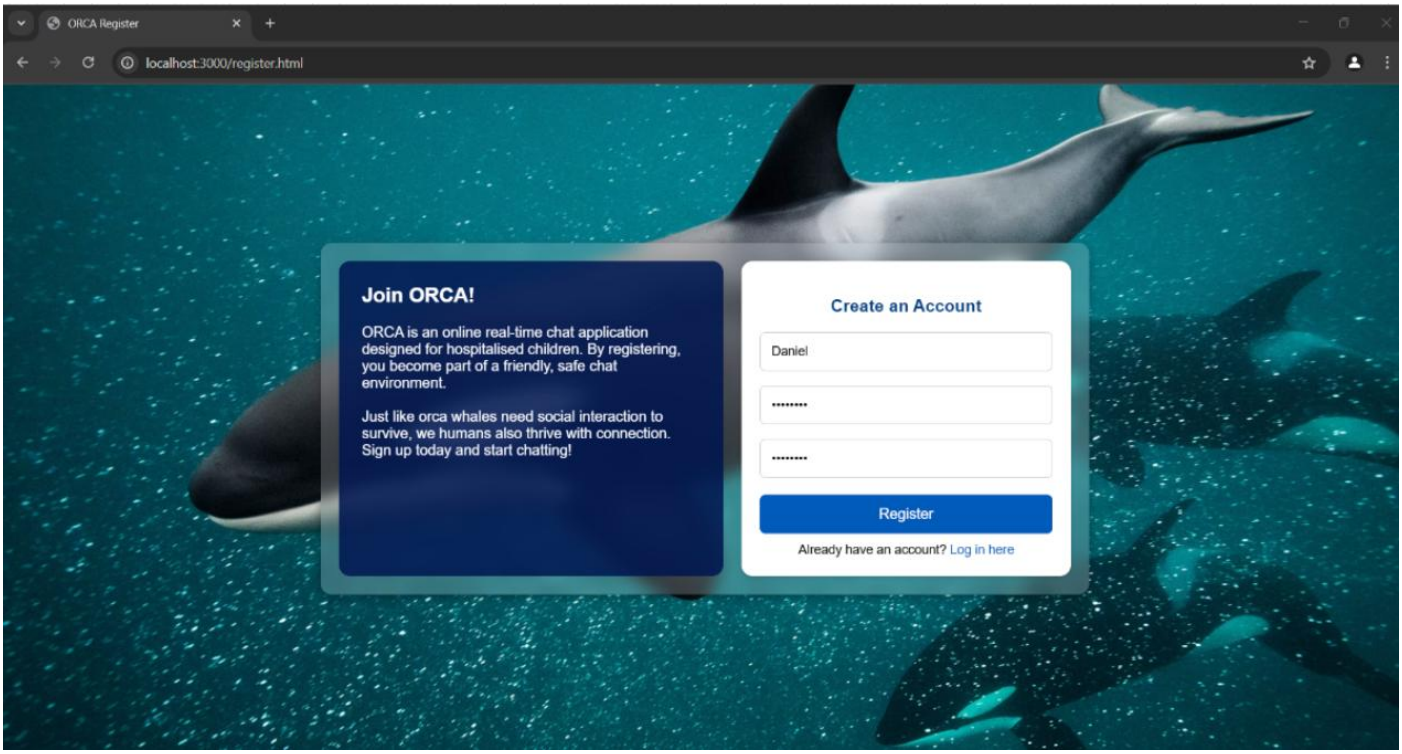
After entering invalid credentials



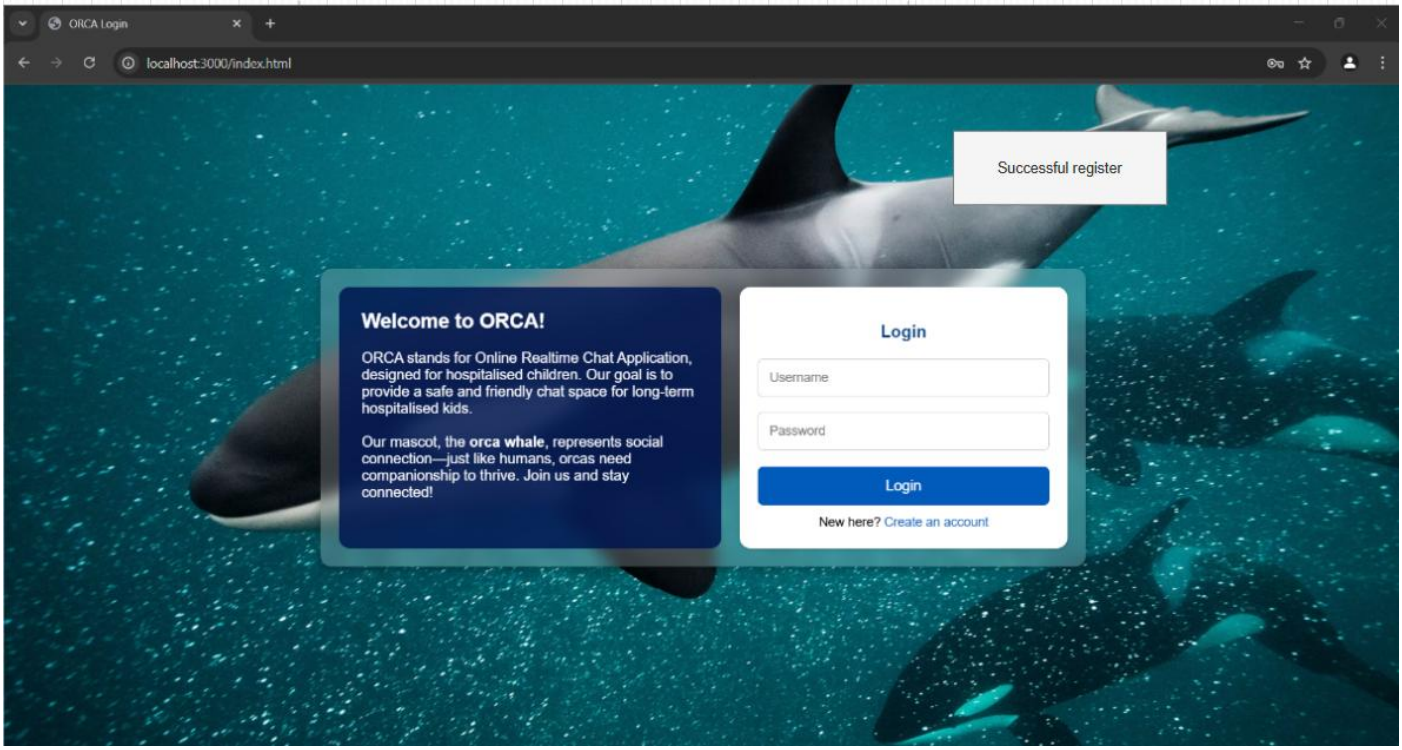
As shown above, an error message is displayed, showing that the user cannot input incorrect login credentials to access the chat panel. Furthermore, an error message is displayed so that the user can be notified of the error. Improving the overall useability and functionality as users can try again with different credentials but also keeping the chat panel secure from unauthorised users.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
13	Ensure that the user cannot register an account on the website using invalid credentials.	I will create an account using the following credentials: Username: Daniel Password: Password	Invalid	The user should not be able to register on the website, creating a new account.

Before entering invalid credentials



After entering invalid credentials



The test was unsuccessful, as the user can create an account with invalid credentials. For my project, the user must have a username larger than 12 characters, a password larger than 8 characters and the username must contain at least two numbers.

Remedial action

To correct this problem, I will add a condition to check whether the credentials that the user has input are valid. I will do this by editing the server-side handle of the password and username validation.

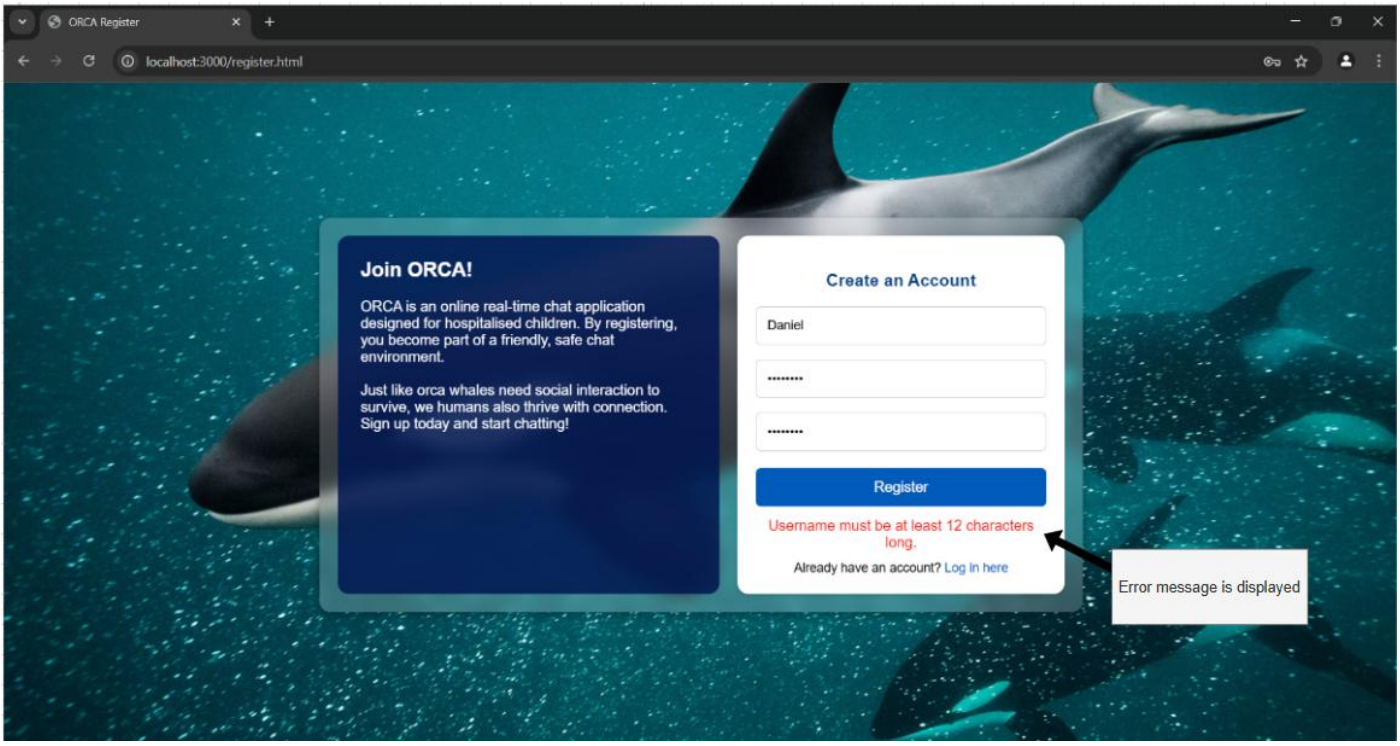
```
8   // Helper function to validate user input
9   const validateUserInput = (username, password) => {
10      if (!username || !password) {
11          return "Username and password are required.";
12      }
13      if (username.length < 12) {
14          return "Username must be at least 12 characters long.";
15      }
16      if ((username.match(/\d/g) || []).length < 2) {
17          return "Username must contain at least two numbers.";
18      }
19      if (password.length < 8) {
20          return "Password must be at least 8 characters long.";
21      }
22      return null; // No validation errors
23  };
```

In the auth.js file, I added this function that will take the username and password and check whether they adhere to the requirements. Also, the function will return the corresponding error message so that the user can understand what they may need to change about their credentials to register.

```
// Register Route
router.post('/register', async (req, res) => {
    try {
        const { username, password } = req.body;

        // Validate user input
        const validationError = validateUserInput(username, password);
        if (validationError) {
            return res.status(400).json({ message: validationError });
        }

        // Check if username already exists
        const userExists = db.prepare("SELECT * FROM users WHERE username = ?").get(username);
        if (userExists) {
            return res.status(400).json({ message: "Username already exists" });
        }

```

Finally, in the '/register' route, I call the function. The function will return an error message if the password of the username doesn't adhere to the requirements. This error message is sent to the client as a 'bad' https response with the error message as the payload to be displayed. However, if both the username and password adhere to the requirements, then no message is returned from the function; therefore, no 'bad' http response is sent.

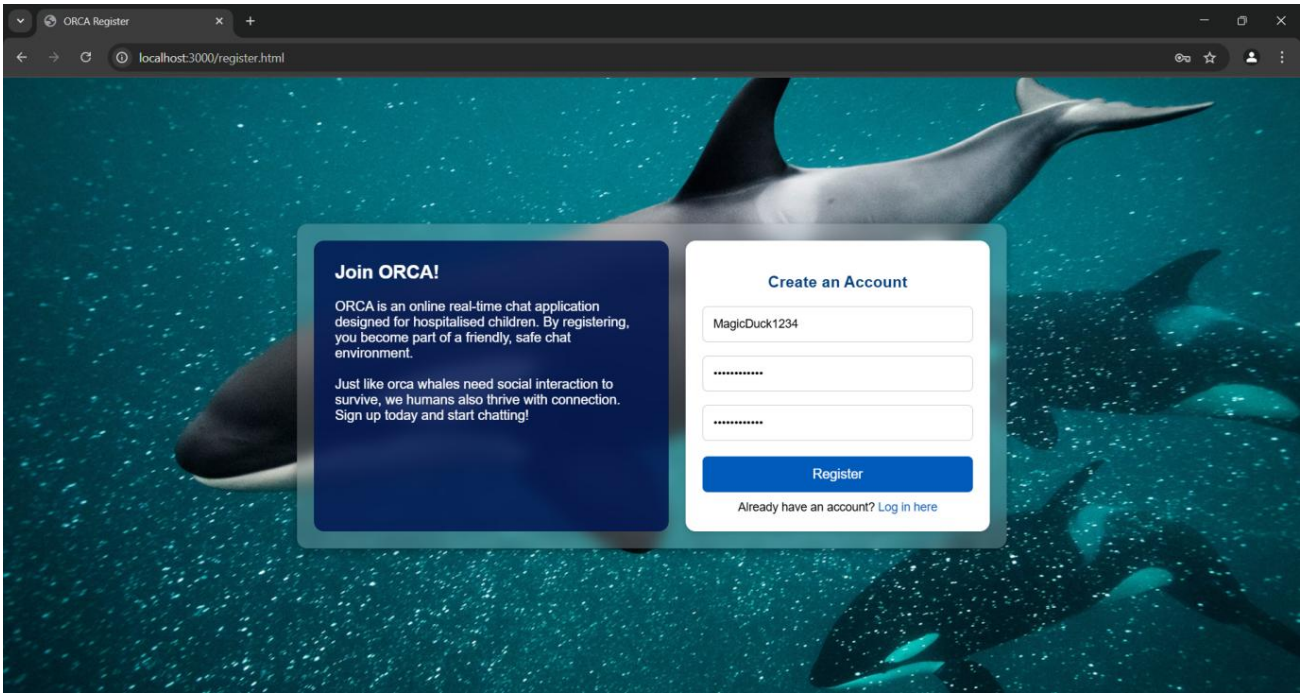
After changes



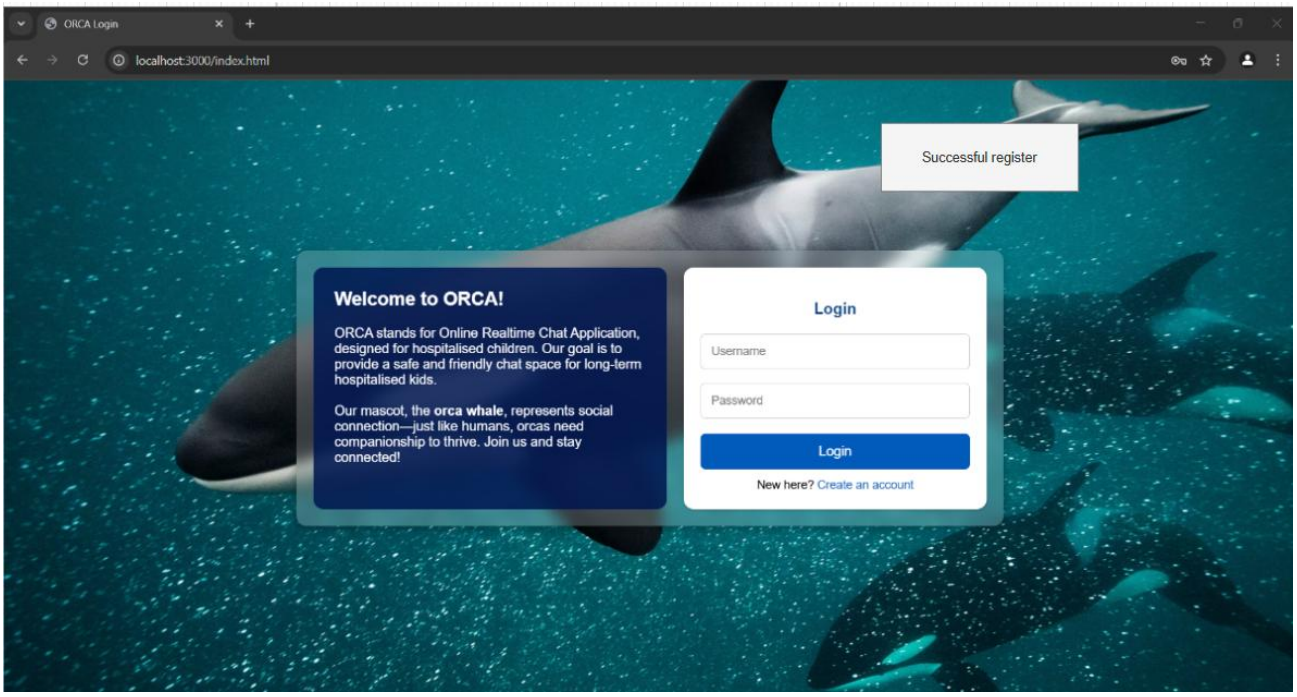
Here, you can see that an error message is now displayed, showing the user that their credentials are invalid and what they must change about it for it to be valid. This is now a successful test after the remedial action.

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
14	Ensure that the user can register an account on the website using valid credentials, then log in.	I will create an account using the following credentials: Username: MagicDuck1234 Password: Password123	normal	The user should be able to register on the website, creating a new account which can be login to access the chat panel.

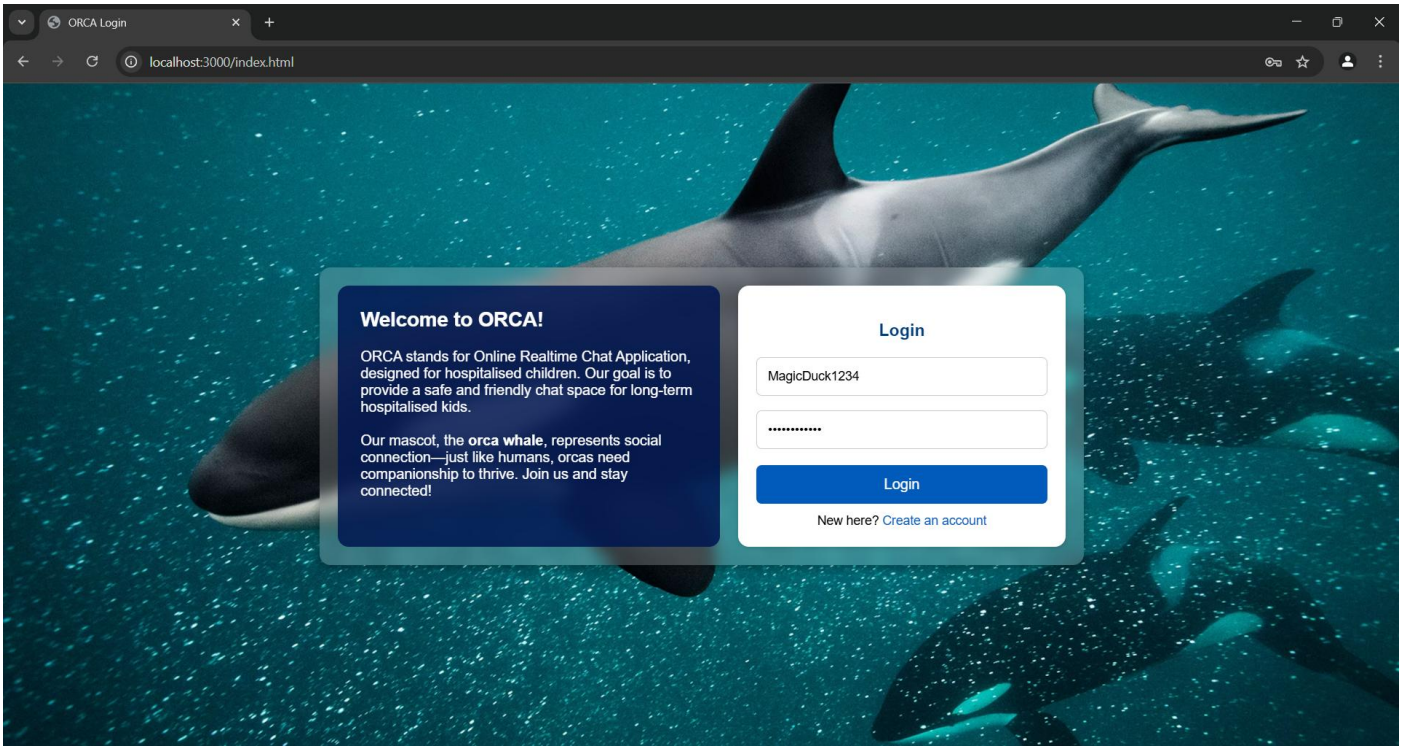
Before entering credentials



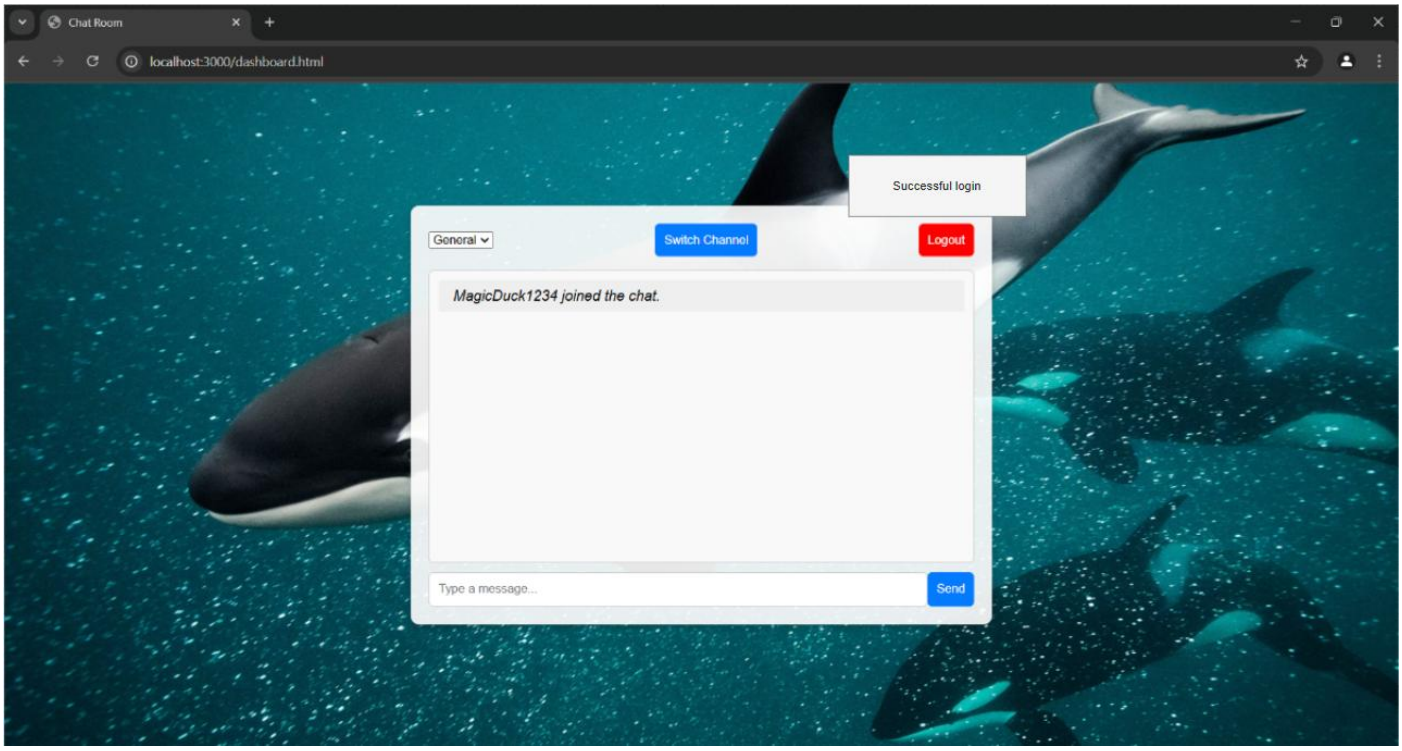
After entering credentials



Before login



After login



Chunk Review (Backend & Networking)

Overall, this section of the project was successful as I was able to fulfil the success criteria fully. The importance of this section is only secondary to the chat functionality, but it's still critical for my stakeholder. This is because a section of the project establishes the framework that the stakeholder is to use to maintain the website. Furthermore, the back end ties together the whole project into a finished solution. To review the rest of this section, the uses of account validation will ensure that invalid accounts can't be created. Moreover, the verification used to verify the user's credentials against the user database when the user logs in will ensure that unauthorized access to the chat panel is denied.
Moving on to the networking section, the stakeholder will be able to run the server on multiple sockets. This is very useful for the stakeholders, for they will have many users wanting to use the website. Too many users on one socket could slow down the performance of the server. However, by splitting the users into different sockets, the load of the many users can be shared equally between multiple cores. This is an efficient way of scaling the website to the demand of the website traffic. Finally, the addition of multiple channels allows users to communicate over multiple channels about different topics. This fulfils an important part of my stakeholder's solution, as users can find channels that they are most interested in and, therefore, meet people with similar interests.

Development (Moderation & User Safety)

The success criteria of moderation and user safety are also very important to the stakeholders, as children will be using the website. Therefore, the website must be safe and free of inappropriate content. I will meet this criterion by censoring the words of the user before they are sent to the server to be displayed to all the users on the client side.

```
31  // Function to censor words in a message
32  function censorMessage(message) {
33      let words = message.split(/\b/); // Split by word boundaries
34
35      return words.map(word => {
36          if (bannedWords.includes(word)) {
37              return "#".repeat(word.length); // Replace with hashtags
38          }
39          return word; // Keep non-banned words unchanged
40      }).join(""); // Reassemble the sentence
41  }
42
```

In this function, I take the message, I split it up into its words, and I run this function on each word using the .map() function. By using the .includes() function, which returns true or false depending on if the word is found within the array. If the word is in the bannedWords array, then each character in the word is replaced with '#'. Finally, the word is then joined back to the message. Once all the words have been checked (censored if a banned word), the message is put back together and then finally returned.

```
// Once the submit button is pressed, the client chencks if the message is not empty
// Then the client sends this message to the client
form.addEventListener('submit', (e) => {
    e.preventDefault();
    if (input.value) {
        // Creates an unique identifier for each message
        let clientOffset = `${socket.id}-${Date.now()}`;
        socket.emit('chat message', censorMessage(input.value), clientOffset, channel_ID, username);
        input.value = '';
        input.placeholder = "Type a message...";
    } else {
        input.placeholder = "Invalid input";
    }
});
```

Here is how the function is used; when sending the message to the server, the message is censored for banned words.

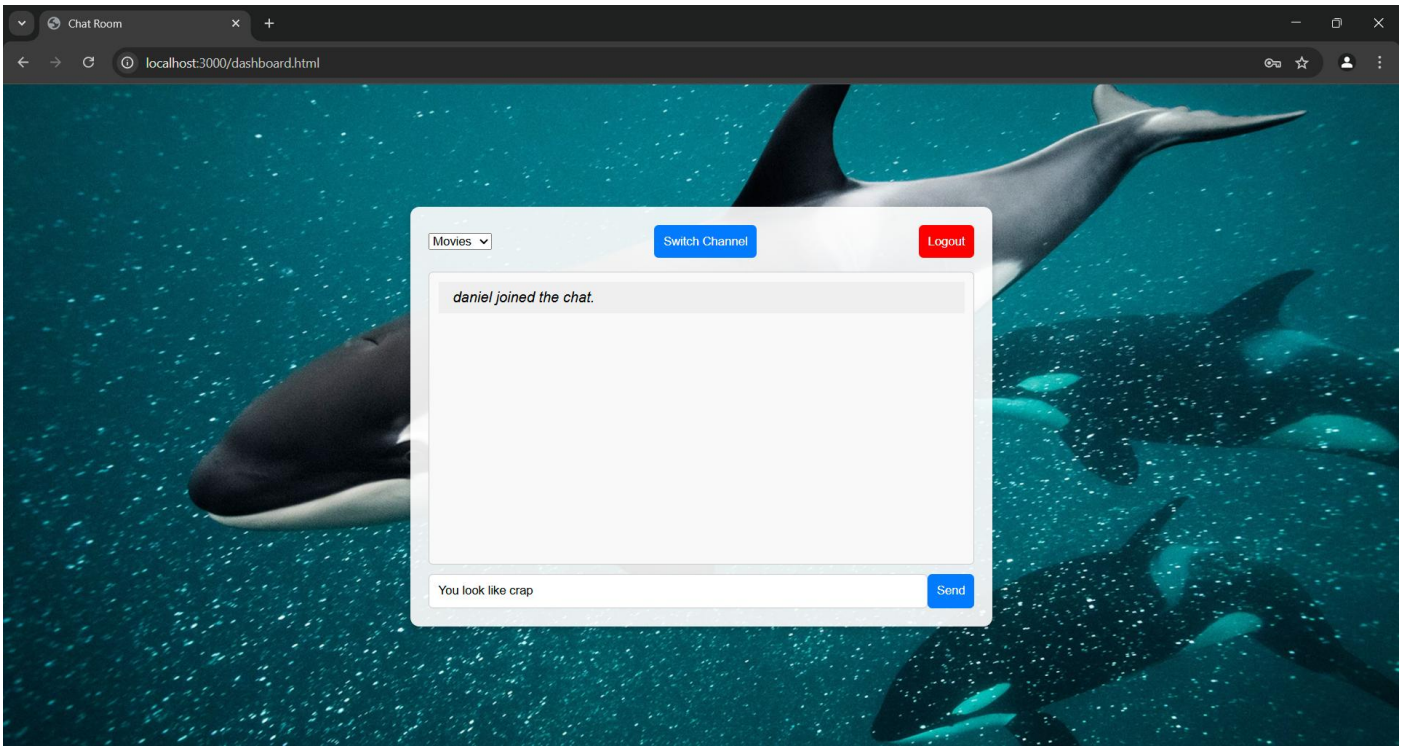
```
// List of banned words (you can expand this list)
const bannedWords = ["crap", "curse", "offensive"];

```

Here is the bannedWords array; it can be updated with more words or linked to an external banned words text file. However, for testing purpose these will be the banned words.

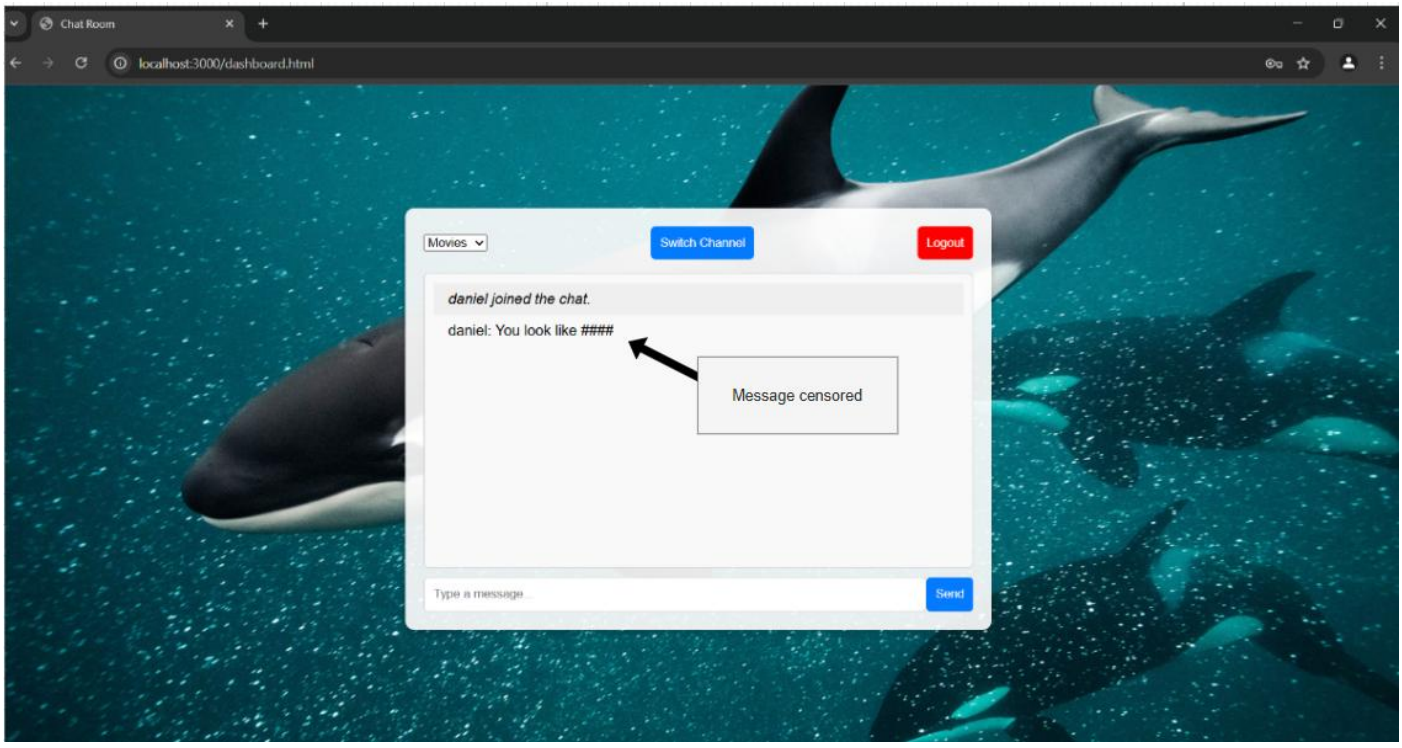
Iterative Tests (Moderation & User Safety)

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
15	Ensure that the user couldn't send banned words in the chat panel.	Entering banned words which are deemed inappropriate (They are stored in an external library), I can test to see that inappropriate language is censored, safeguarding the children who use the website. For example, entering: 'You look like crap'. It will be sent to the server where it should be censored.	normal	Once the message is received by the server, the server will compare it to the banned words. Checking if the message contains banned words, censoring them if found. So, in the example, the message: 'You look like crap' Will become: 'You look like ####' This is how it will be outputted to the rest of the users.

Before sending message

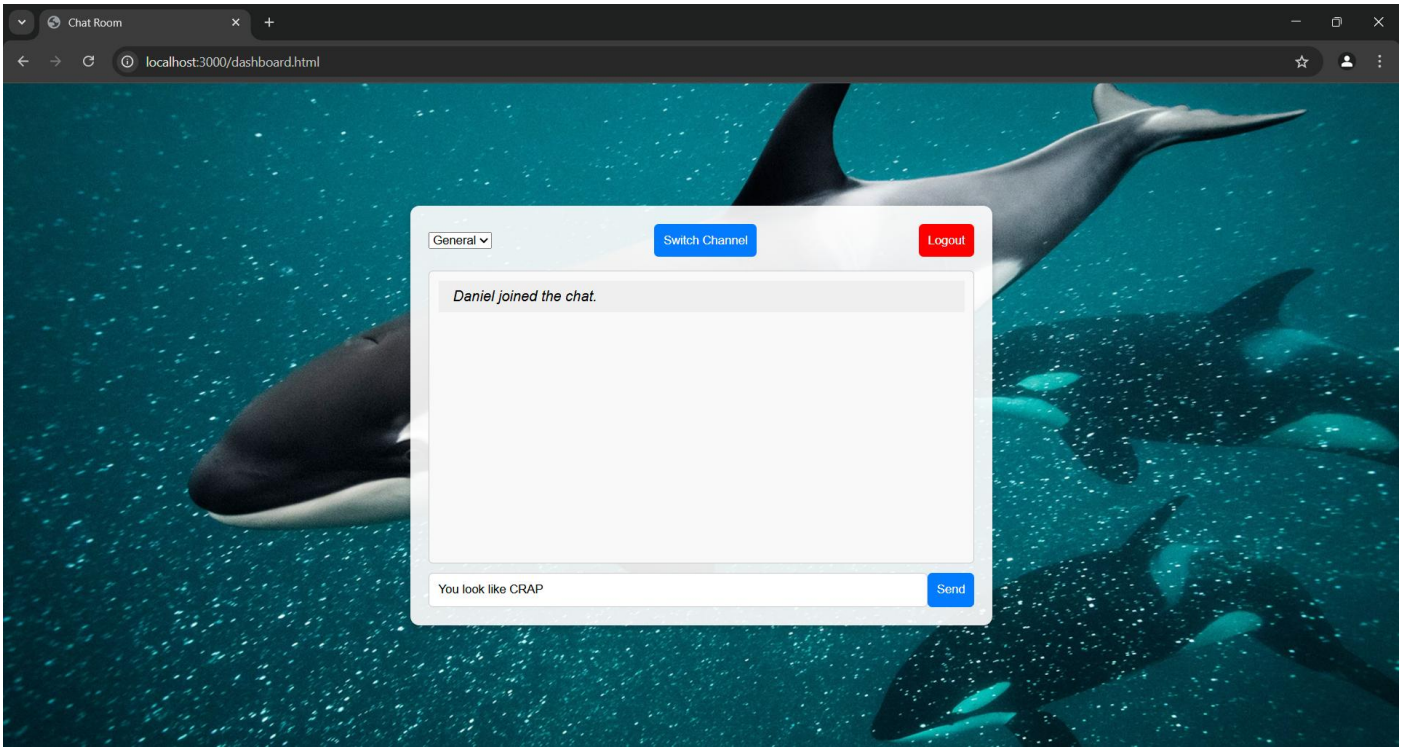


After sending message

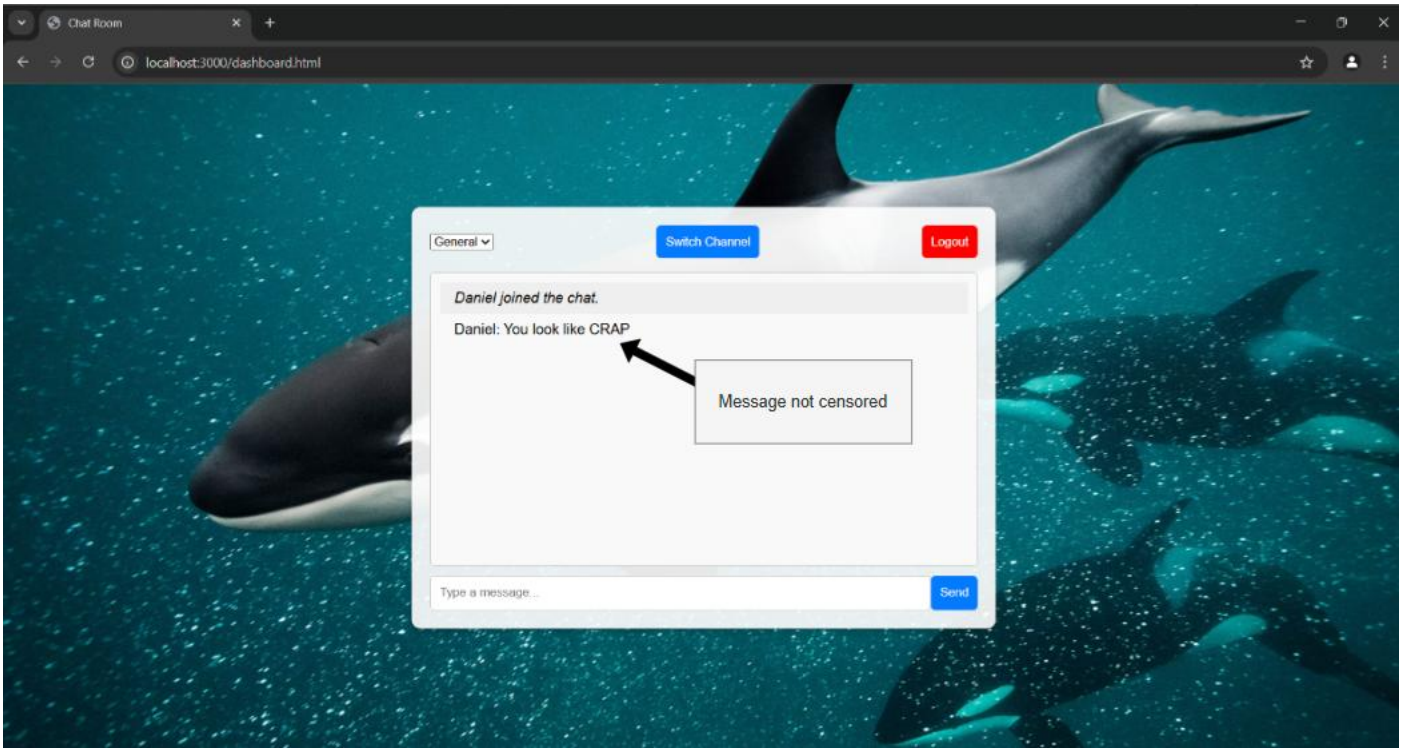


Test No.	Test Purpose	Test Data	Data type	Expected Outcome
16	Ensure that users cannot bypass the chat filter using capitalisation	I will send the message: 'You look like CRAP'	normal	Once the message is received by the server, the server will compare it to the banned words. Checking if the message contains banned words, censoring them if found. So, in the example, the message: 'You look like CRAP Will become: 'You look like ####' This is how it will be outputted to the rest of the users.

Before sending message



After sending message



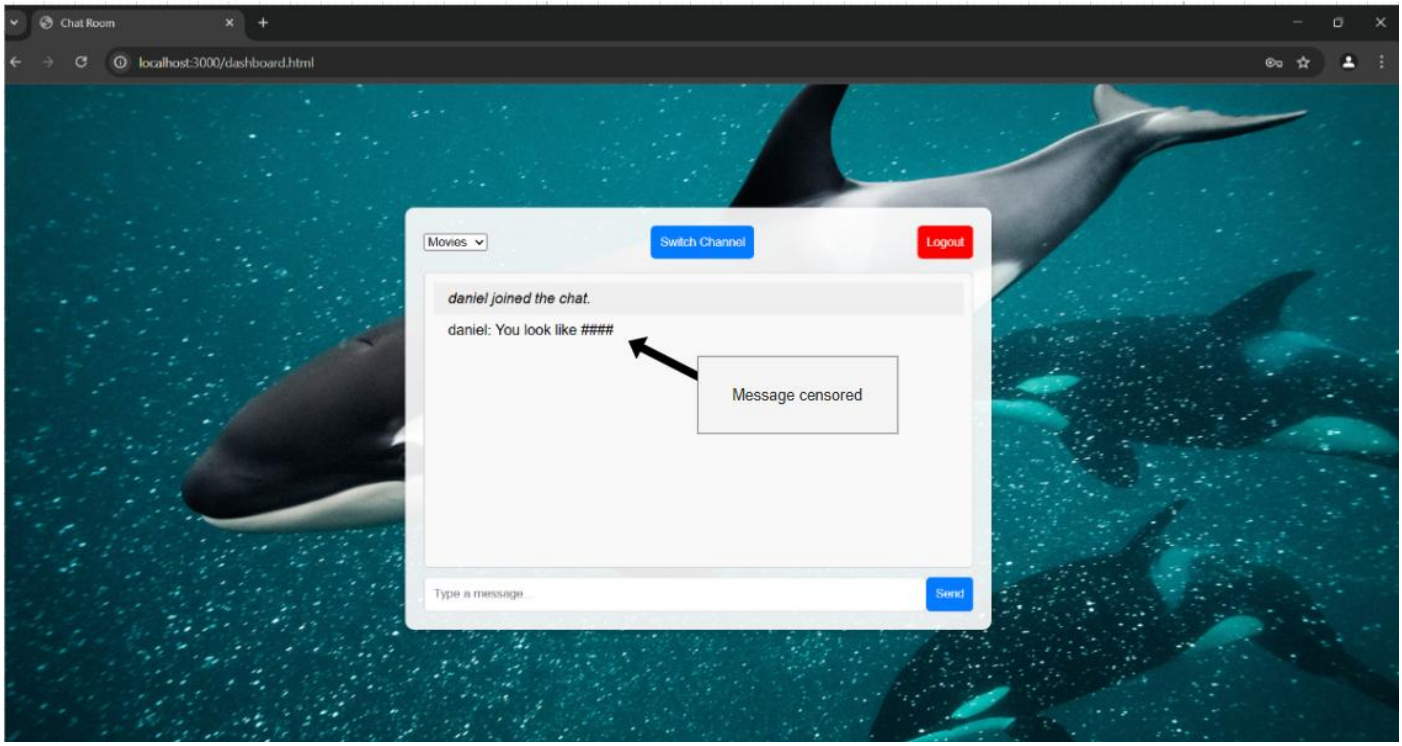
This test was unsuccessful, as the message was not censored. The user mustn't be able to bypass the chat filter, as this puts other users at risk. Therefore, the chat filter must be more robust to counteract potential bypasses.

Remedial action

```
33  // Function to censor words in a message
34  function censorMessage(message) {
35      let words = message.split(/\b/); // Split by word boundaries
36
37      return words.map(word => {
38          let lowerCaseWord = word.toLowerCase(); // Puts the word into lowercase
39          if (bannedWords.includes(lowerCaseWord)) {
40              return "#".repeat(word.length); // Replace with hashtags
41          }
42          return word; // Keep non-banned words unchanged
43      }).join(""); // Reassemble the sentence
44  }
```

To prevent the user from bypassing the chat filter by using capitalisation, I will convert the work into lowercase before comparing it to the banned words in the banned words array. This will counteract the user's attempts at bypassing and improve the maintenance of the website. This is because a developer only needs to add the word once into the banned words array and not multiple versions of the word capitalised.

After changes



Here, you can see that now, even when the banned word is capitalised, the banned word will still be censored. Therefore, I now consider this test a success.

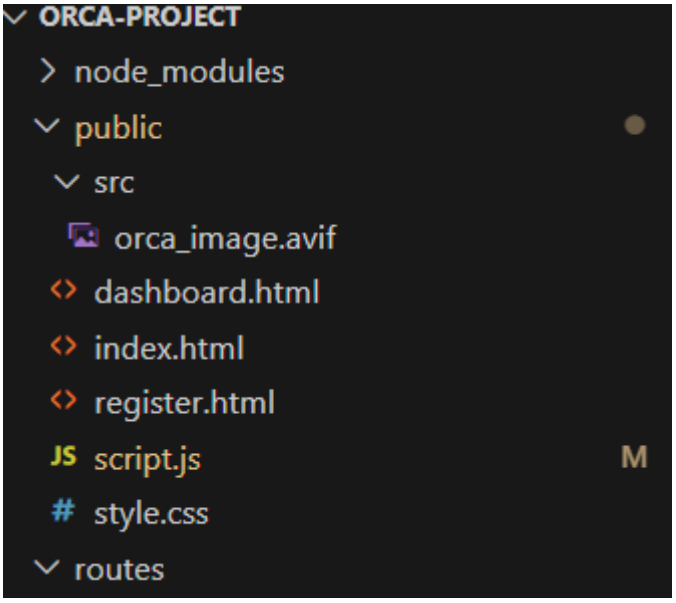
Chunk Review (Moderation & User Safety)

Overall, in this part of my project, I was able to complete what I set out to do, which was to create a chat filter. This chat filter's purpose is to safeguard children from the users of the website, which will have a range of younger and older children using the website. Furthermore, I was successful in implementing a simple method chat filtering that requires no server-side processing. Moreover, the implementation of converting the words into lowercase will improve maintenance for the developers when they would want to add more banned words to the banned words array, as they would not have to add different versions of the word in lowercase and uppercase. However, I would have liked to add an admin panel, which would have given admins the ability to delete messages and users. This would've been a further level of safeguarding as messages that are inappropriate and are not censored by the chat filter could be deleted.

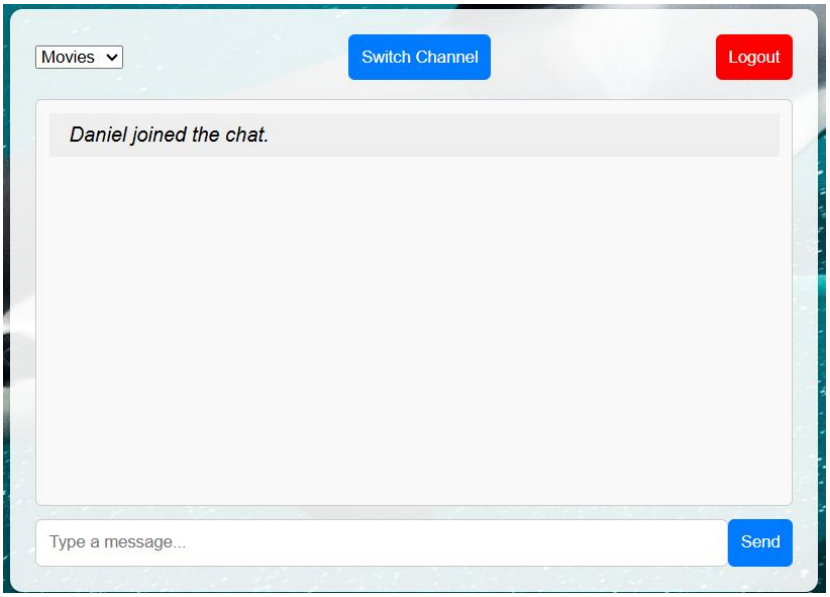
Development (Website functionality & User Interface)

In this section, I aim to create functional elements for the user interface like buttons, links and improve the overall design of the website.

First to create the background of the website I downloaded an image of orcas from the internet and stored in within the source folder (src), which is public to the client. The image is to be rendered on the client's browser and cached for faster retrieval when switching pages.



Next, I will create the many buttons used to navigate the website and discuss their structure. Although most of these features have been discussed in previous sections of this document, I will give a brief overview of them. All buttons on the website are rendered and processed on the client side; this is to reduce server workload and makes the buttons highly responsive.



The logout button redirects the client to the login page by changing the URL of the user's browser to the index page (which is the login page).

```
45      function logout() {
46          window.location.href = 'index.html';  // Redirect to index.html
47      }
```

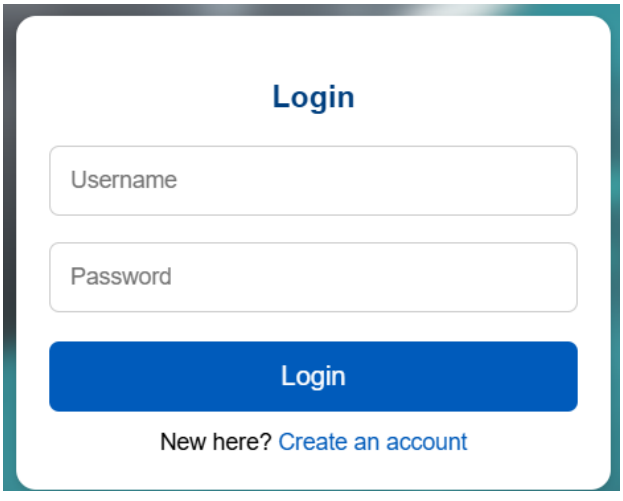
The send button is a HTML form. This form is listened to by the scripts.js file on the client side for when the user clicks submit (the send button). The code that then runs will get the message from the input box (the chat bar) and send it to the server to be processed.

```
46  // Once the submit button is pressed, the client chencks if the message is not empty
47  // Then the client sends this message to the client
48  form.addEventListener('submit', (e) => {
49      e.preventDefault();
50      if (input.value) {
51          // Creates an unique identifier for each message
52          let clientOffset = `${socket.id}-${Date.now()}`;
53          socket.emit('chat message', censorMessage(input.value), clientOffset, channel_ID, username);
54          input.value = '';
55          input.placeholder = "Type a message...";
56      } else {
57          input.placeholder = "Invalid input";
58      }
59  });
```

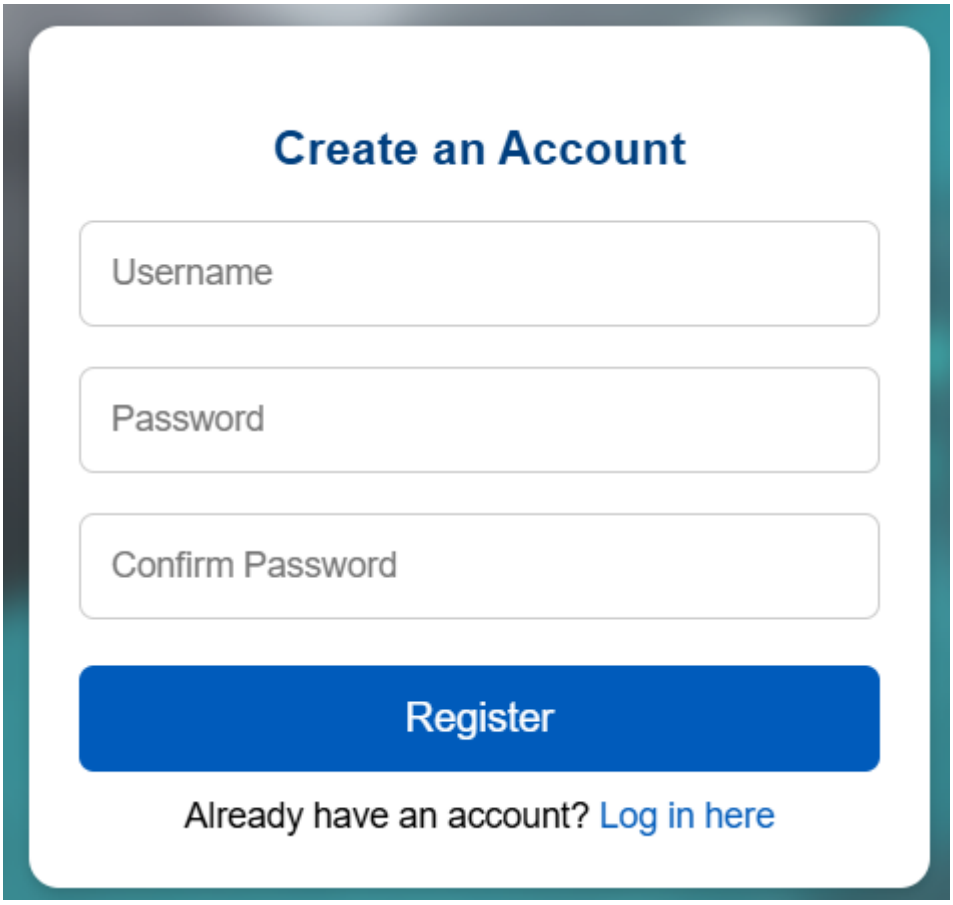
Next the switch channel button works by retrieving the current value of the drop-down menu on the top left side once clicked. This new channel is sent as an event to the server as the join channel event, which will switch the client's current channel.

```
35      // When the client switches channels this event will run
36      function switchChannel() {
37          let newChannel = document.getElementById("channel-select").value;
38          socket.emit('join channel', newChannel, username, (response) => { // the client joins the new channel
39          console.log(response);
40          document.getElementById("messages").innerHTML = ""; // Clear messages when switching
41      });
42      channel_ID = newChannel;
43      }
```

On the login page, I have a button and a link to help the user navigate the website.



The login button will retrieve the username and password input boxes and send them to the server to be processed. The link below the login button will redirect the user to the register page when clicked.

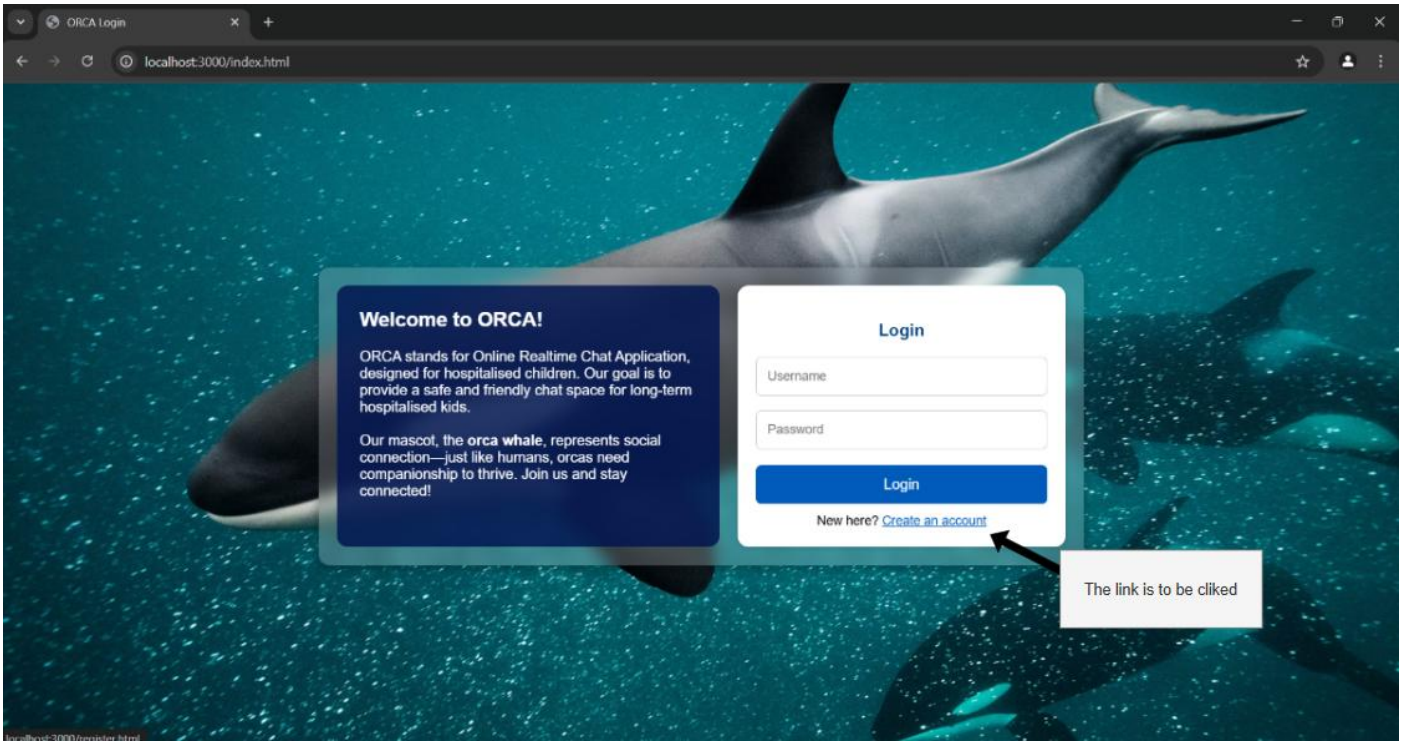


The register page works the same as the login page. It has a register button that will retrieve the username and password and send it to the server to be processed. Likewise, the login link at the bottom will redirect the user to the login page when clicked.

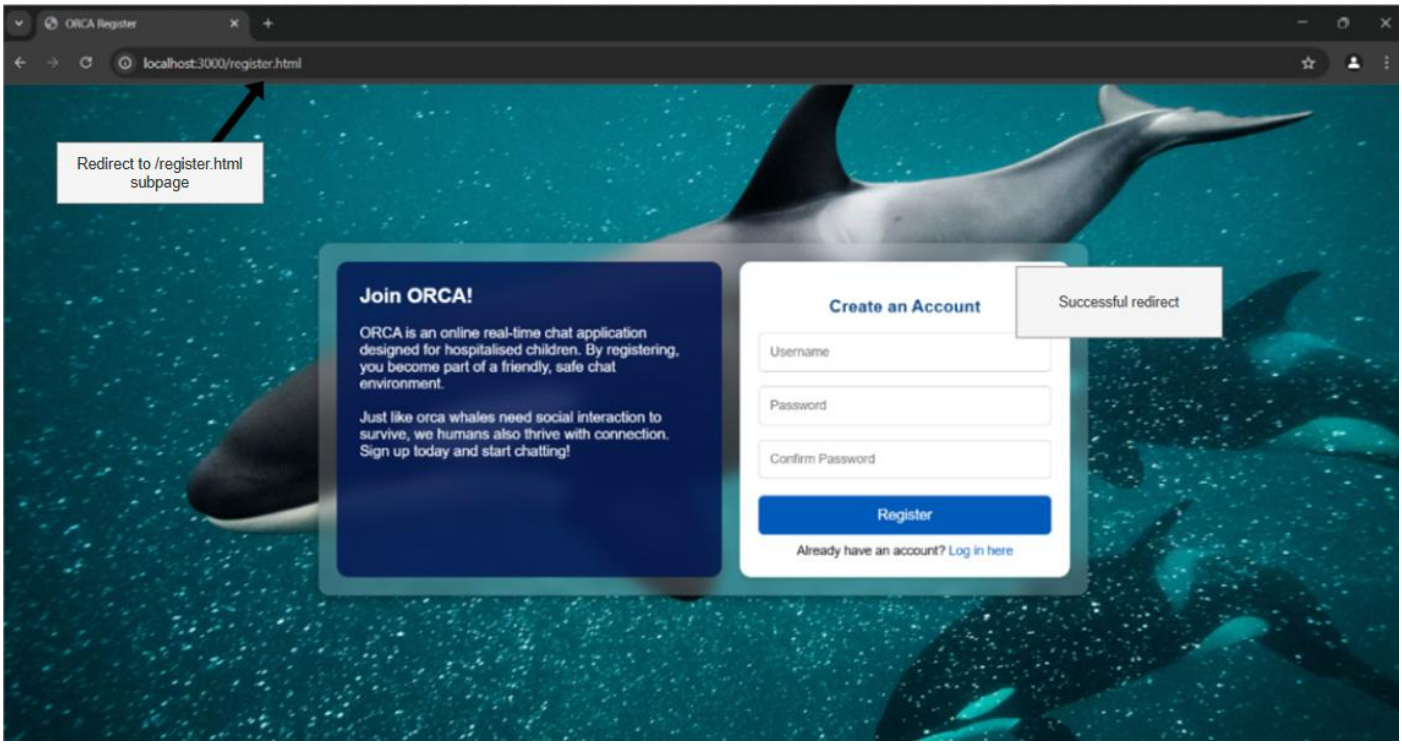
Iterative Tests (Website functionality & User Interface)

Test No.	Test Purpose	Test Data	Data type	Expected Outcome
17	Ensure that a user can interact with the website.	Clicking on buttons to test if the website outputs to the user. For example, users should be able to click on links to take them to subpages on the website. Race Cars – This should take the user to the topic channel (subpage) on race cars.	normal	The user should be able to interact with the website when clicking on buttons. Also, when the user clicks on the link they should be taken to the corresponding subpage (topic channel).

Before clicking on link



After clicking on link



Chunk Review (Website functionality & User Interface)

Overall, this chunk was a success as the overall design of the website uniform between all pages and the buttons all function. This chunk is important to my stakeholder because the website needs to be navigable by a range of users with differing ages. Other websites can be difficult for younger users to navigate due to cluttering and small text. The design of my website is simple, clear and colourful, which also improves accessibility. This is because users with physical disabilities like visual impairment will be able to use the website due to its striking colours and large fonts and buttons.

EVAULATION

In this section, I will evaluate how successful I was in completing my four success criteria by using my post-development test plan. In this test plan, I will include white box testing where I will be evaluating the functionality of the internal code on its robustness, usability and if the functions run as expected. Also, I will conduct black box testing by having a range of participants of varying ages to assess the user's experience. Furthermore, these participants will be one older child (+13) and one younger child (-13) to help me understand if the solution is successful with other age groups.

Finally, I will also conduct new tests to ensure that I cover the entire scope of the project. Here are my four success criteria:

1) **Moderation and user safety** – Ensuring security and safety on the website is especially important for my stakeholders since they are responsible for the care of the children and young people who will use the website.
2) **Real-time chat** – Creating a website with real-time functionality will be the most important feature of the project since it is the main solution of my stakeholder.
3) **Website functionality and user interface** – This is not as complex as real-time chat capabilities, but it will pose a challenge as it can be very time-consuming to create a user-friendly website with topic channel capabilities. Furthermore, I must create a user-friendly website since young people of all ages will be using the website. However, I plan to use pre-made assets and designs to help create a good user interface to minimise time spent on graphical improvements.
4) **Networking and backend** – This is the most complex feature of the project since it will be a new problem for me to solve. Fortunately, many applications like node.js can help me to manage most traffic to the website and network for the website. Node.js especially allows me to create multi-level user access and uses JavaScript as its programming language, which makes it easier for me to transfer the skills I already have.

Post Development Tests for Function

Success Criteria No.	Test No.	Test Purpose	Test Data	Expected outcome	Outcome	Page No.
2	1	Ensure users can send normal messages over the website in real time	Entering the messages: "Hi, how are you?" "I'm good, thanks! How about you"	The user should be able to view the messages after the server outputs the received messages.	Success	95
2	2	Ensure that multiple users can send messages on a topic channel.	By using three users, I will input: User 1 – "Hello, my name is Daniel". User 2 – "Hello, my name is John". User 3 – "Hello, my name is James". User 4 – "Hello, my name is Ben". All at the same time.	When the server receives all these messages, the server should output them one at a time. The messages should be displayed in order of the time the server receives the messages.	Success	96
4	3	Ensure recording of message history and chat logs	Users should be able to see previously sent messages on the topic channels. By entering the messages: "Hi, this is John." "Hello, this is Jane."	The server should save previous messages even after the user leaves the website. Once the new user joins the topic channel, I should be able to see:	Success	97 - 98

			From two different accounts, then joining on a new account.	"Hi, this is John." "Hello, this is Jane."		
1, 4	4	Ensure that users can access different topic channels that are appropriate to their age.	Using a user with the access level of 'olderChildren' and a user with the access level of 'YoungerChildren' to check if topic channels for older children (13+) are only accessible by them and admins. Also, check to make sure that the older children cannot access younger children's only topic channels.	Older children trying to access younger children's topic channels should see a pop-up telling them that they cannot access the topic channel. This should happen when younger children try to access older children-only topic channels as well.	Fail	98
3	5	Ensure that a user can interact with the website.	Clicking on links to assess if the website outputs to the user. For example, users should be able to click on links to take them to subpages on the website. Create an account – This should take the user to the register page (subpage).	The user should be able to interact with the website when clicking on links. Also, when the user clicks on the link, they are redirected to the corresponding subpage (the register page).	Success	99
4	6	Ensure that the user can join the server on different sockets.	I will create four instances of the website, and using four different accounts, I will join the server on four different sockets. localhost:3001, localhost:3003, localhost:3006 localhost:3007 And then send the message (Hello all) to ensure that all users are in-sync.	The server should be available on multiple sockets and receive messages in sync with the server.	Success	100
3	7	Ensure that users can switch channels	Using two users, I will have then in two different channels; I will have one send a message in their channel. Then, I will show the other user joining their channel.	The users should be able to send messages on multiple channels, and the messages sent on different channels should only be received by those in that channel. However, once a user joins the channel, they should receive all the past messages sent.	Success	101
4	8	Ensure that users can log in to the website with a valid account.	I will give the user a valid account. Username: MagicDuck1234 Password: Password123	The user should be able to access the chat panel after entering the correct credentials.	Success	102
4	9	Ensure that the user can register	I will create an account using the following credentials:	The user should be able to register on the	Success	103 -104

		an account on the website using valid credentials, then log in.	Username: MagicDuck1234 Password: Password123	website, creating a new account which can be used to login to access the chat panel.		
1	10	Ensure that the user cannot send banned words in the chat panel.	Entering banned words which are deemed inappropriate (They are stored in the banned words array), I can test to see if inappropriate language is censored, safeguarding the children who use the website. For example, entering: 'You look like crap'. It will be sent to the server where it should be censored.	Once the message is received by the server, the server will compare it to the banned words. Checking if the message contains banned words, censoring them if found. So, in the example, the message: 'You look like crap' Will become: 'You look like ####' This is how it will be outputted to the rest of the users.	Success	105
1	11	Ensure that Admins can delete, create, and alter users.	Using the admin panel, I will test user creation, deletion and alteration. For example, I will enter in (in creation mode): Krp, RedRider, Password123!, Daniel, Okafor, 12/8/2007 (corresponding with the user table). This will test the user creation. Entering in: 21, Password, newpassword123! When in altering mode, it should alter the user with ID 21. Entering: "21" When in deletion mode, should we delete user 21 from the table?	When using the admin panel, the admin should be able to use the different modes to delete, create, and alter users. In the example (when in creation mode), entering in: Krp, RedRider, Password123!, Daniel, Okafor, 12/8/2007 It should be entered into the next free space in the table. In the example (when in altering mode) entering: 21, Password, newpassword123! Will change the password of the user with the ID 21 in the table to "newpassword123!". In the example (when in deletion mode), entering in: "21". Should delete the user with the ID 21 from the table.	Fail	106

Post Development Tests for Robustness

Success Criteria No.	Test No.	Test Purpose	Test Data	Expected outcome	Outcome	Page No.
2	12	Ensure that users can send boundary messages over the website in real time.	I will send a message with special characters like "&<>", ensuring that special characters can be sent as well.	The user should be able to view the messages as they are entering once the server outputs the received messages.	Success	106 -107
2	13	Ensure that users cannot send invalid messages over the website in real time.	Entering an empty message	The message should not be sent to the server, instead, the user should be warned of their invalid input.	Success	107 -108
2	14	Ensure that messages that are sent while the user is disconnected are sent once the user reconnects.	By disconnecting user 1, then sending the message "I think that I'm disconnected". Then reconnecting the user back to the server.	The messages should be sent once the user reconnects to the server. We should see user 2 receive the messages once user 1 reconnects.	Success	108 -109
2	15	Ensure that when a user connects back to the server, they are updates on any messages sent during disconnect.	User 1 will send - "Hi Jamie" Then, user 2 will send "Hi Daniel". To ensure that the two users can receive messages before disconnecting user 1. Once user 1 is disconnected, user 2 will send – "Where did you go, Daniel?". Then, User 2 will reconnect to the server.	The server should save previous messages even after the user disconnects from the website. Once the user 1 reconnects to the topic channel, user 1 should have all the messages sent by user 2 while user 1 was disconnected. Displaying – "Where did you go, Jamie?"	Success	110 -111
4	16	Ensure that the user can't register an account that has already been created.	Using the credentials below, I will register a new account. Username: MagicDuck1234 Password: Password123 (This account has already been created.)	The user should be warned with an error message that the username has been created already.	Success	112
4	17	Ensure that the user cannot log in with invalid account information.	I will give the user an invalid account Username: InvalidAccount12 Password: Password1234	Invalid credentials should warn the user with an error message. Also, the user should not be redirected to the chat panel.	Success	113
4	18	Ensure that the user cannot register an account on the website using	I will create an account using the following credentials: Username: Daniel Password: Password	The user should not be able to register on the website, creating a new account.	Success	114

		invalid credentials.				
1	19	Ensure that users cannot bypass the chat filter using capitalisation	I will send the message: 'You look like CRAP'	The message below should be displayed: 'You look like CRAP Will become: 'You look like ####' This is how it will be outputted to the rest of the users.	Success	115

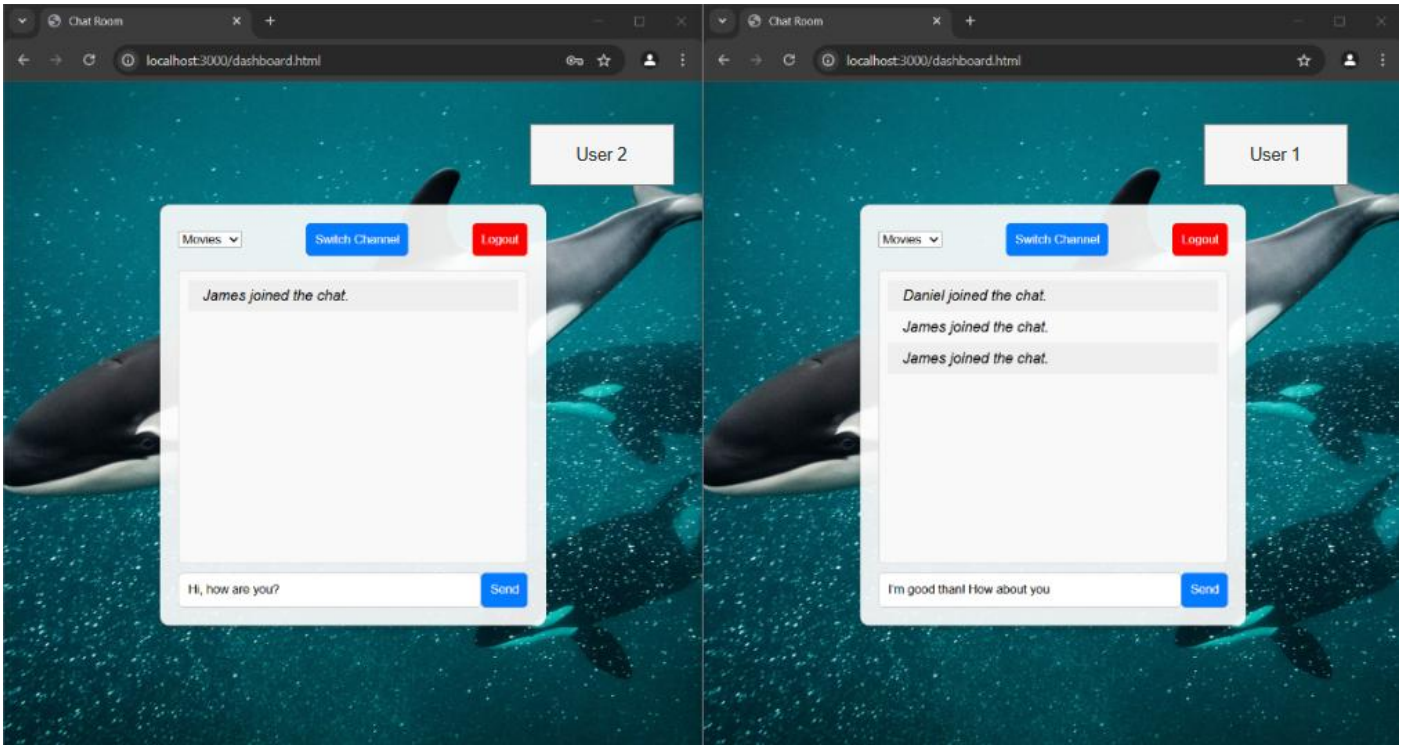
Post Development Tests for Usability

Success Criteria No.	Test No.	Test Purpose	Test Data	Expected outcome	Outcome	Page No.
3	20	Ensure that the user can navigate the login page using the link on the page.	Let the Older child (+13) navigate the login page with no instruction and see if he can navigate the login page with ease. Then do the same with the younger child (-13)	Both age groups should be able to navigate the login page with ease.	Success	116 -117
3	21	Ensure that register errors are communicated to the user clearly.	Using the same two groups of an older child (+13) and a younger child (-13), I will let them set up accounts, recording their attempts.	Both age groups should be able to register on the website with ease.	Success	117 -120
3	22	Ensure that login errors are communicated to the user clearly.	Using the same two groups of an older child (+13) and a younger child (-13), I will give them prearranged accounts that they will need to log in with. I will also be recording their attempts.	Both age groups should be able to log in to the website with ease.	Success	121-122
1	23	Ensure that the user can communicate over the website with ease in the chat board.	Using the same two groups of an older child (+13) and a younger child (-13), I will let them communicate with one another at the same time to see if communication over the website is user-friendly.	Both age groups should be able to communicate over the website in the chat board with ease.	Success	123
3	24	Ensure that a user can switch channels with ease.	Using the same two groups of an older child (+13) and a younger child (-13), allow them to test to see if they can switch channels with ease.	Both age groups should be able to switch channels with ease.	Success	124

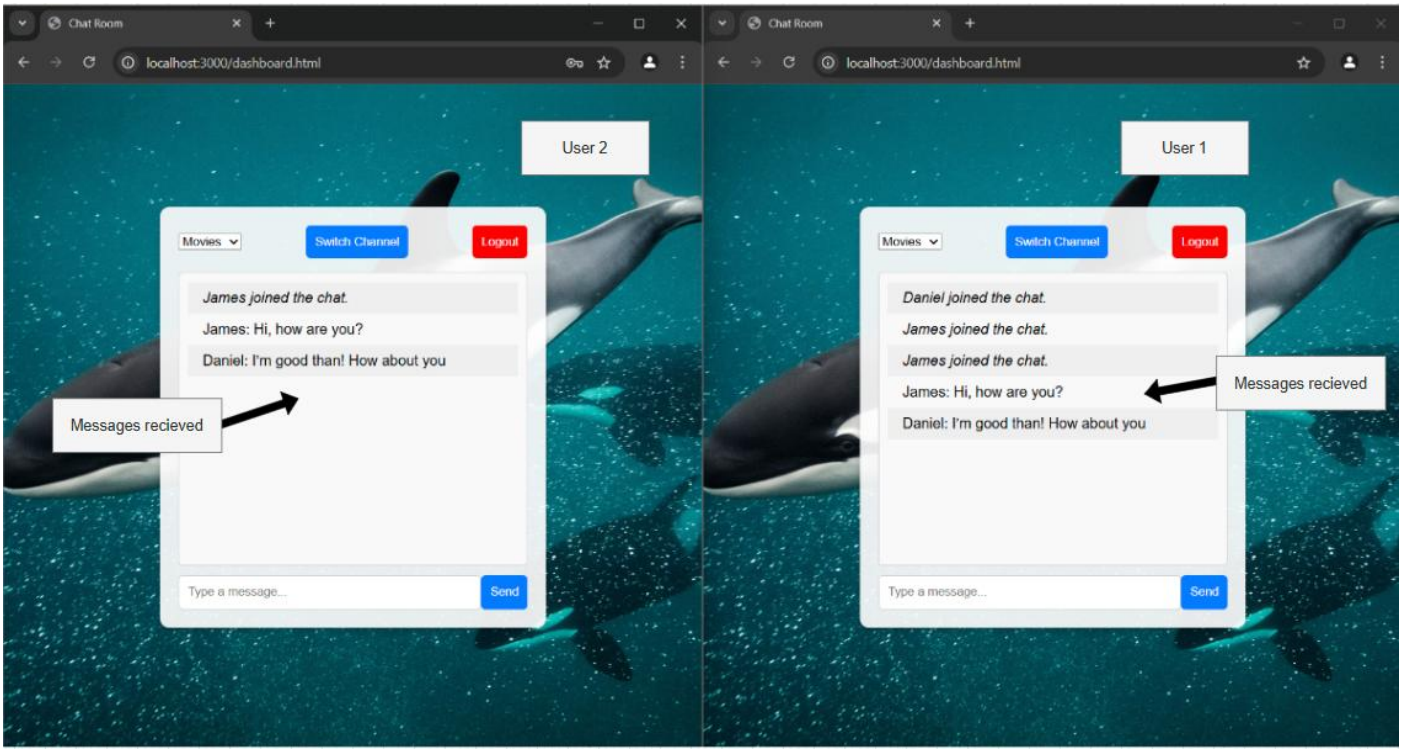
Post Development Tests

Test 1

Before sending message



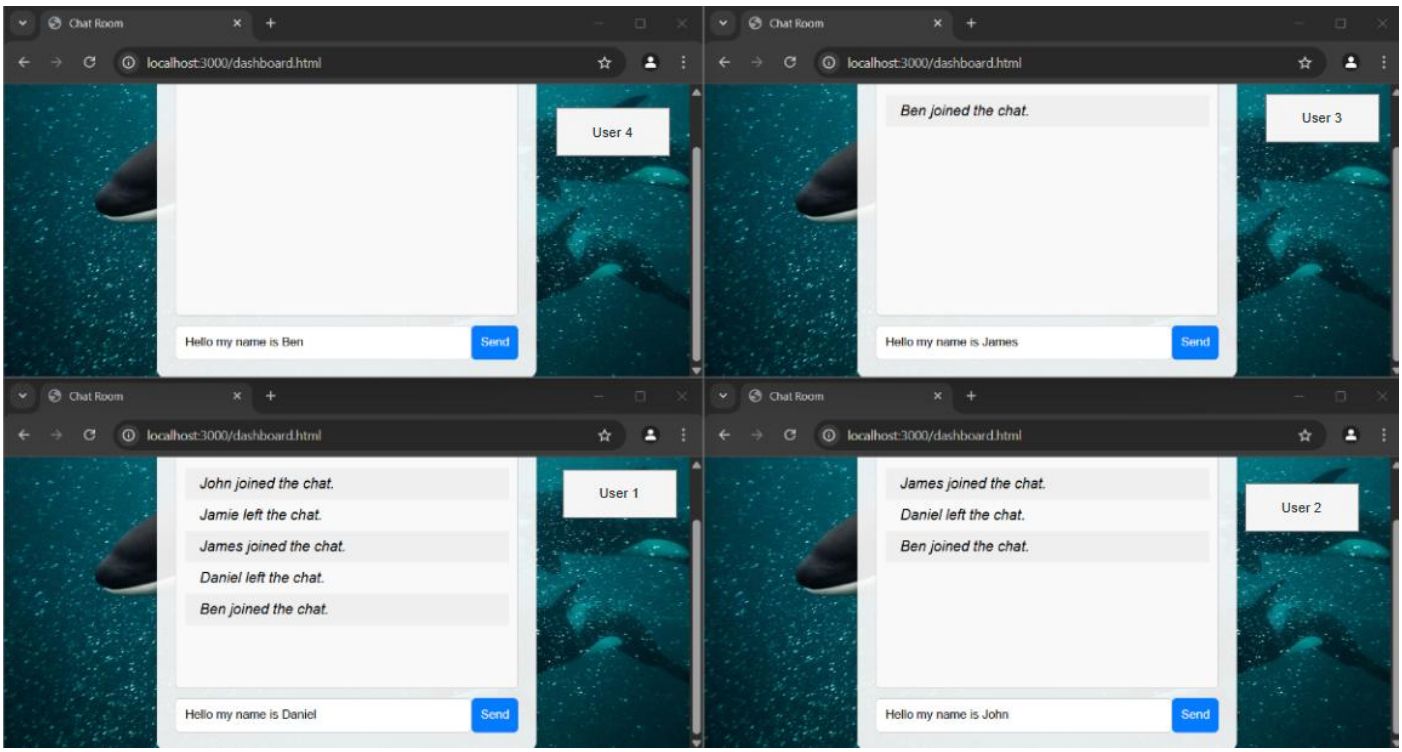
After sending message



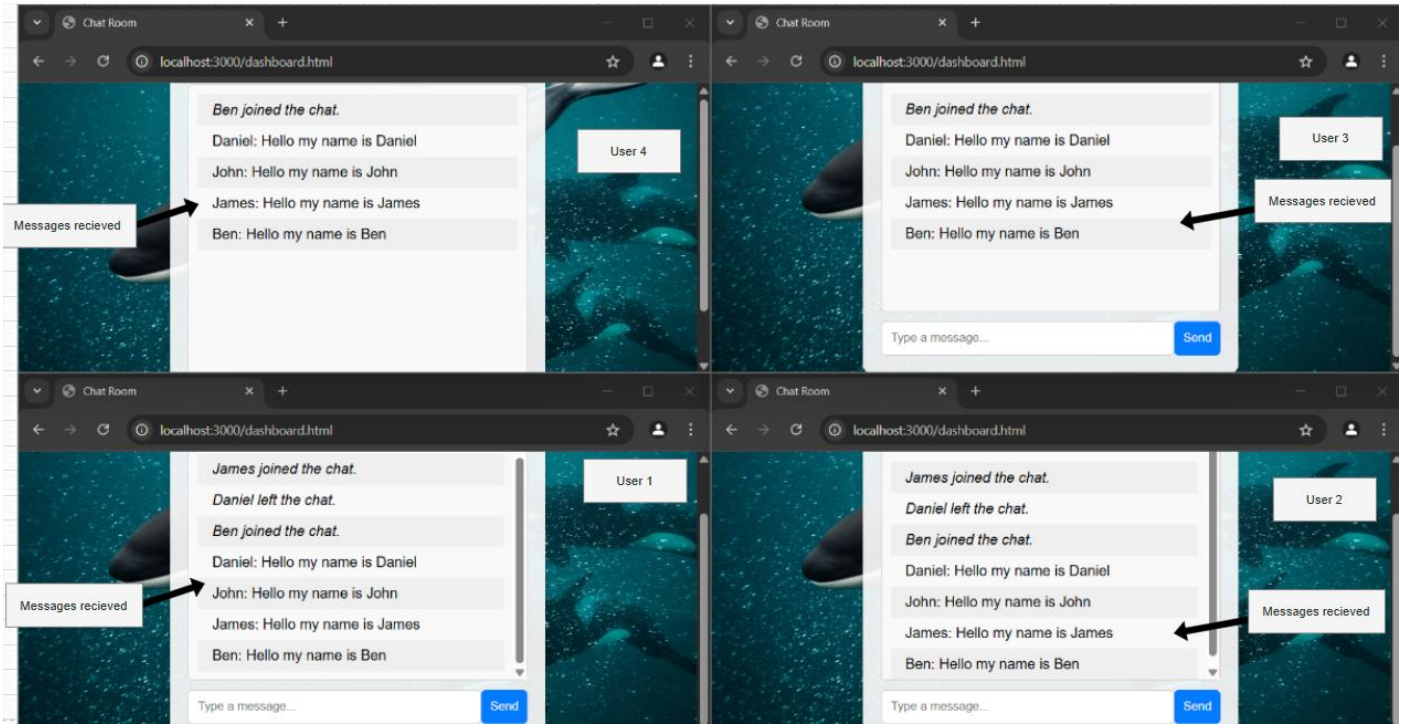
This test was a success because messages were received by both users, therefore, the sending message function works.

Test 2

Before sending messages



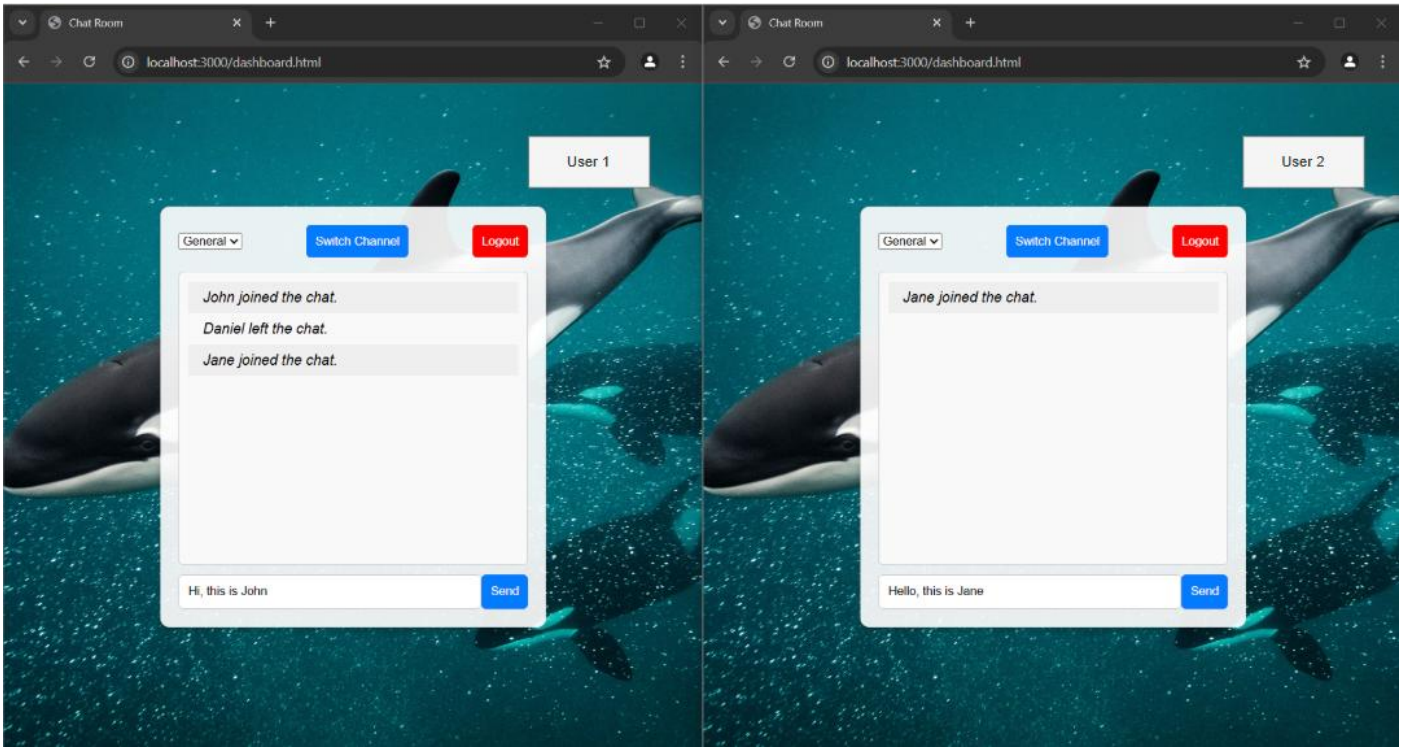
After sending messages



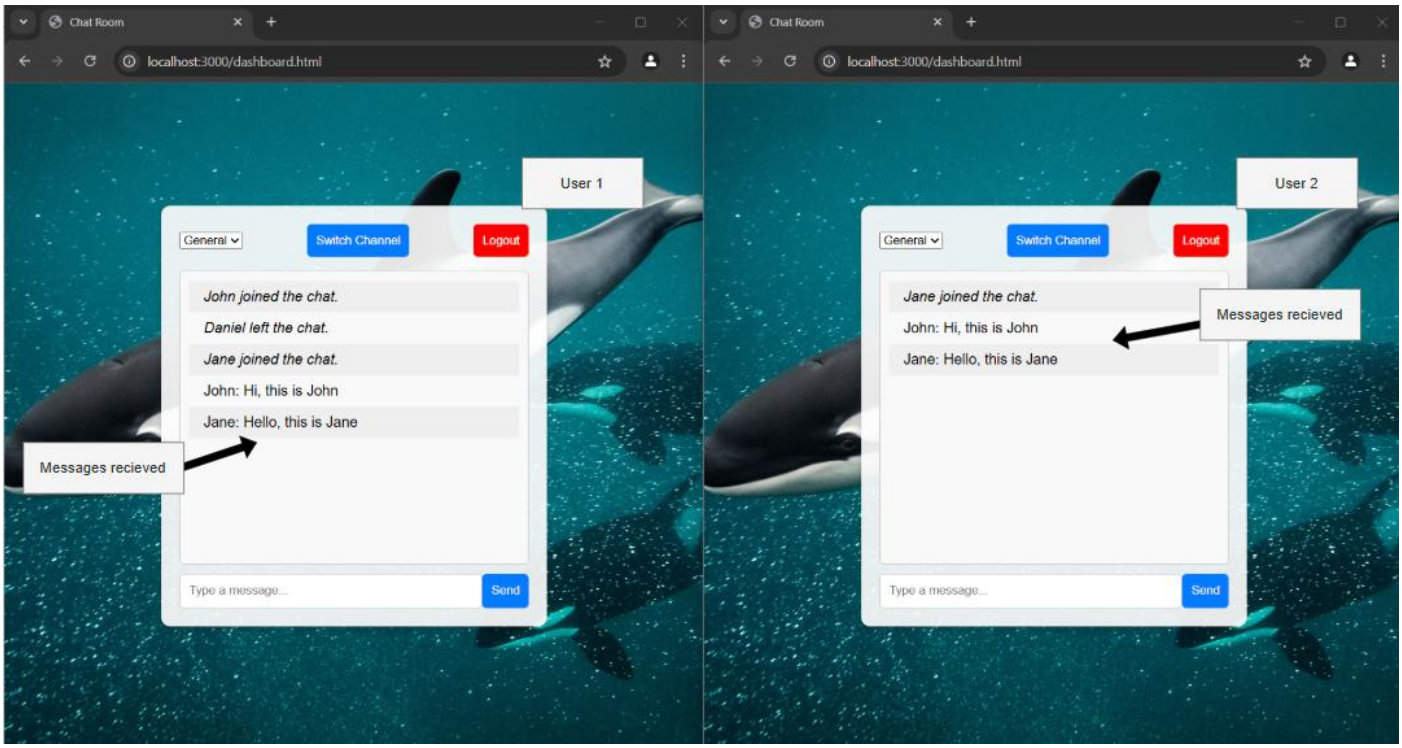
This test was a success, as all four of our users were able to send messages and receive messages from others. This shows that the message function works and that it's scalable to accommodate many users.

Test 3

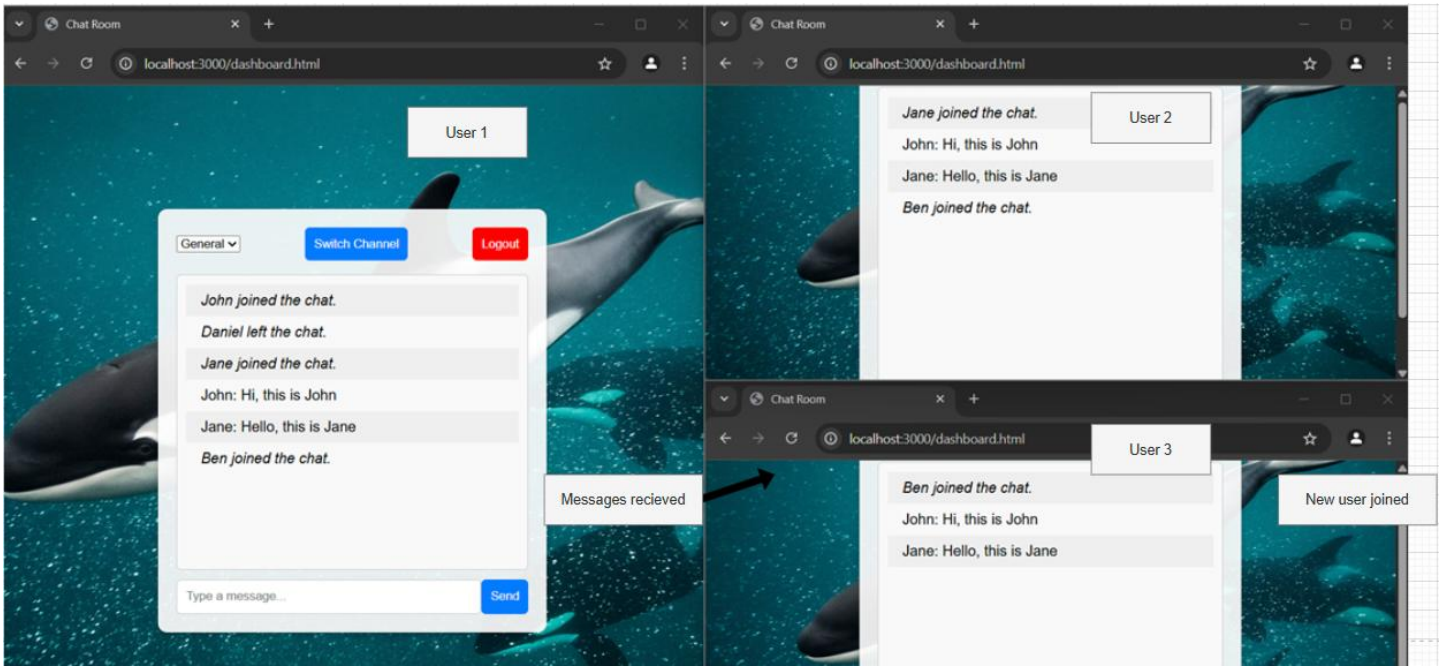
Before sending messges



After sending messages



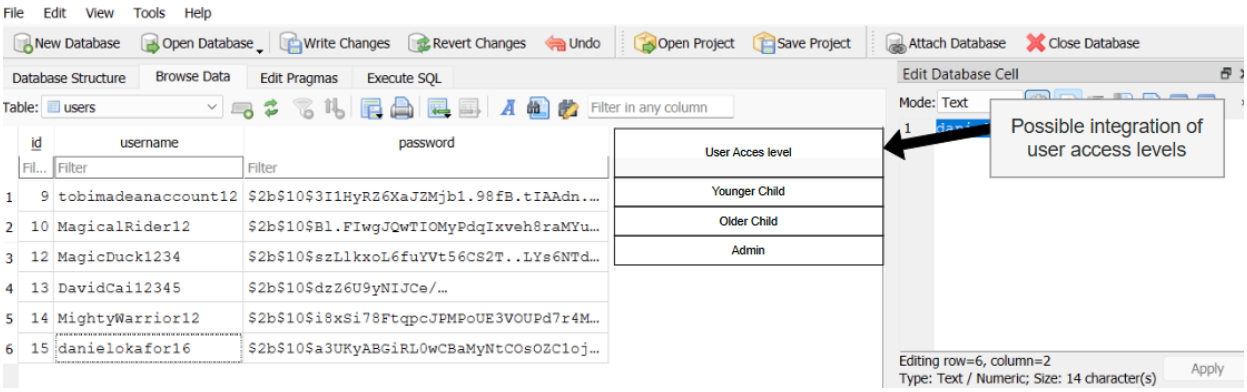
After a new user joins



This test was a success because the new user that joined receive all the old messages sent before the user was connected to the server. This means that the message recording function works, as user can retrieved message that have been sent in the past or when they weren't connected to the server.

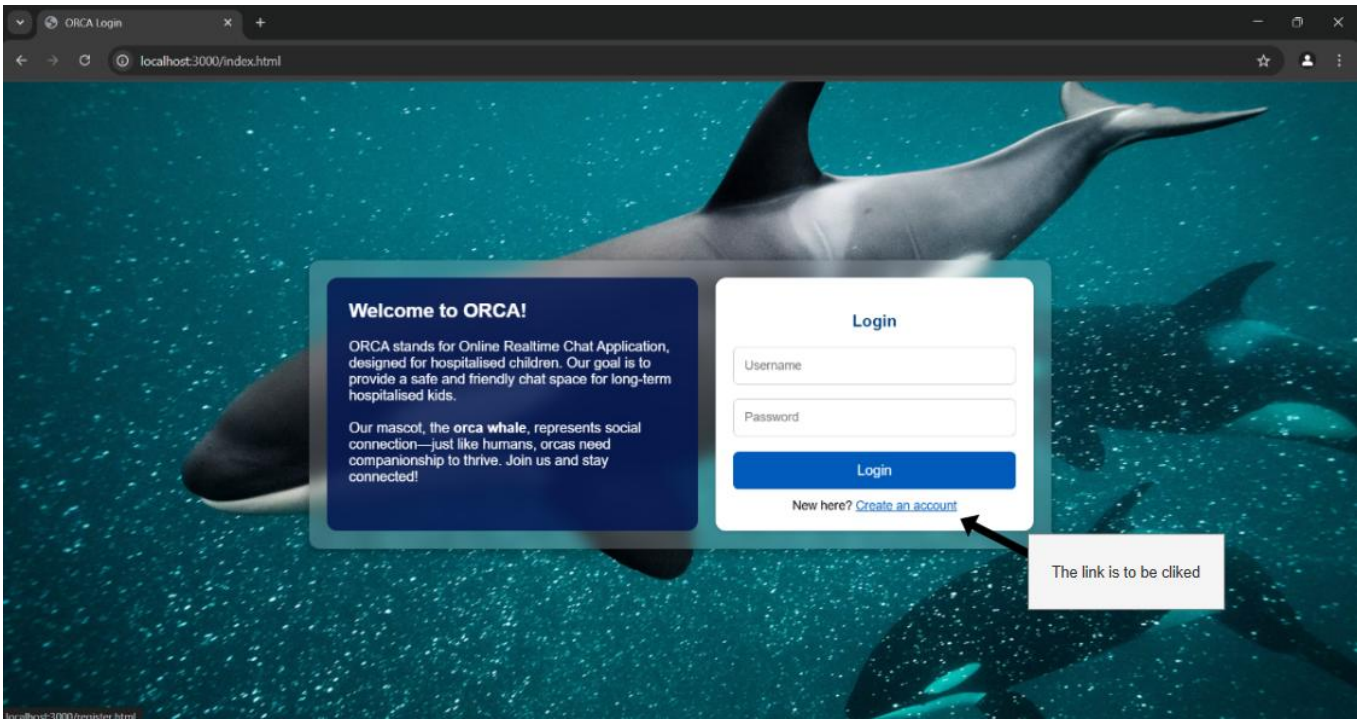
Test 4

However, due to time constraints, I was not able to implement a user access level to the authentication system. The user access levels would be used to determine what topic channels they could access. If they were an admin, they could be taken to a different user interface (the admin panel) with additional controls to moderate and manage the website. However, below, I create a possible solution of user access levels using the user database and storing their access levels. The server will be able to retrieve their access levels when they log in and render the corresponding user interface.

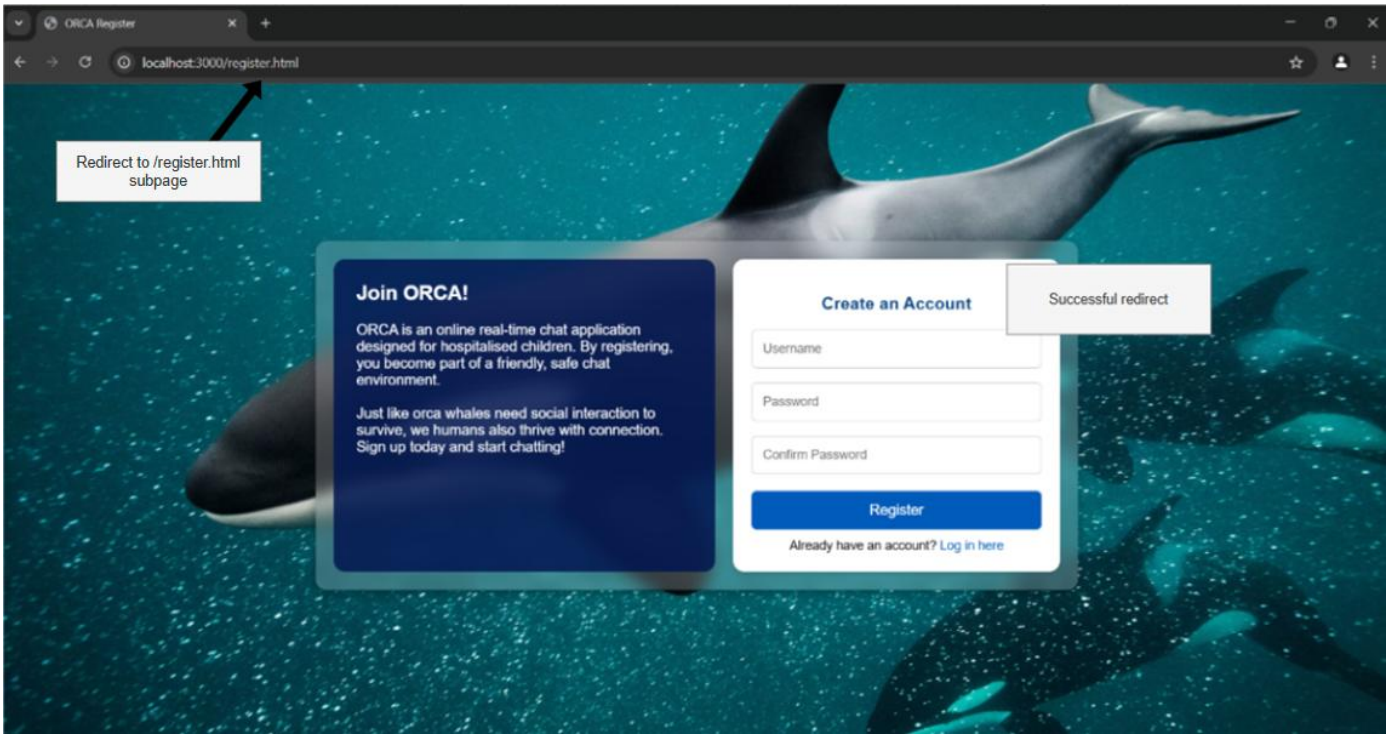


Test 5

Before clicking on link



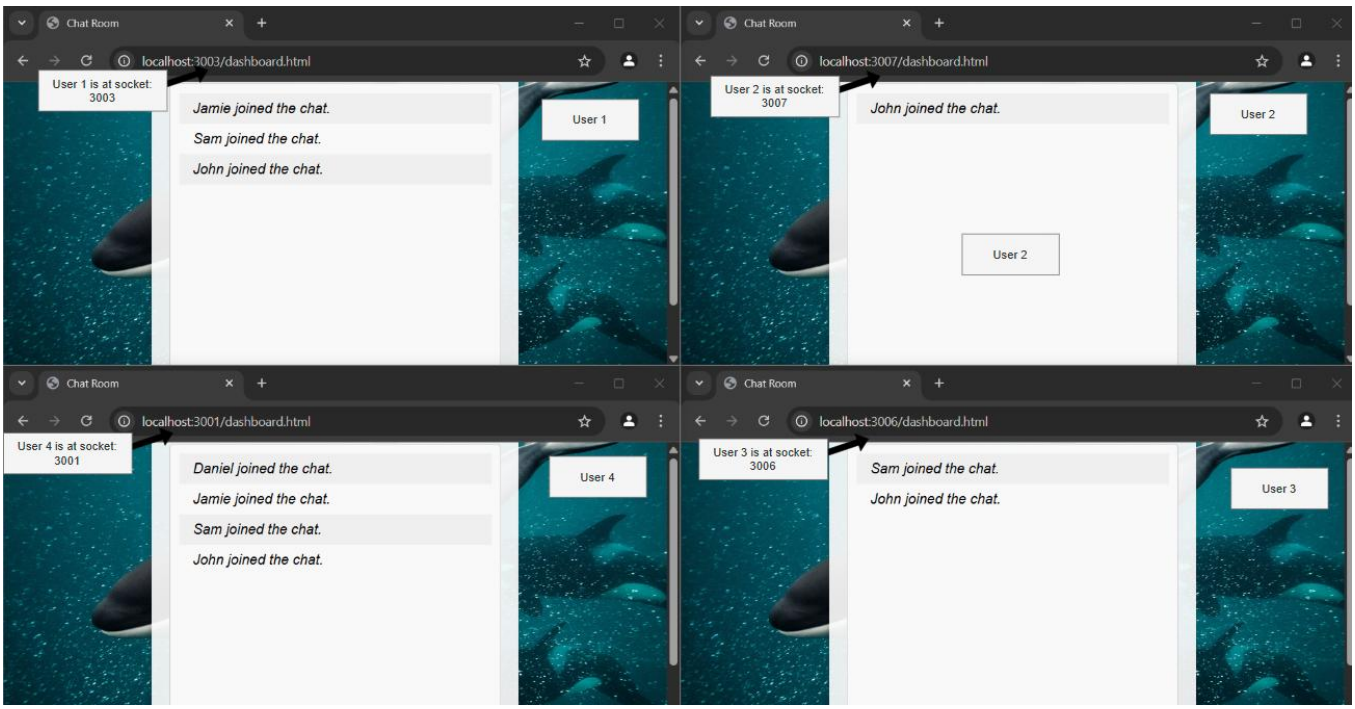
After clicking on link



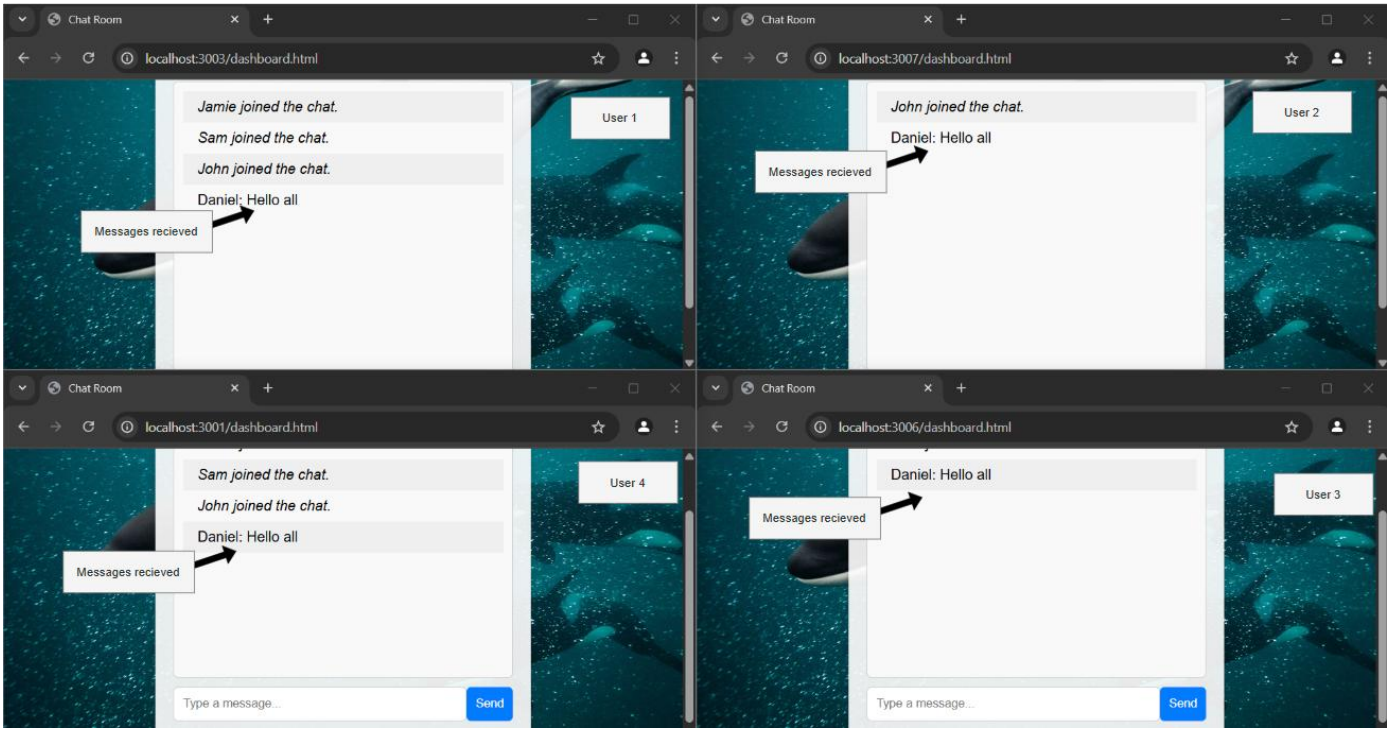
This test was successful because the user was redirected to the subpage /register.html, which is one of the public subpages of the website. Therefore, this proves that the links on the website will send the user to the correct corresponding subpage.

Test 6

Before sending the message



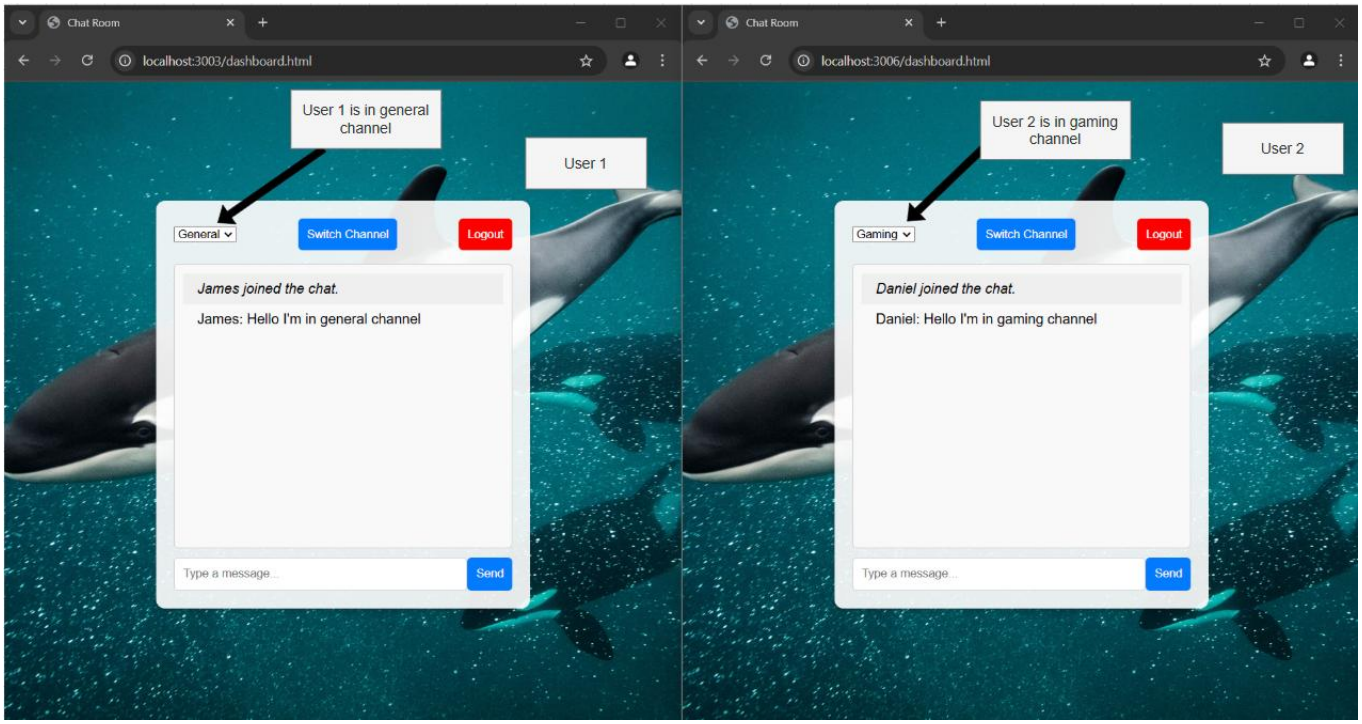
After sending the message



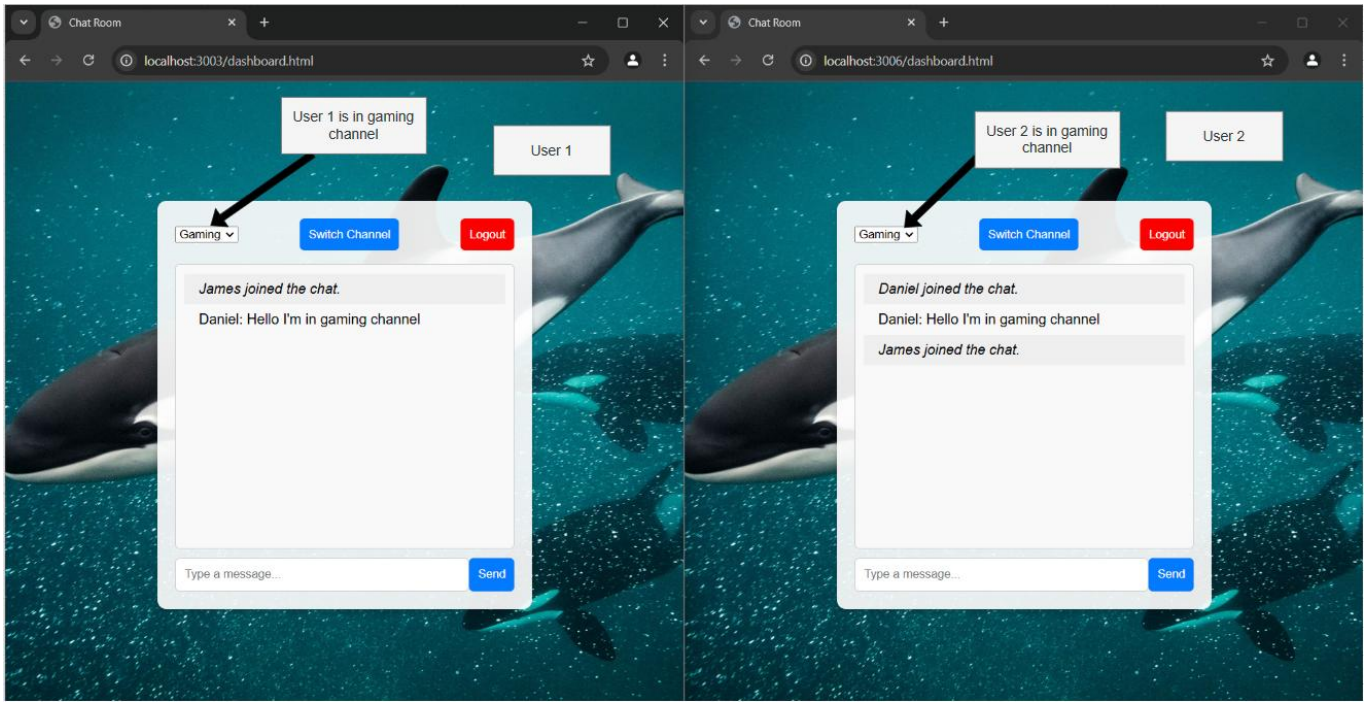
This test was a success because all the users were able to join on different sockets (3000-3007) while still all being able to receive messages from one another (in sync with the server). Therefore, the worker function of scaling the server to run on multiple sockets works.

Test 7

Before switching channels



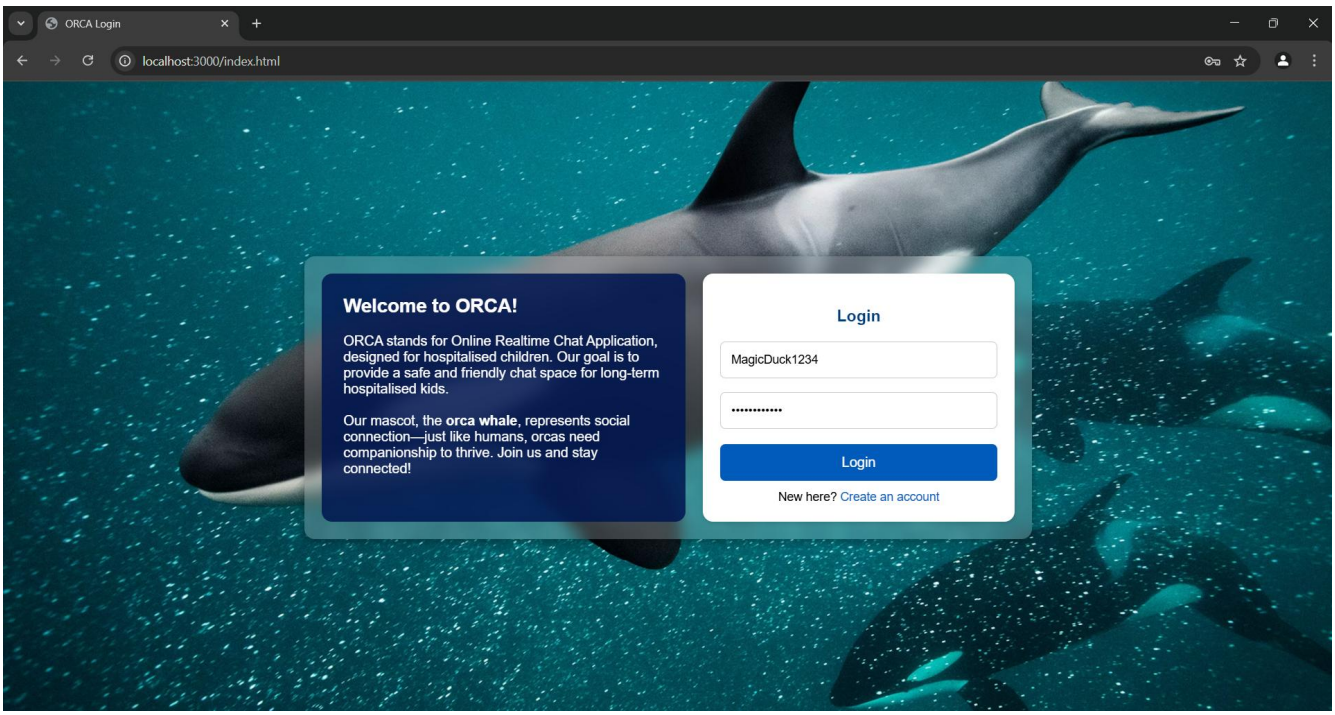
After switching channels



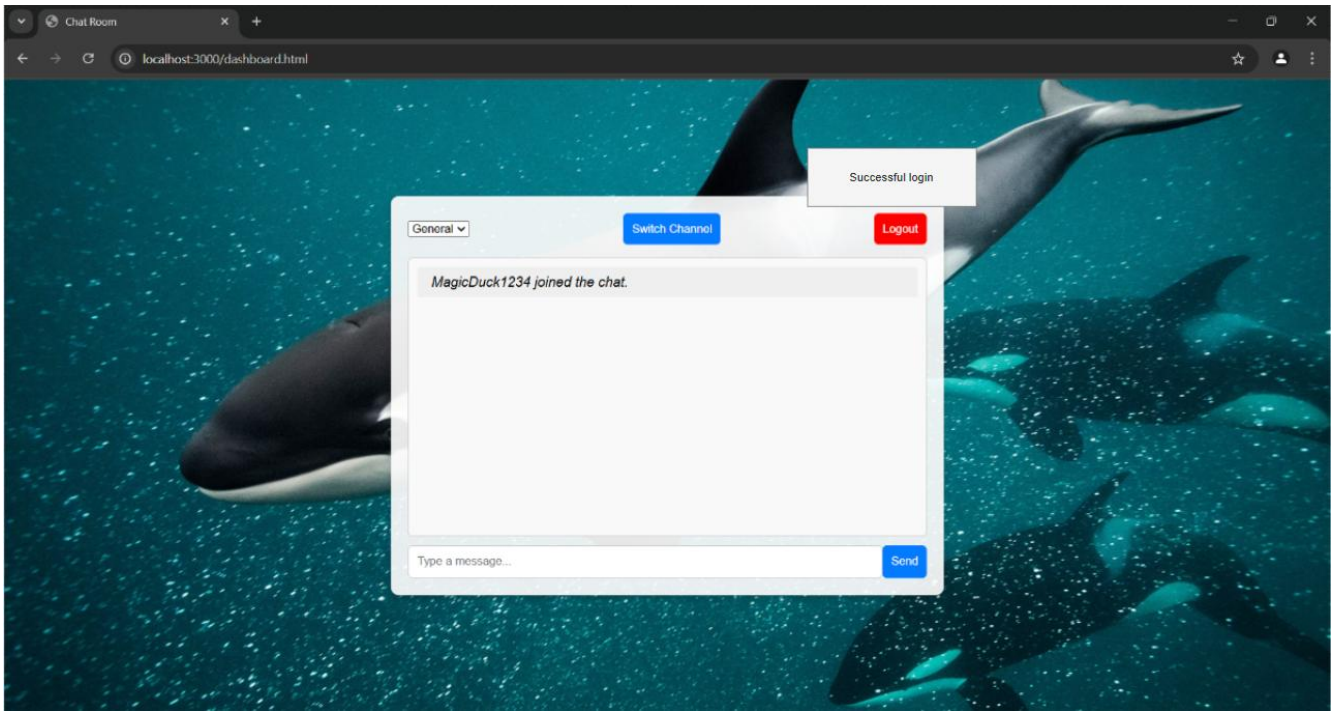
This test was successful because user 1 and user 2 were able to send messages in different channels. This means that messages can be sent over different channels simultaneously. Furthermore, when user 1 switched channels to the gaming channel (the same as user 2), user 1 was able to view the message sent by user 2. Therefore, this means that the channel switching function and the separate channel logic work.

Test 8

Before login



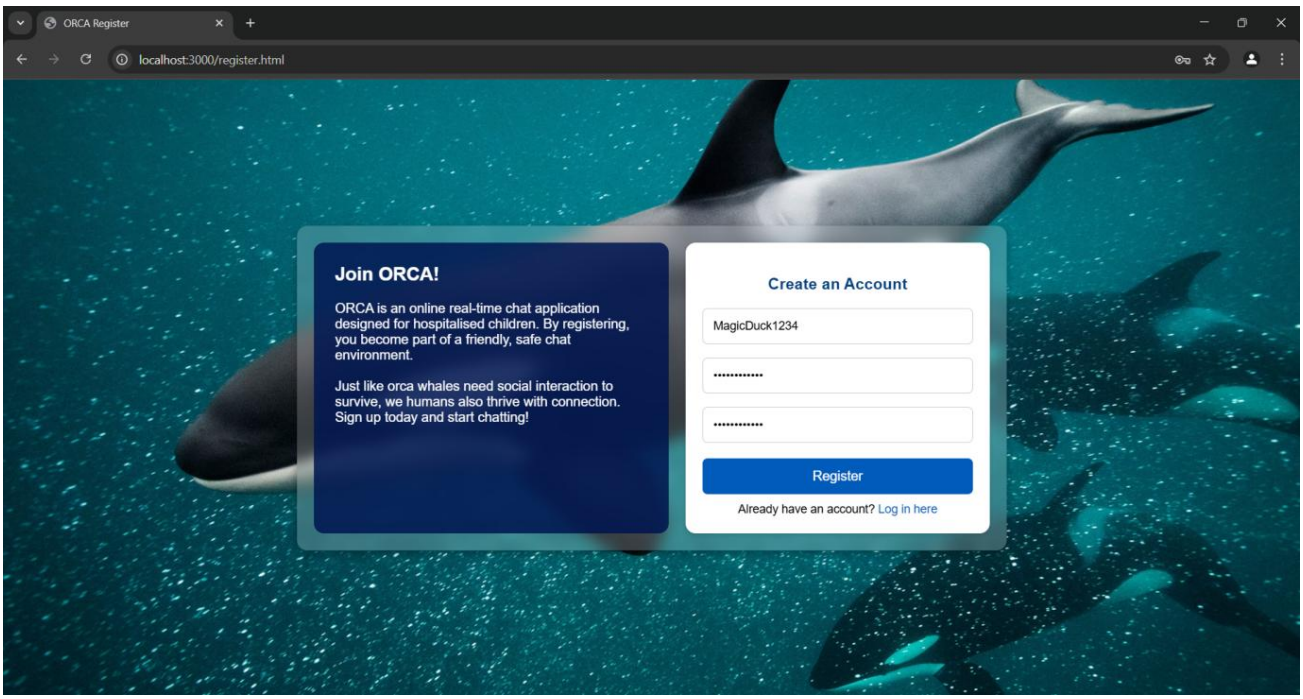
After login



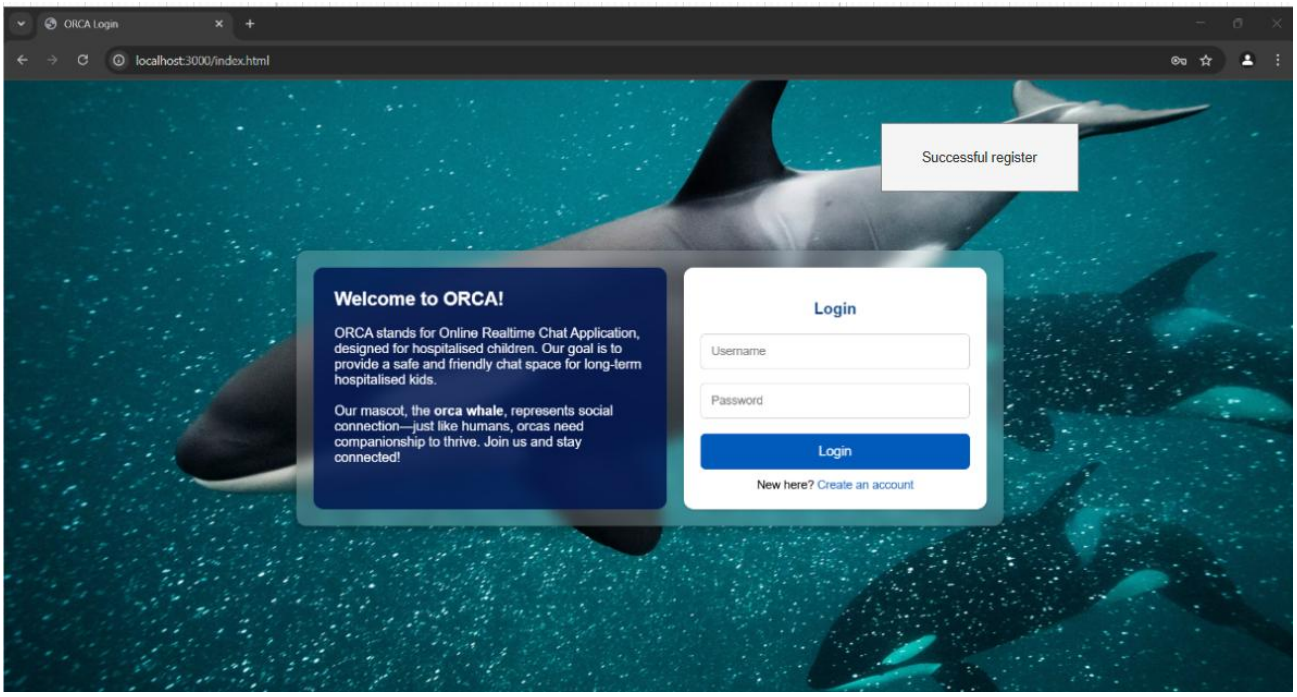
This test was successful because the user was able to enter valid login credentials and enter the website (The chat panel). This demonstrates that the login system works as the website can verify the username and password of the user using the user database (where the user credentials are stored) and then successfully redirect them to the chat panel now that they have been authenticated.

Test 9

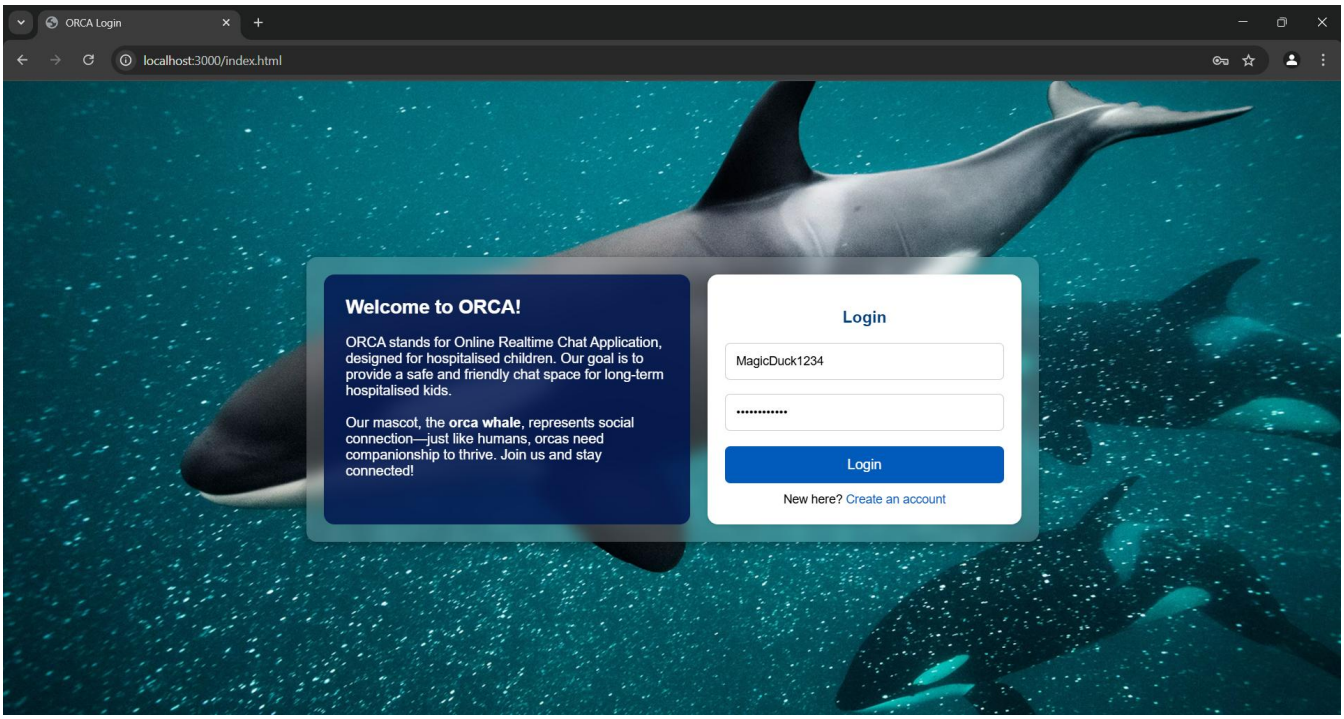
Before entering credentials



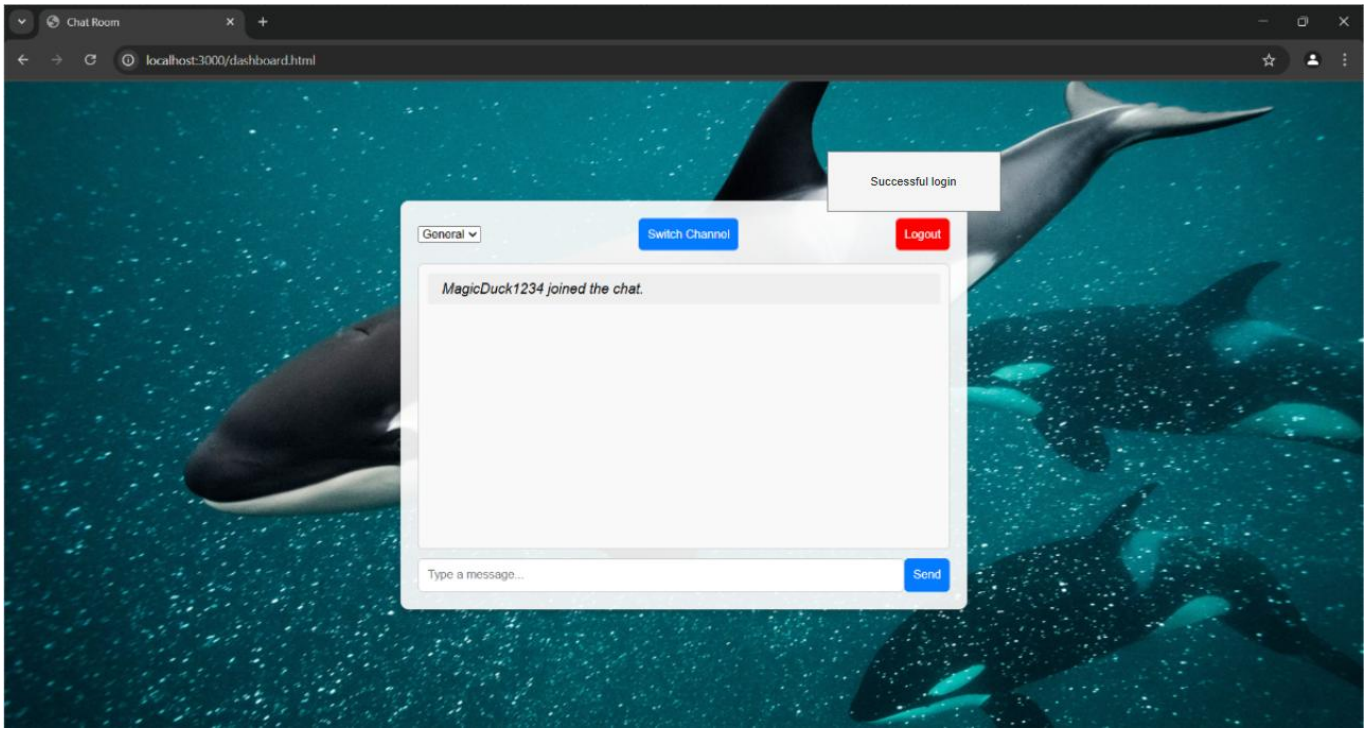
After entering credentials



Before login



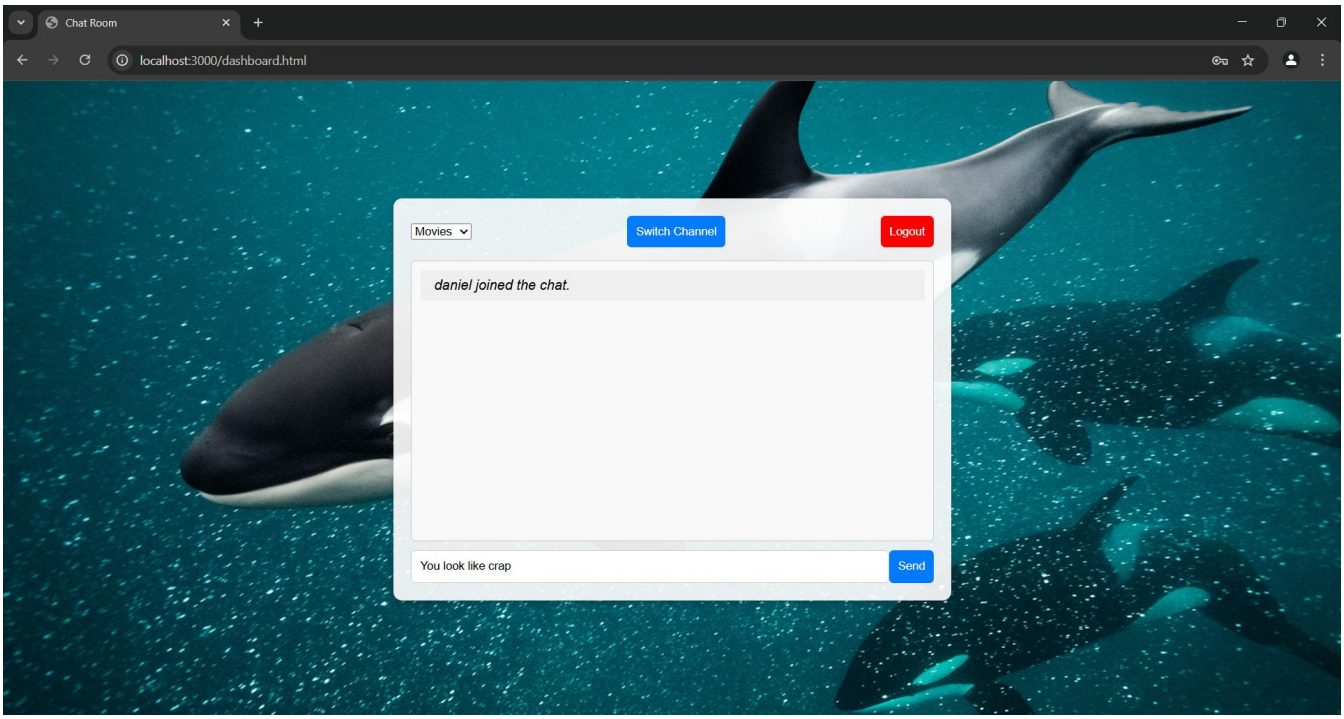
After login



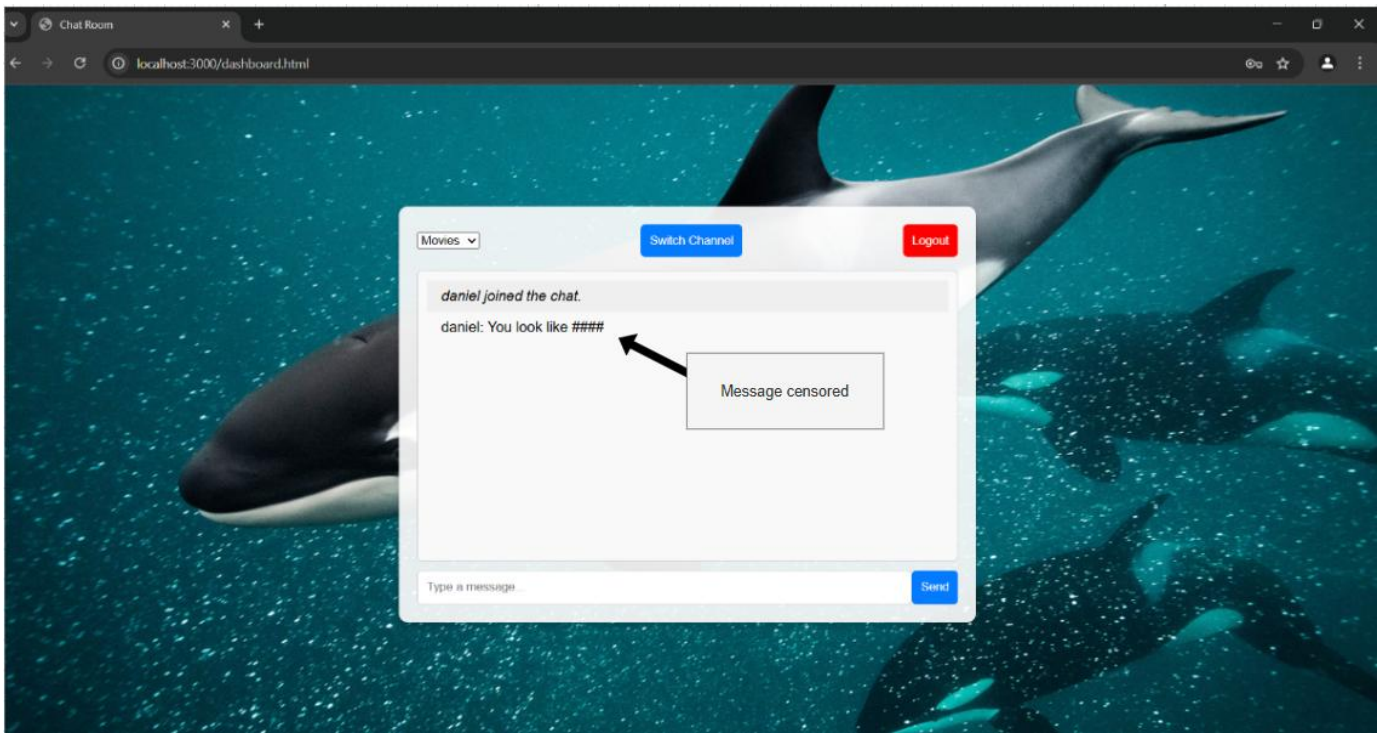
This test was successful because the user was able to create a valid account (password at least 8 characters long; the username be at least 12 characters long; the username must contain at least two numbers; the username cannot already be in the user database). This means that the website can update the user database and insert a new user. Moreover, the server can also retrieve the user's credentials to verify them against the login information. Proving that both authentication and registration systems work.

Test 10

Before sending message



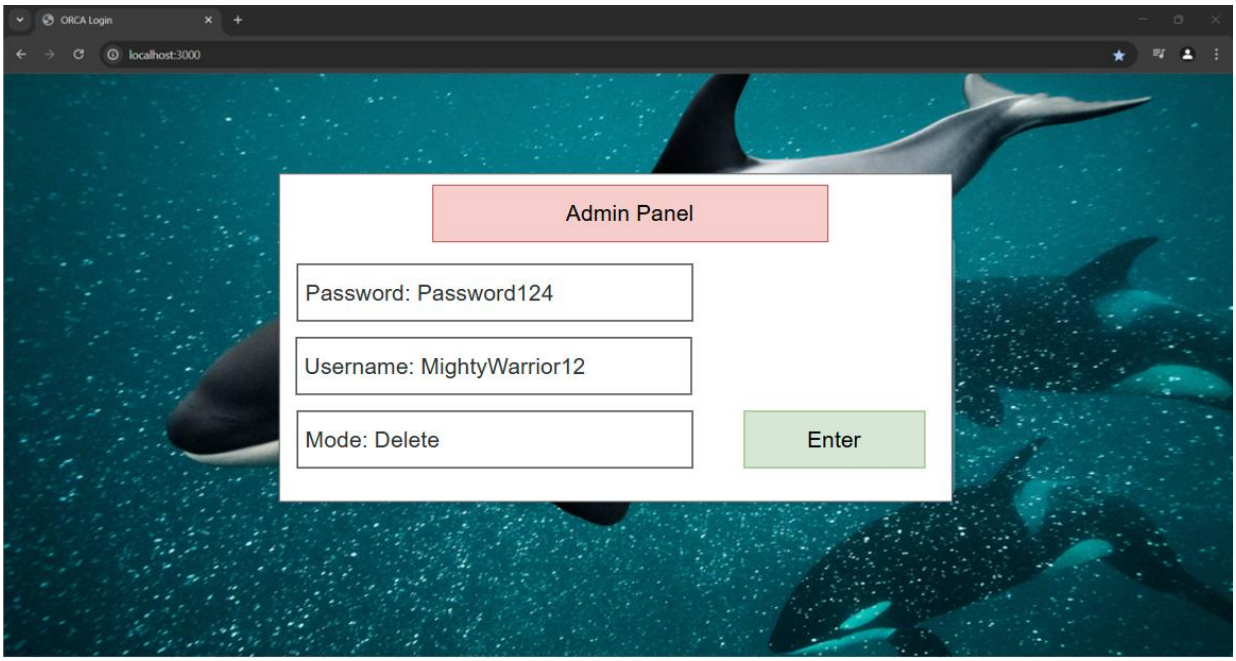
After sending message



This test was successful because the user could not enter a banned word into the chat panel without it being censored. The censored word (crap) was replaced with '#', censoring the word. This proves that the censoring function of the website works, therefore ensuring the safeguarding of users on the website.

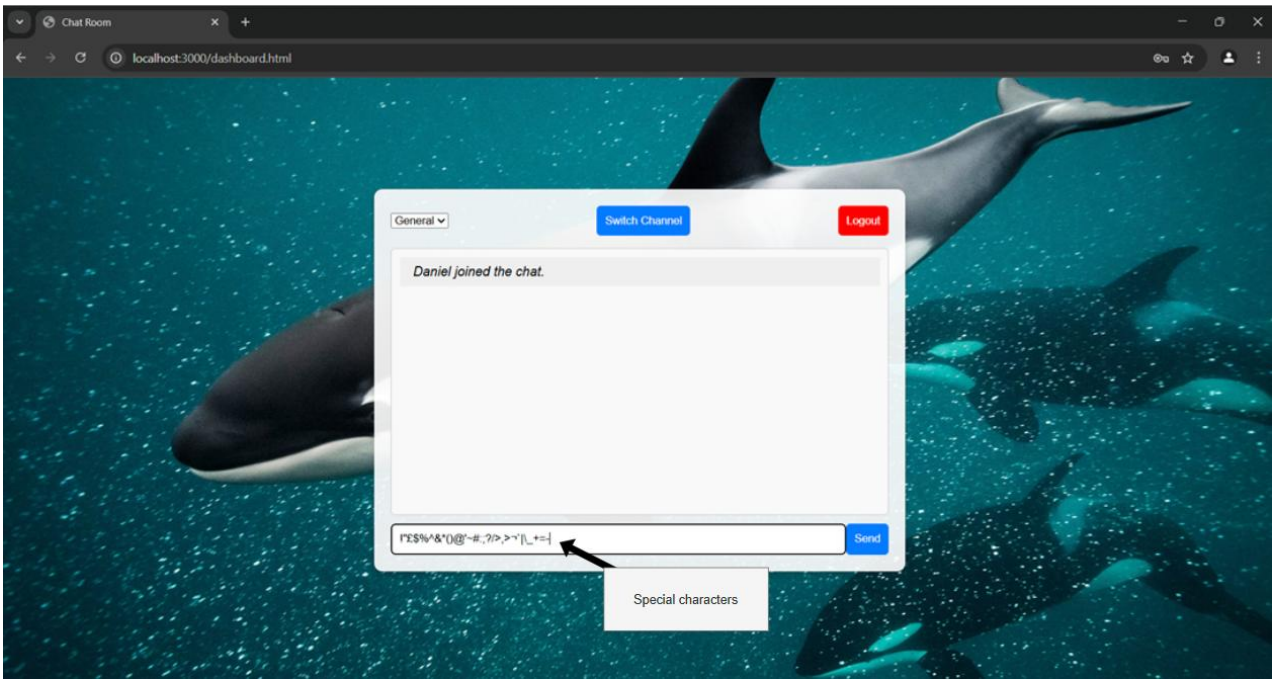
Test 11

However, due to time constraints I couldn't implement an admin panel. However, in future development I would implement a system in where an admin will receive a different user interface from the other users. The admin will have access to the admin panel below where the admin can edit the user database, by changing the mode – insert, delete, alter etc.

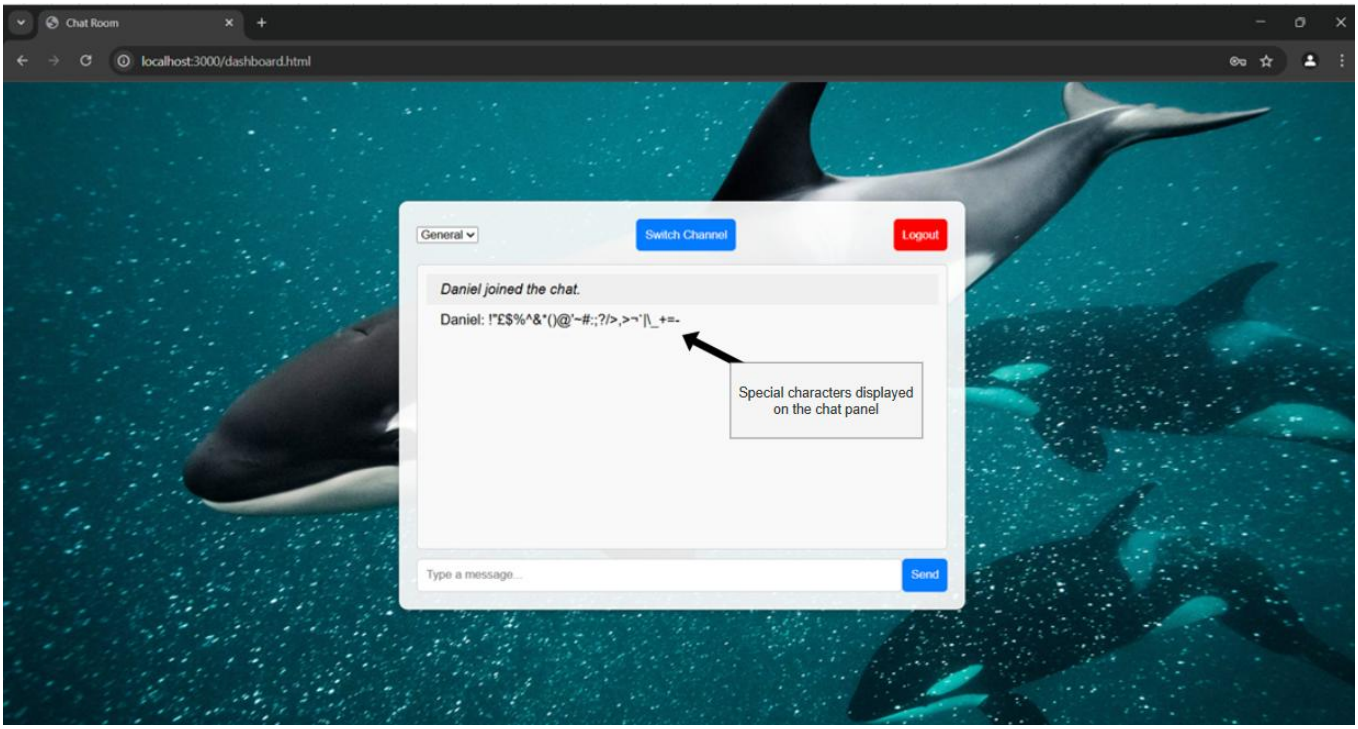


Test 12

Before sending message



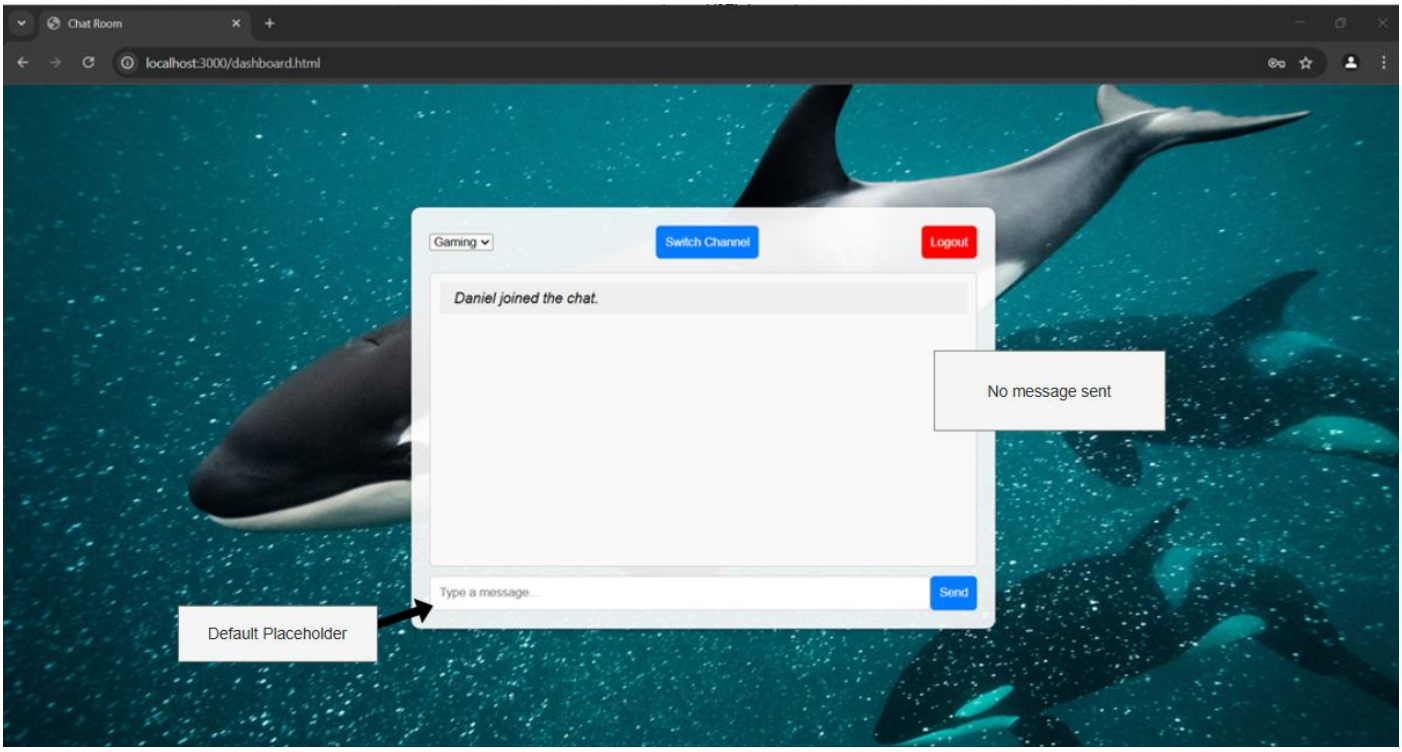
After sending message



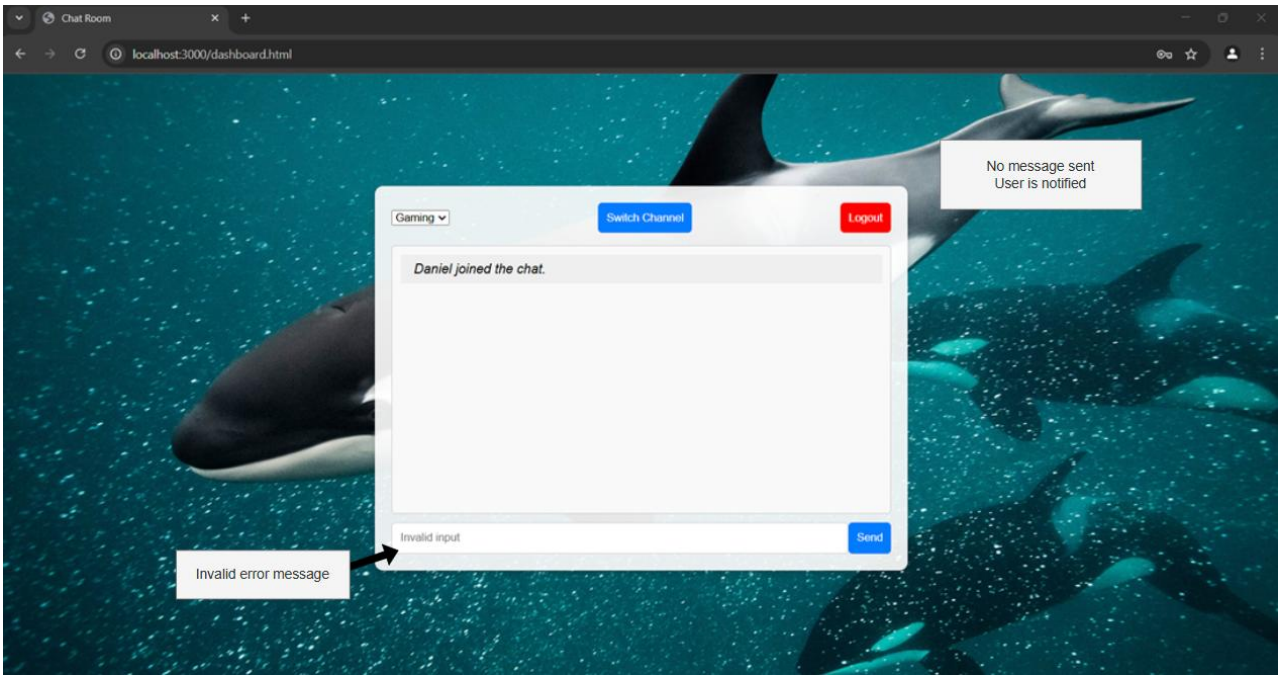
The test was successful because the user was allowed to enter special characters into the chat panel. This is important for our stakeholder as the use of special characters can be used to add emotion to messages. This means that communication over the website can be more natural and inviting. Overall, this test proves that the real-time chat functionality works even for boundary data.

Test 13

Before sending empty message



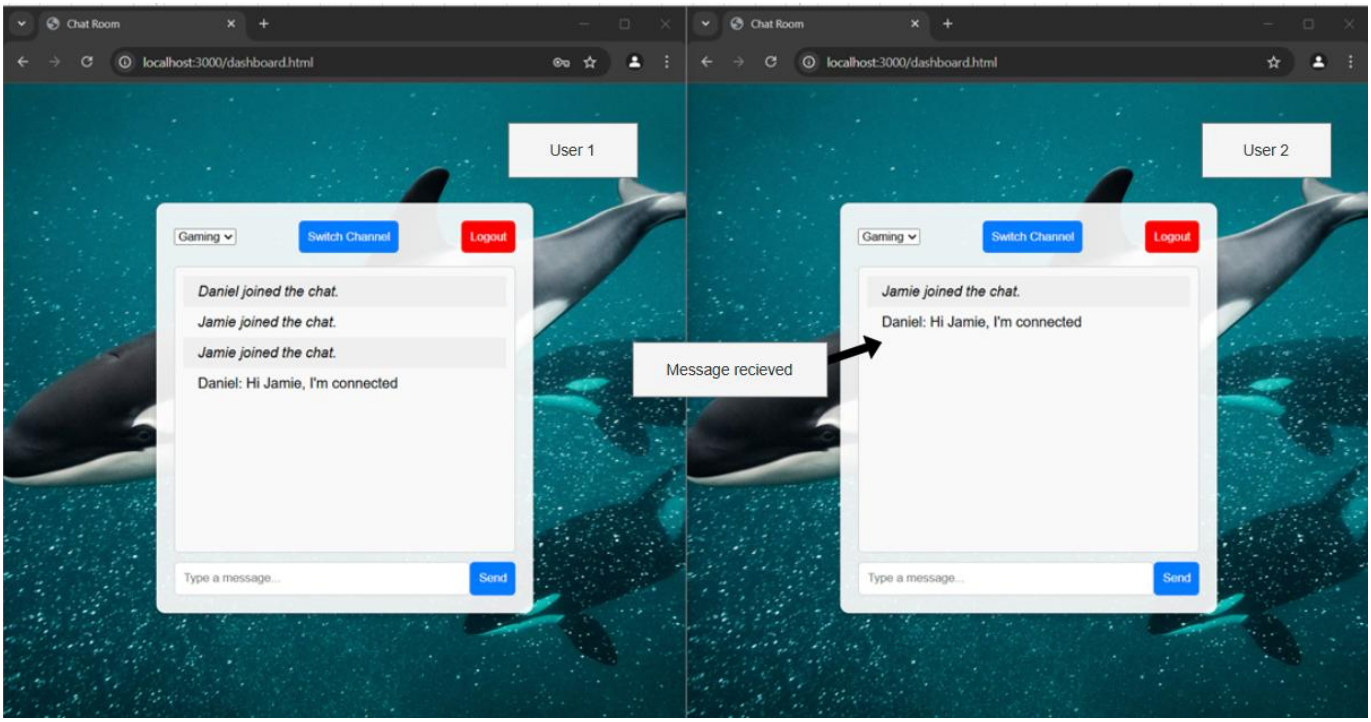
After sending an empty message



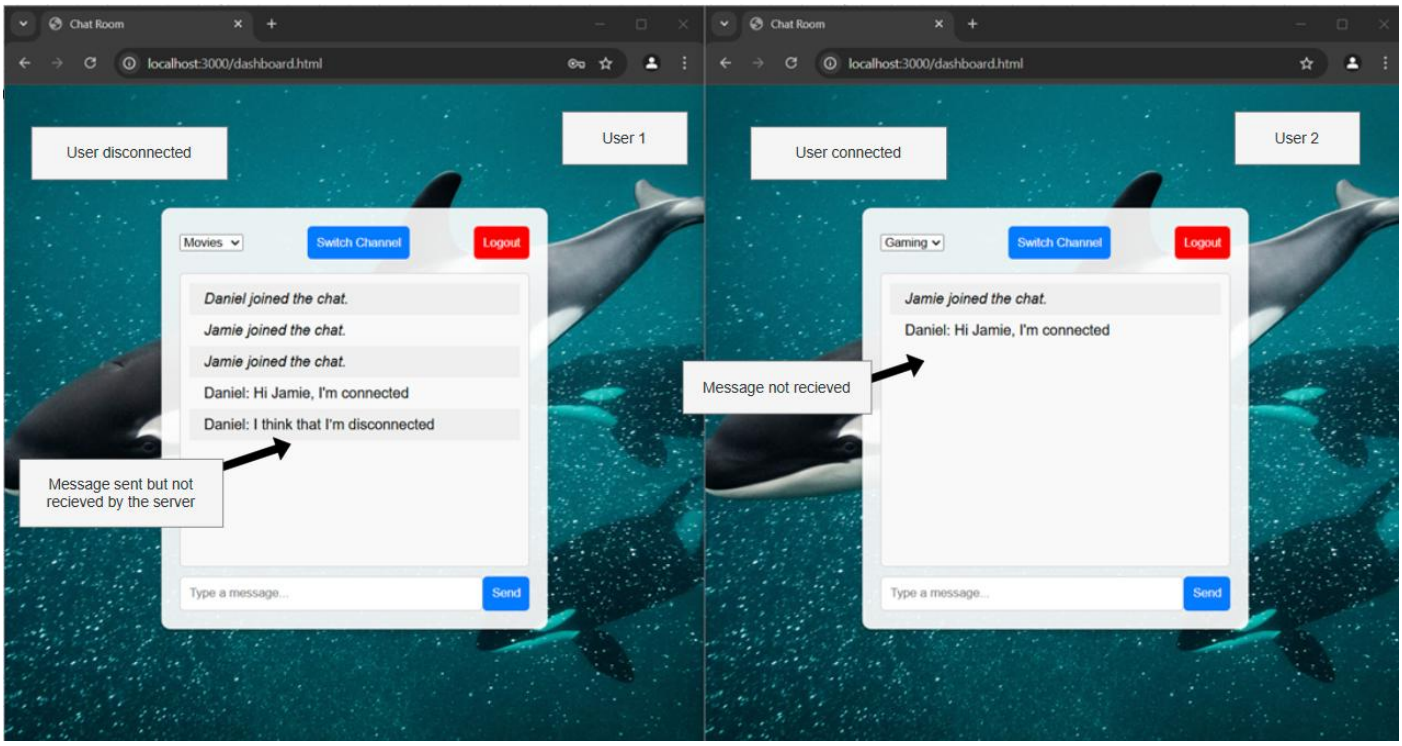
This test was successful because the user could not send an empty message, instead, they were notified that their input was invalid. This is a very subtle and non-invasive way to notify the user, as empty messages create clutter. Therefore, this test proves that invalid data is not accepted when chatting using the chat panel.

Test 14

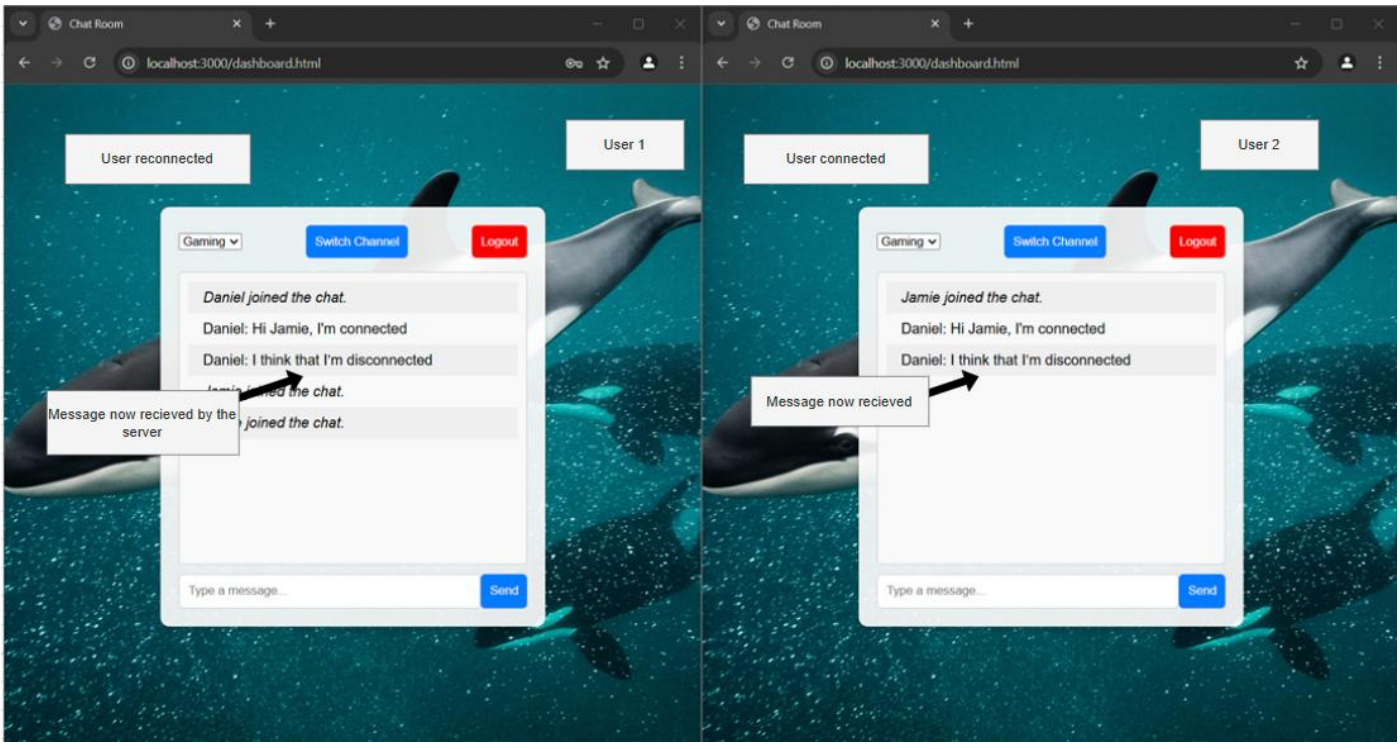
Before disconnecting



After disconnecting



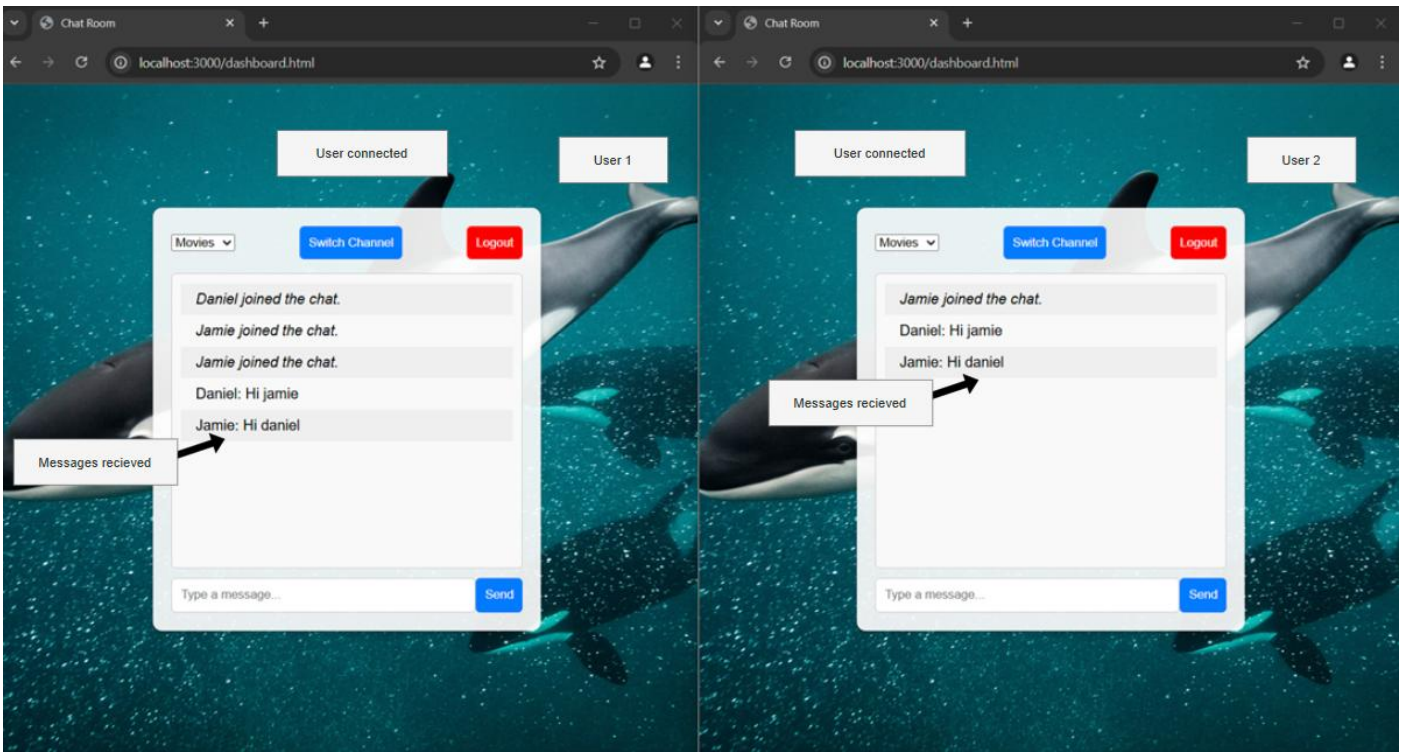
After reconnecting



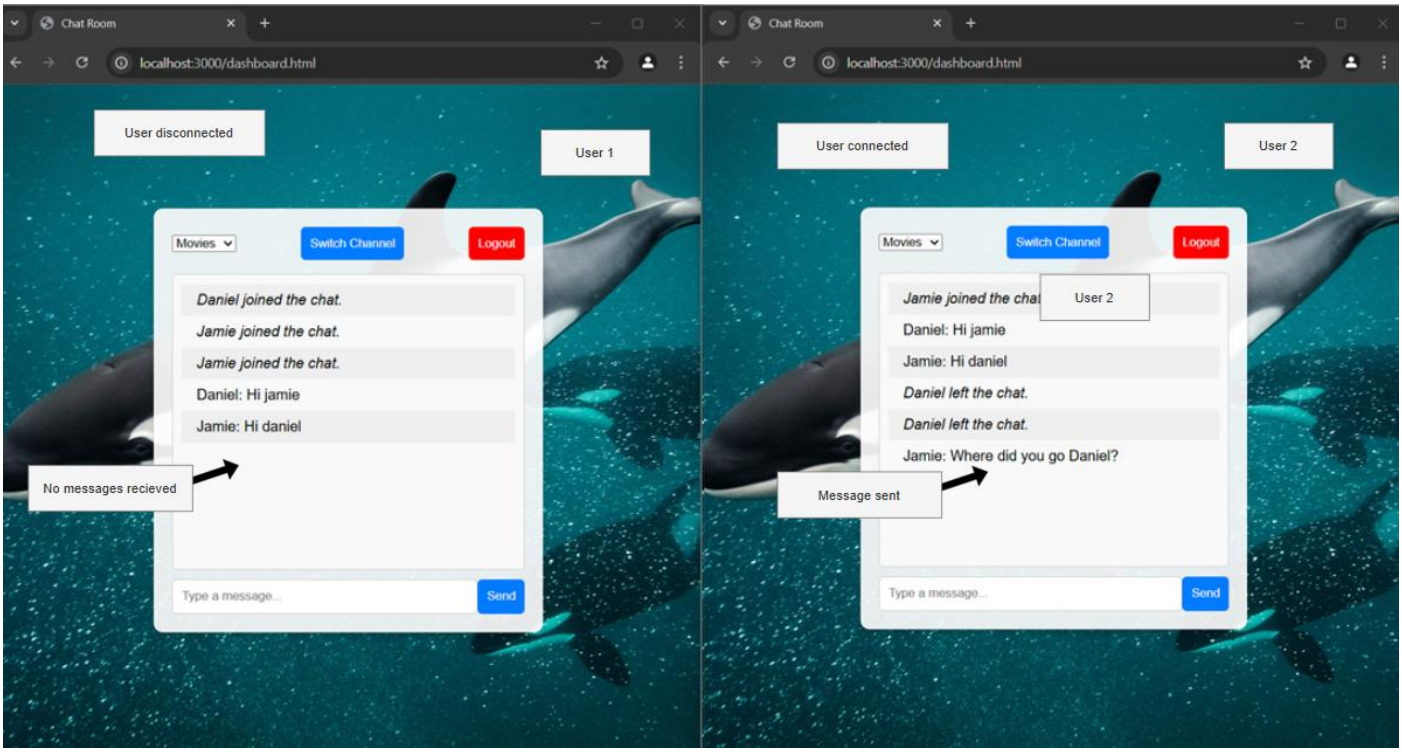
This test was successful because once the user reconnected back to the server, their message was sent even though they were disconnected when they sent the message. This is useful because it ensures the ease of communication on the website, as messages that are sent while disconnected will be sent on reconnection. This test proves the robustness of the real-time chat functionality and how it overcomes problems like users disconnecting.

Test 15

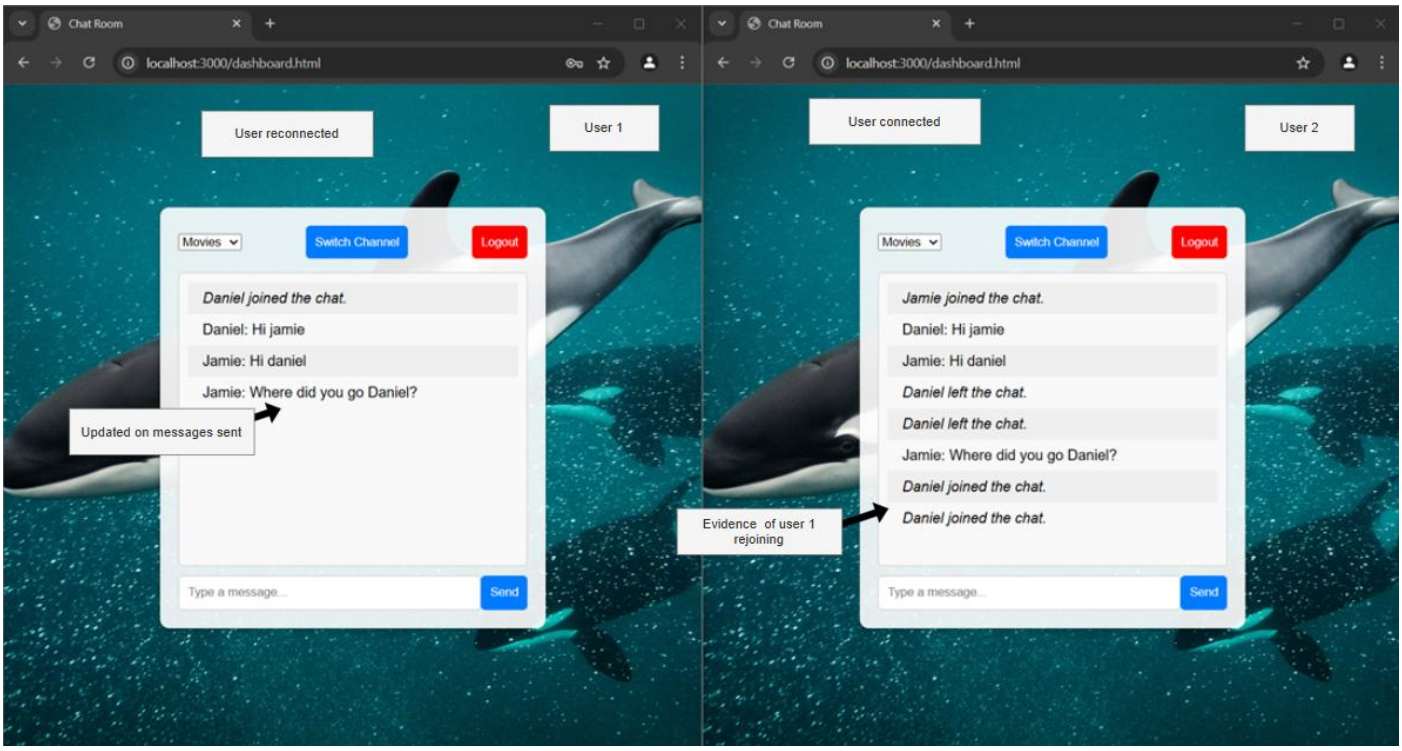
Before disconnecting user 1



After disconnecting user 1



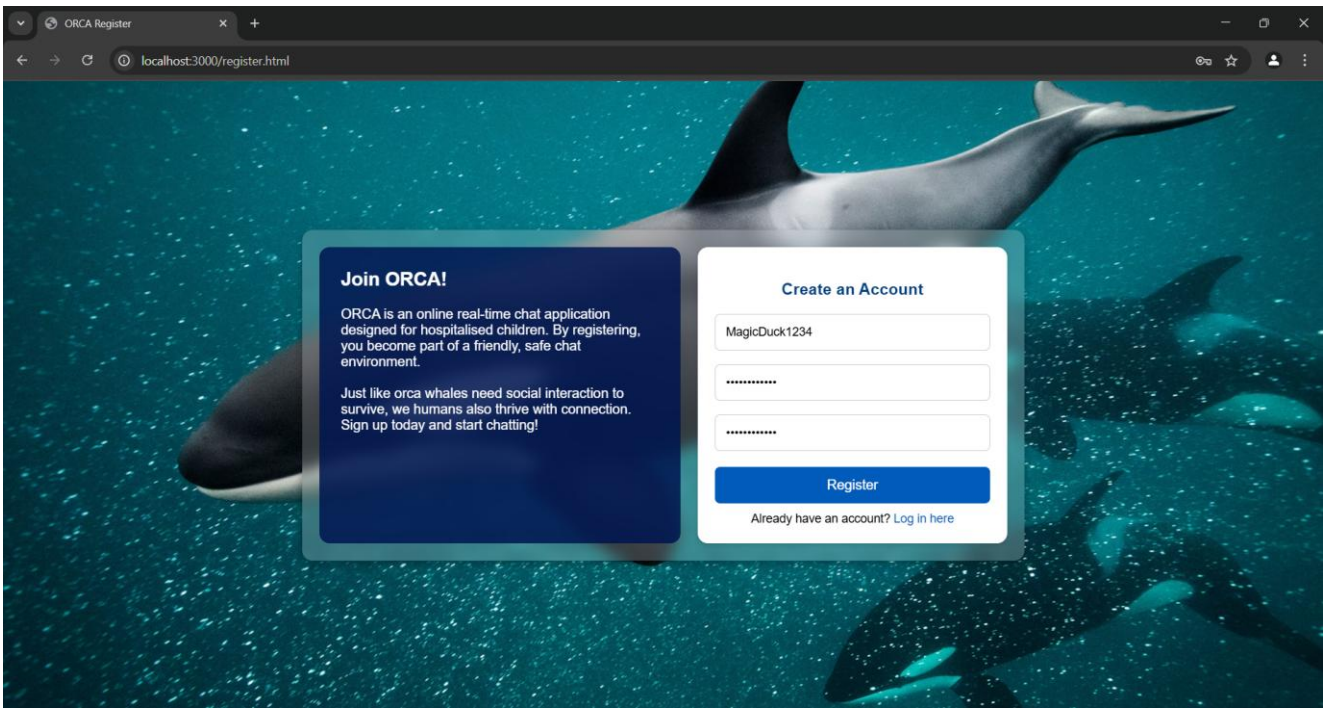
After user 1 reconnected



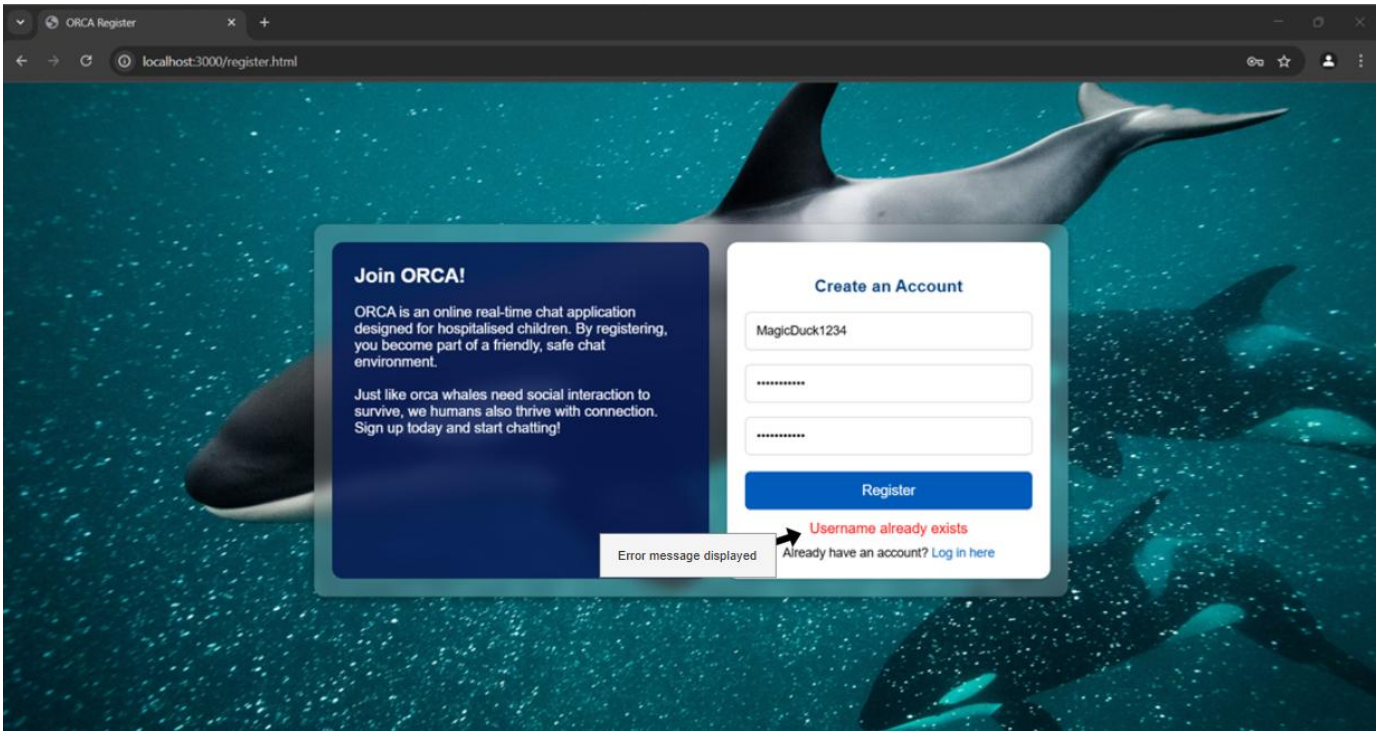
This test was a success because the user was able to rejoin the server and receive all messages sent while they were disconnected. The server updates user 1 on all messages missed once they rejoin. This proves that robustness of the chat functionality as even when a user disconnects, they will be updated on messages sent while they were disconnected. Overall, this improves the ease of communication over the website as users with different bandwidths or internet stability can still interact over the website.

Test 16

Before entering credentials



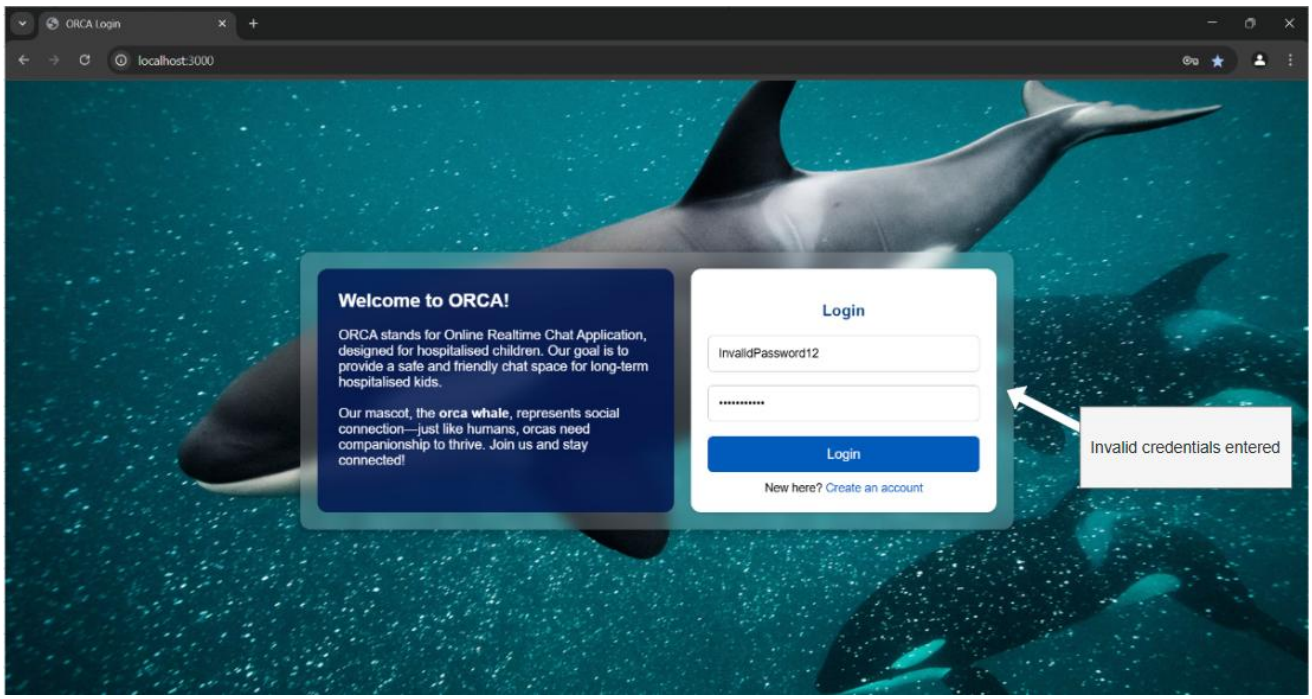
After entering credentials



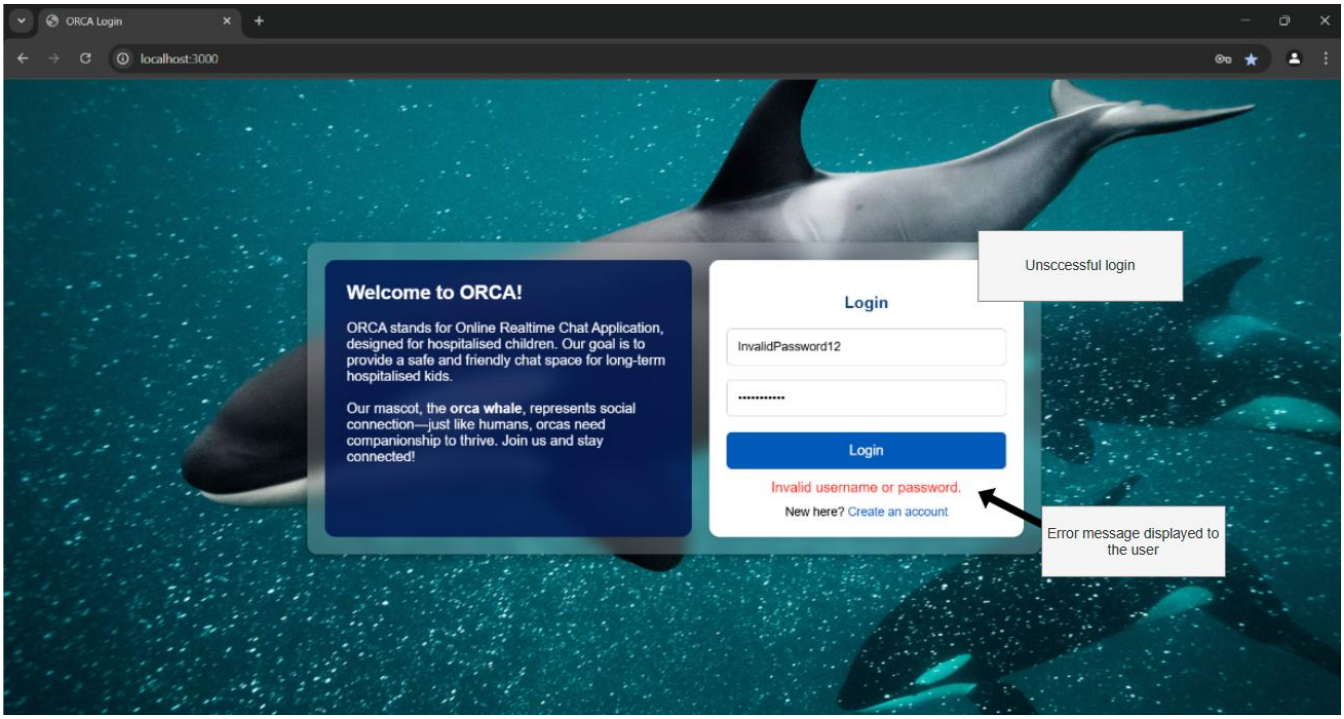
This test was successful because the user could not make an account with a username that is already taken. This is useful because this means that every username is unique, which will make it easier for users to identify other users. Furthermore, this demonstrates that the website will reject invalid credentials, which will help maintain the integrity of the user database by preventing duplicate fields.

Test 17

Before login with invalid credentials



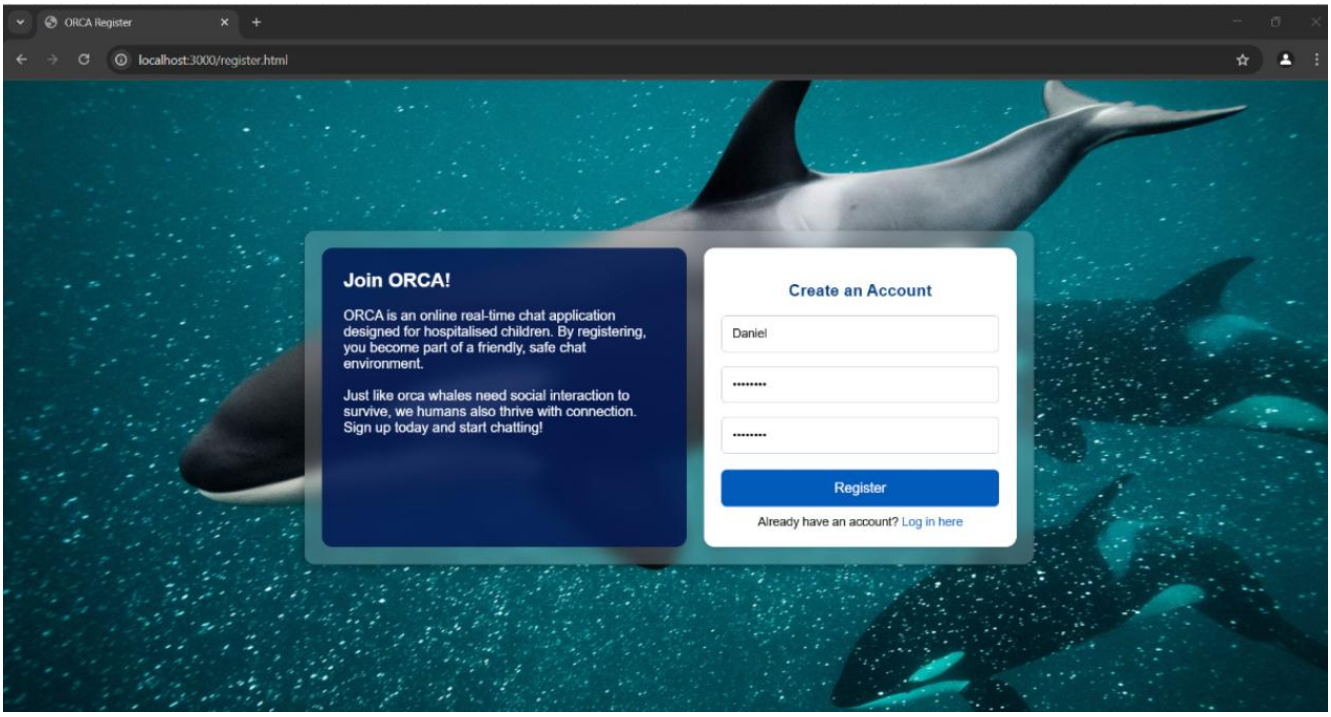
After login with invalid credentials



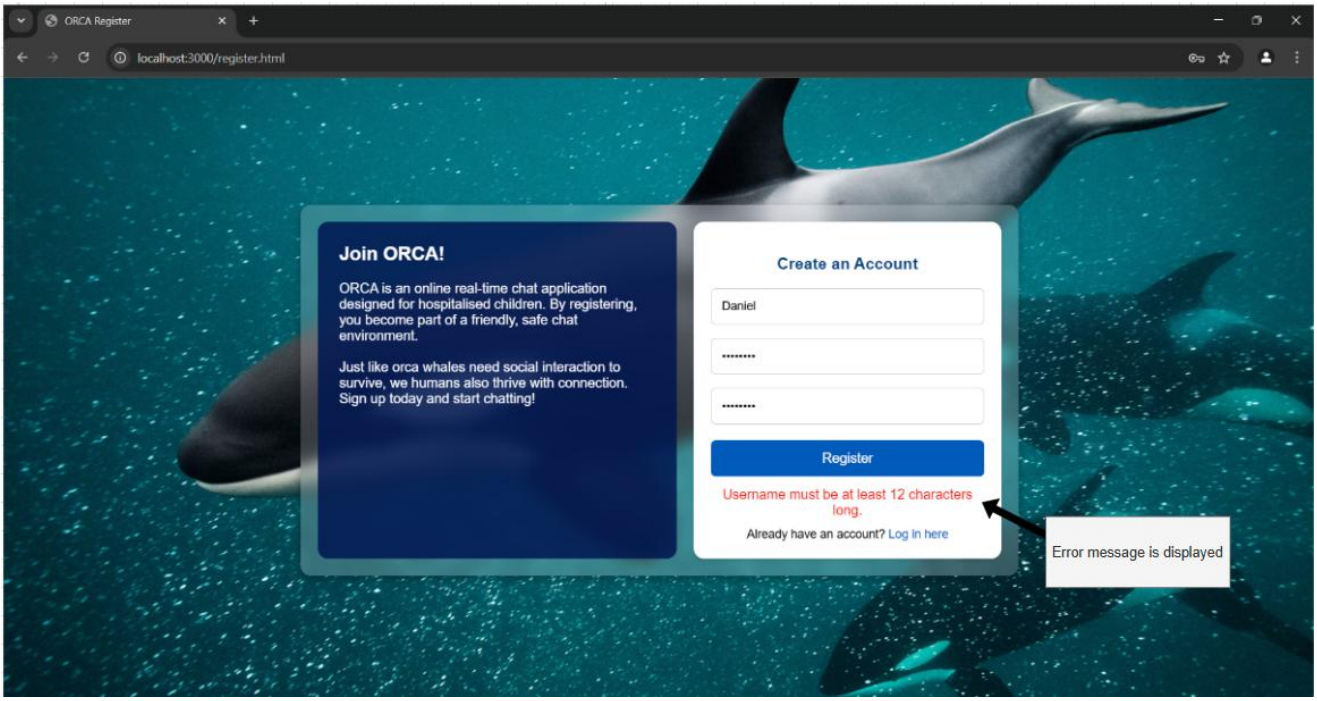
This test was successful as the credentials used in this test were rejected by the system. The invalid credentials used were not in the user database where the login credentials are stored. Therefore, this means that the website verifies the password and the username successfully and outputs an appropriate error message.

Test 18

Before entering invalid credentials



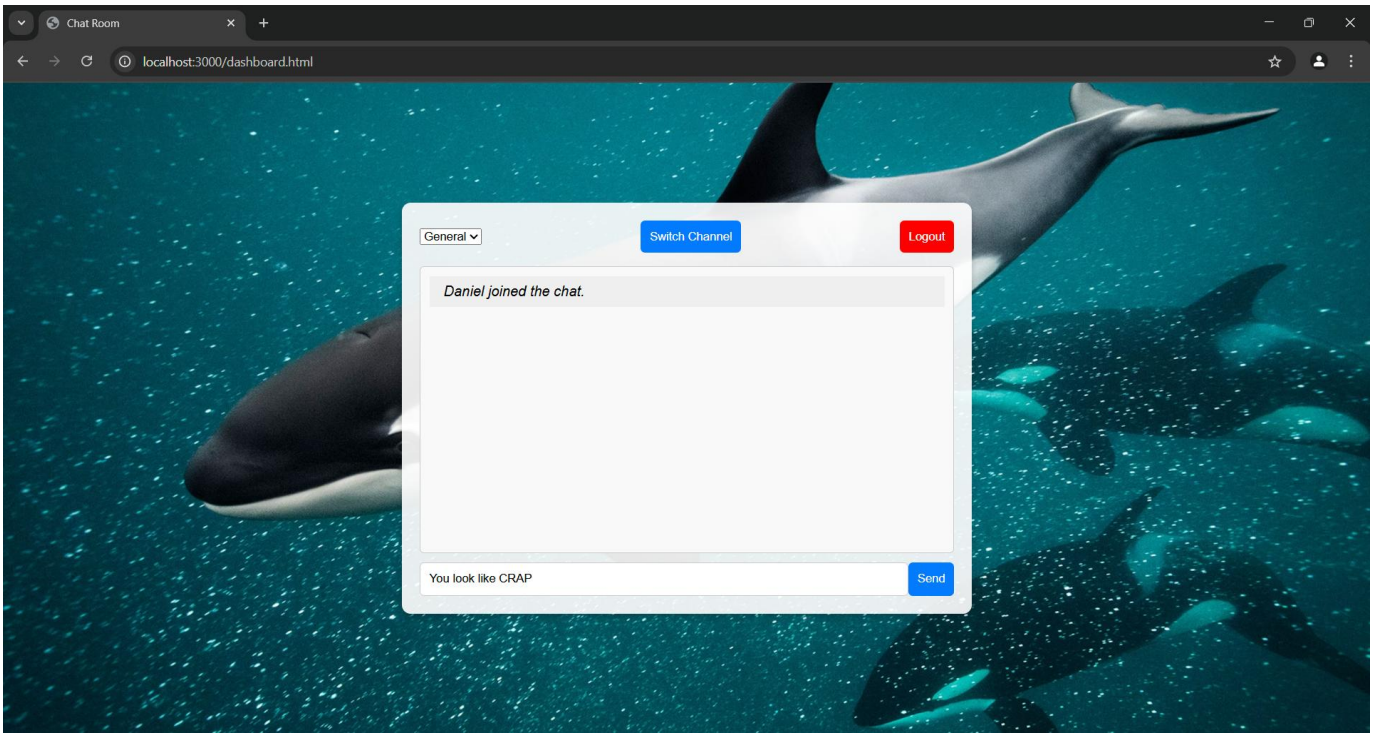
After entering invalid credentials



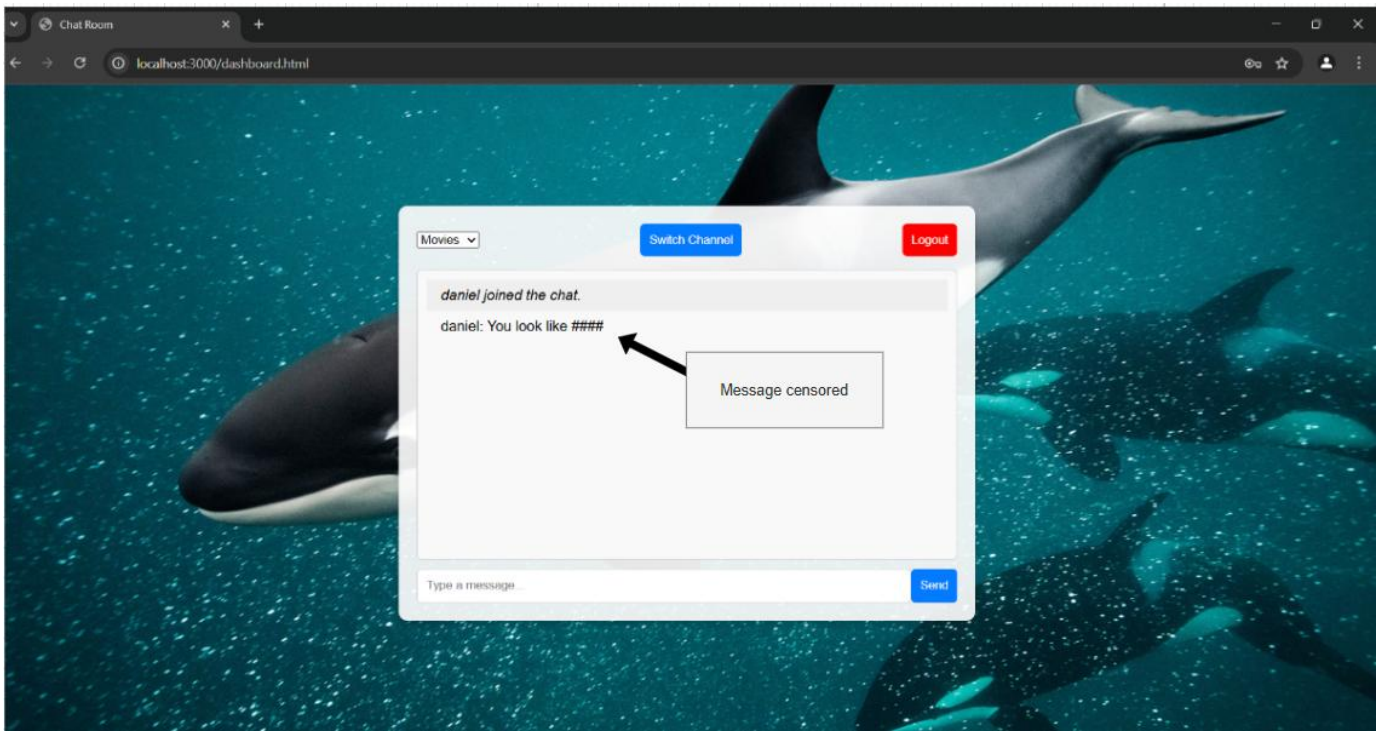
This test was successful because the user could not make an account that had invalid credentials. This means that the credentials did not reach the requirements of; password at least 8 characters long; the username be at least 12 characters long; the username must contain at least two numbers; the username cannot already be in the user database. This proves that the website rejects invalid data, protecting the integrity of the user database.

Test 19

Before sending message



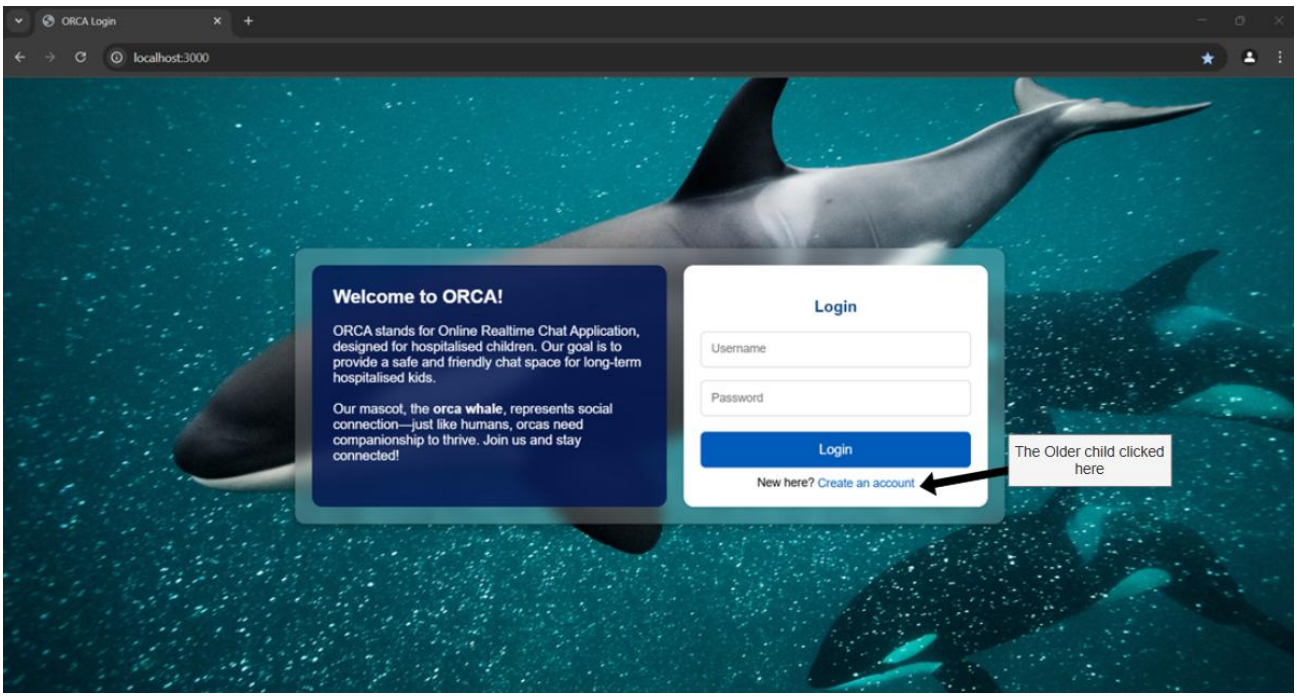
After sending message



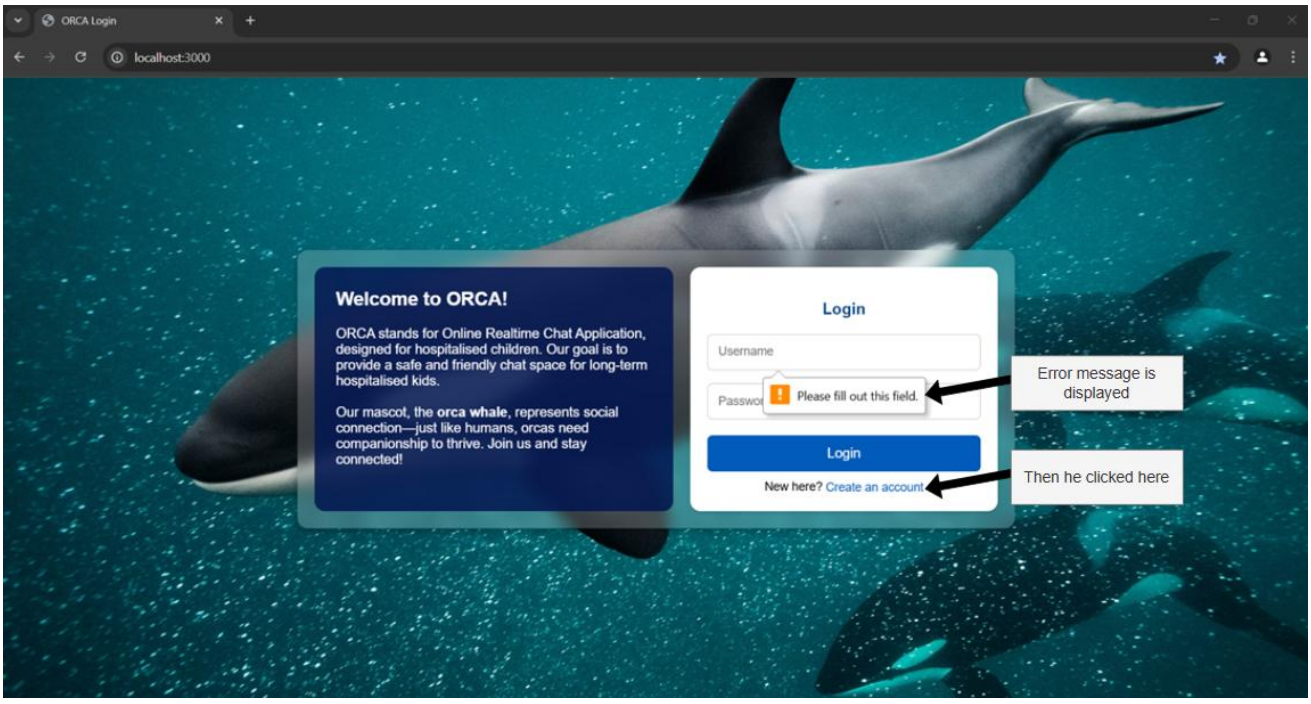
This test was successful because, the user could not bypass the chat filter by using capitalisation. This ensures that users can't use inappropriate language (banned words) on the website. Furthermore, this prove the robustness of the chat filter as it will still censor banned words even if they are capitalised.

Test 20

The page that both the older child and the younger child saw was the login page. The older child instantly clicked on the 'create an account' link to create an account.



Whereas the younger child clicked on the login button and was met with an error. He was then able to read the rest of the page where he found the redirect link to create an account.

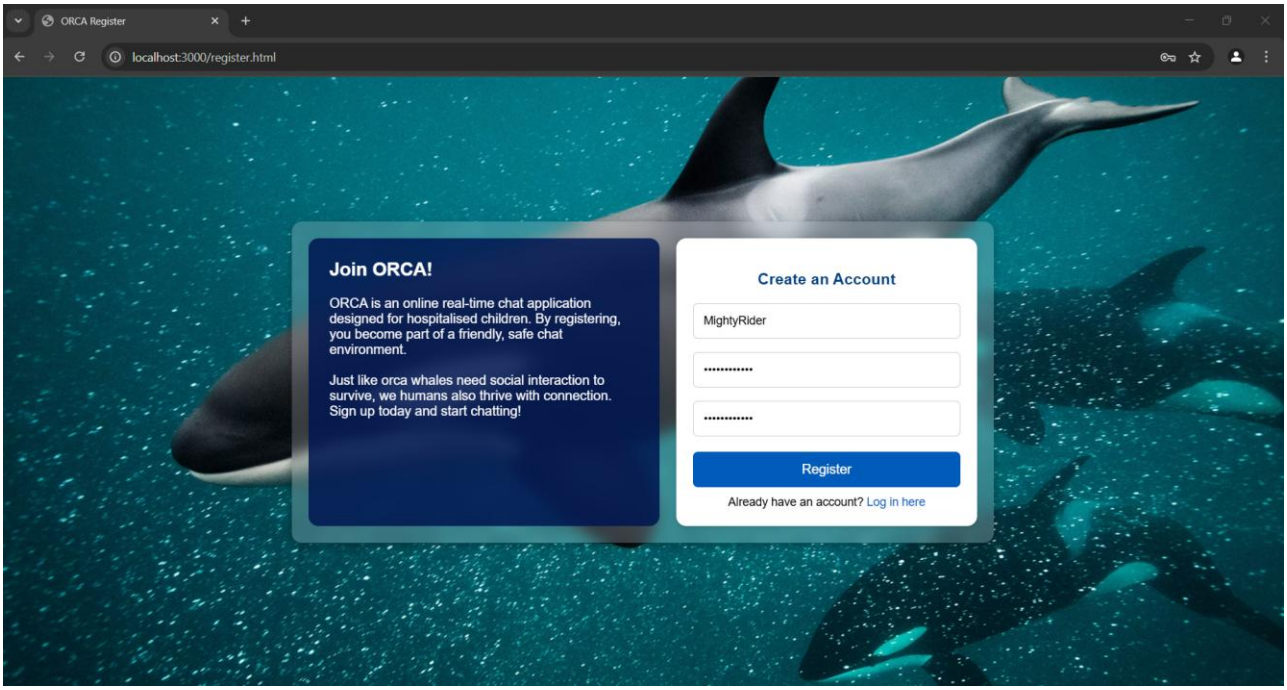


This test was successful because both users were able to navigate the login page, and they both understood that they did not yet have accounts. Therefore, they both clicked on the registration link below, which took them to the register page. The link stands out against the white login box, which helps it to stand out. Furthermore, the link is accompanied by text

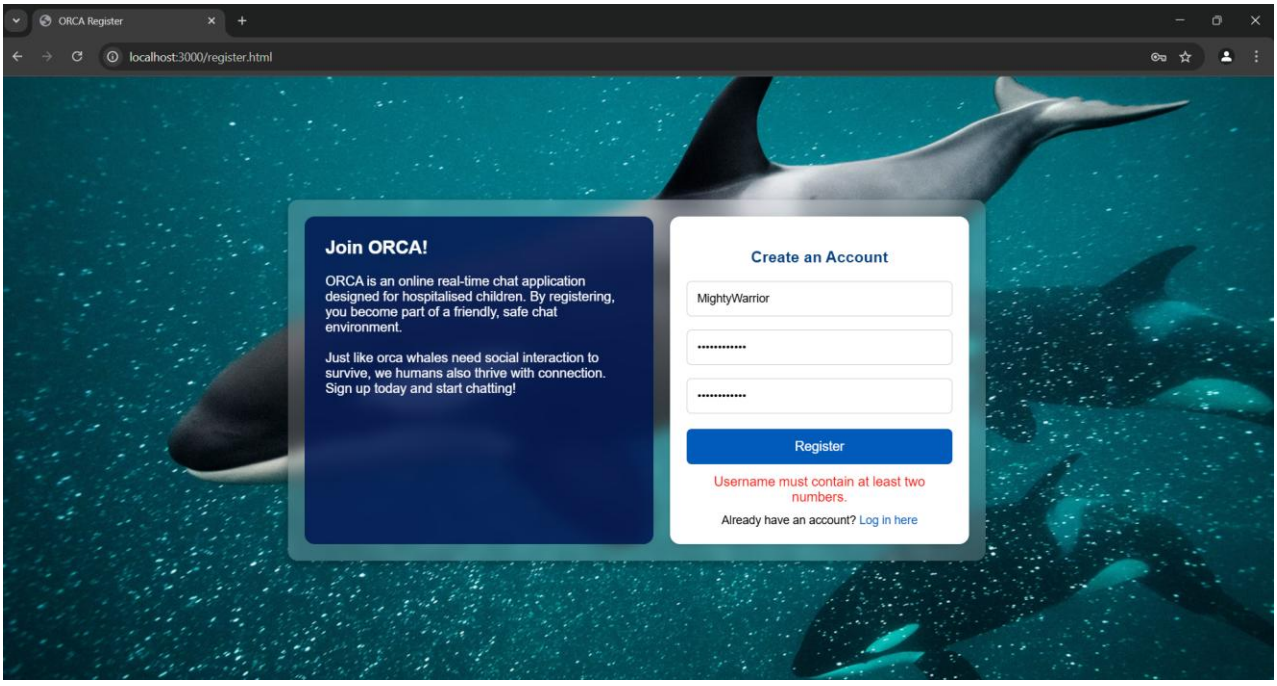
(New here?), and the link itself explains its purpose (Creating an account). This combination of hints and clear design helps the login page to be easy to navigate and understand.

Test 21

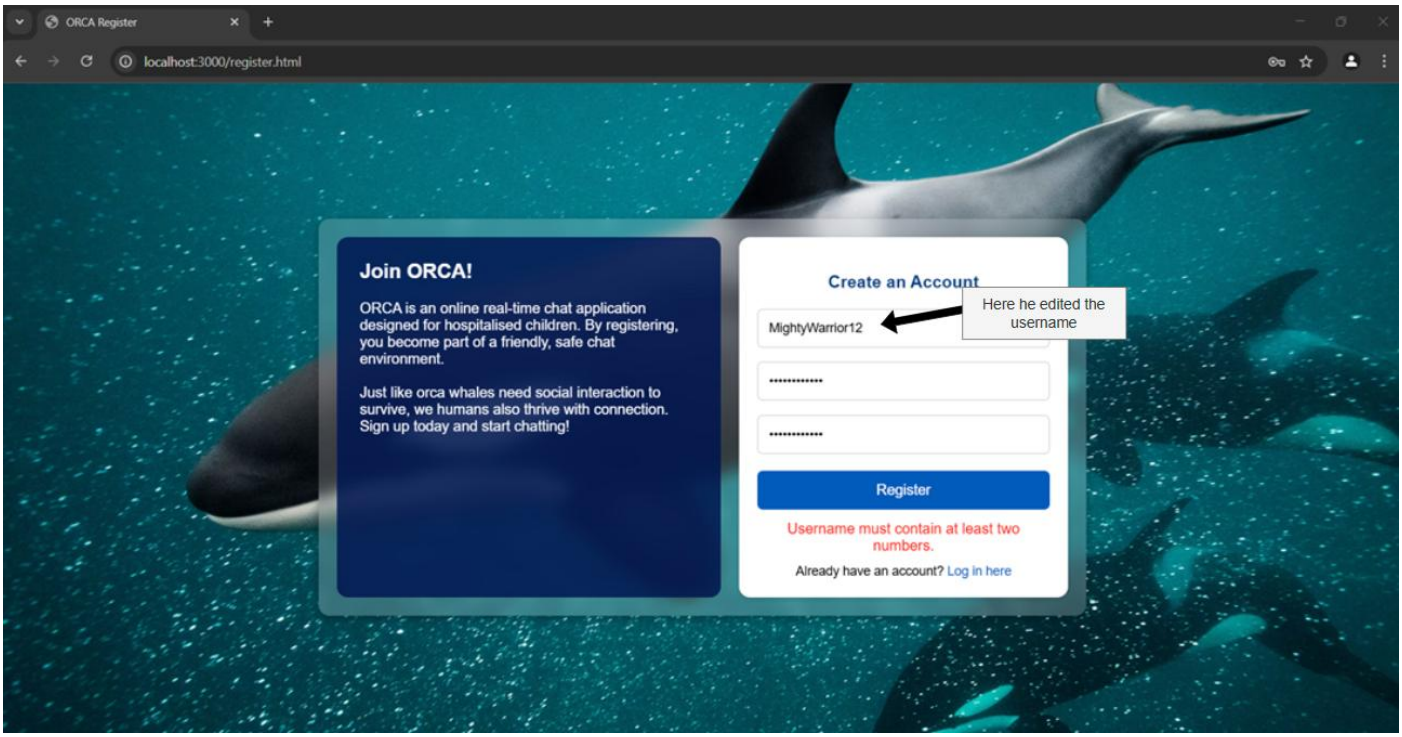
Now, on the register page, the older child was filled in the username input with 'MightyWarrior'. He then made a password and inputted in corrected in the confirm password input box without issue.

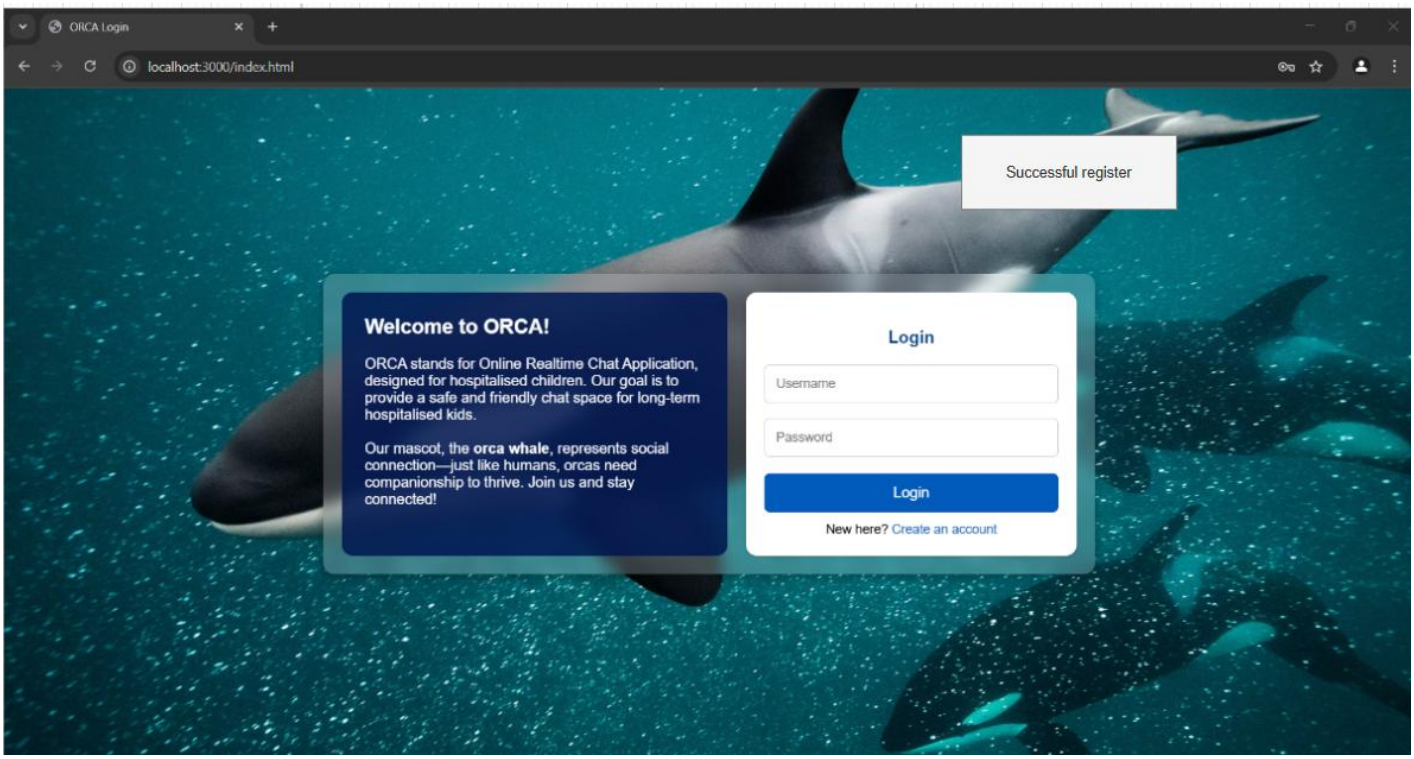


However, when he clicked the register button, he was met with an error that said his username must contain at least two numbers.

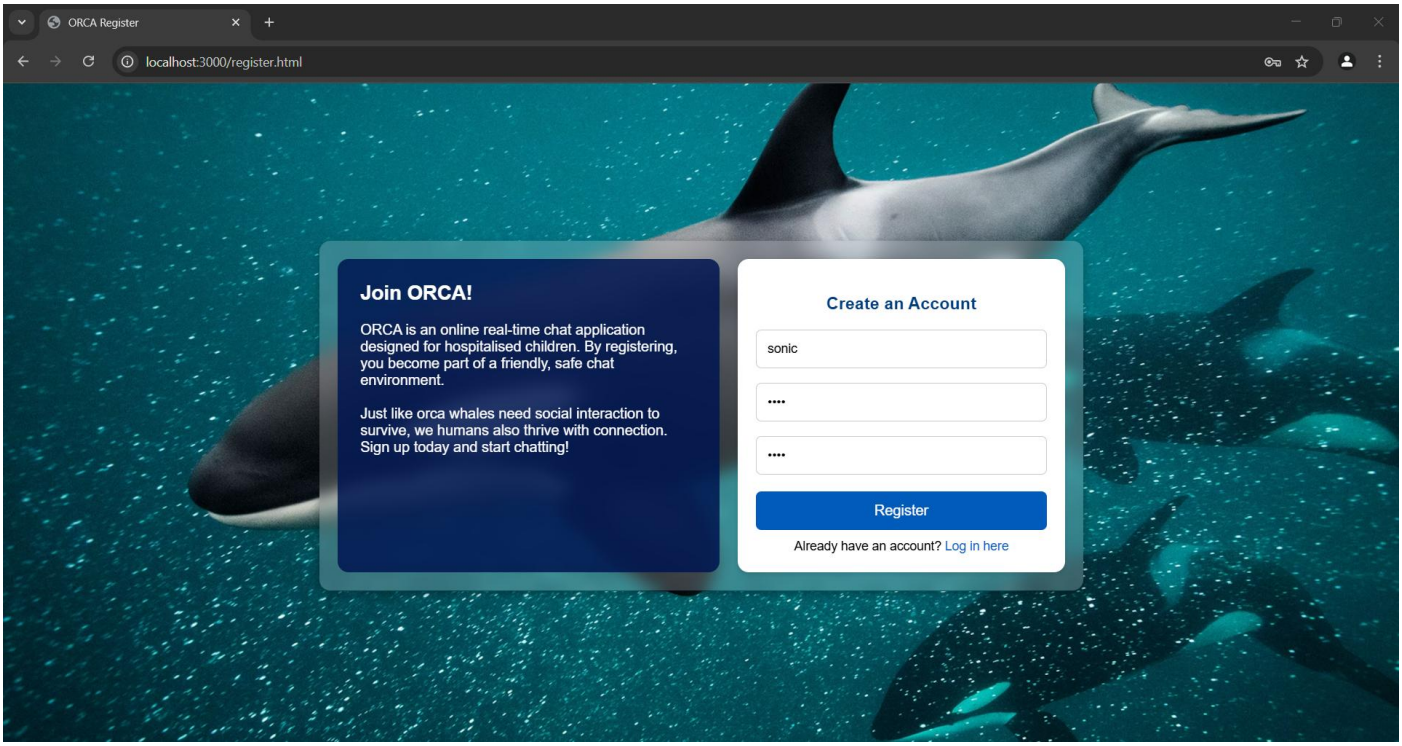


After being notified of the error, the older child was able to edit his username to 'MightyWarrior12' and successfully registered a new account.

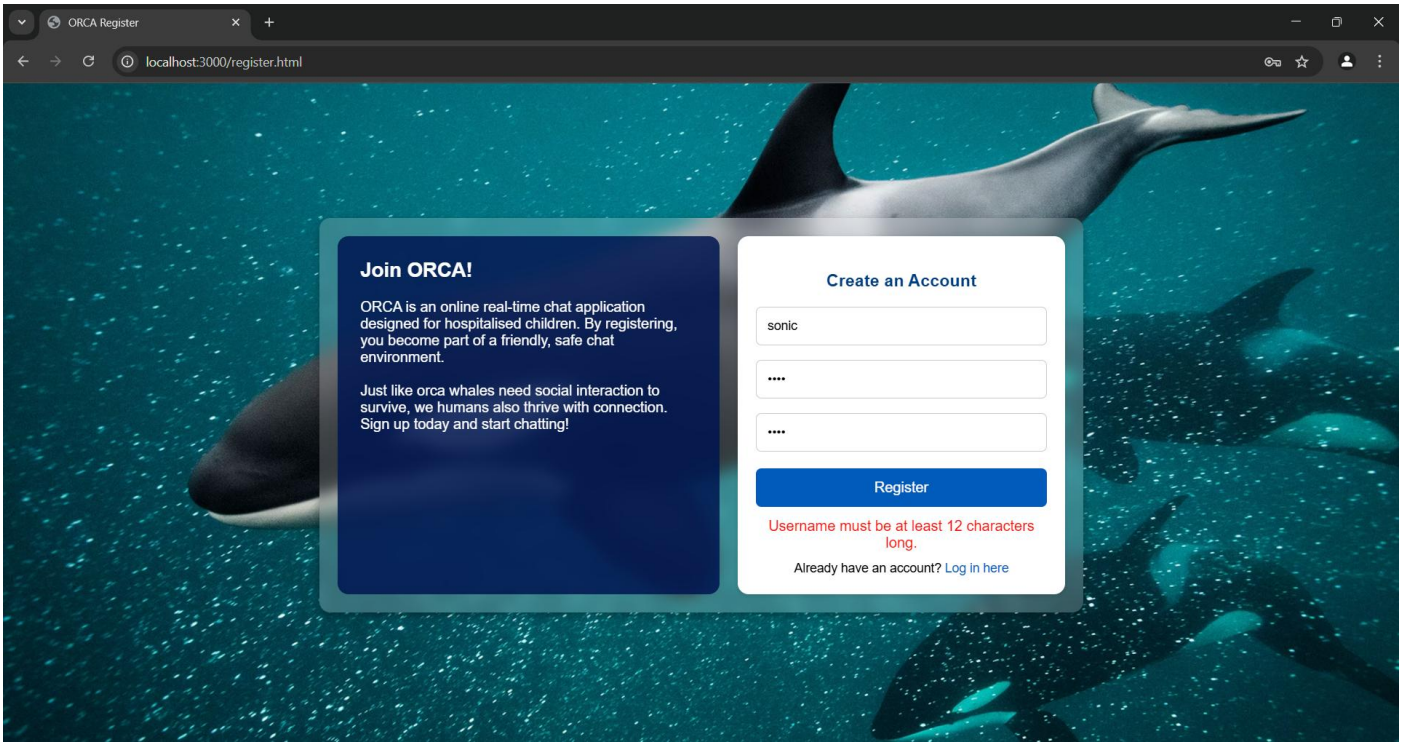

ORCA Register
localhost:3000/register.html
Join ORCA!
ORCA is an online real-time chat application designed for hospitalised children. By registering, you become part of a friendly, safe chat environment.
Just like orca whales need social interaction to survive, we humans also thrive with connection. Sign up today and start chatting!
Create an Account
MightyWarrior12
Here he edited the username
Register
Username must contain at least two numbers.
Already have an account? Log in here


ORCA Login
localhost:3000/index.html
Successful register
Welcome to ORCA!
ORCA stands for Online Realtime Chat Application, designed for hospitalised children. Our goal is to provide a safe and friendly chat space for long-term hospitalised kids.
Our mascot, the orca whale, represents social connection—just like humans, orcas need companionship to thrive. Join us and stay connected!
Login
Username
Password
Login
New here? Create an account

On the other hand, the younger child creates an account using the username ‘Sonic’ and with a password shorter than 8 characters.

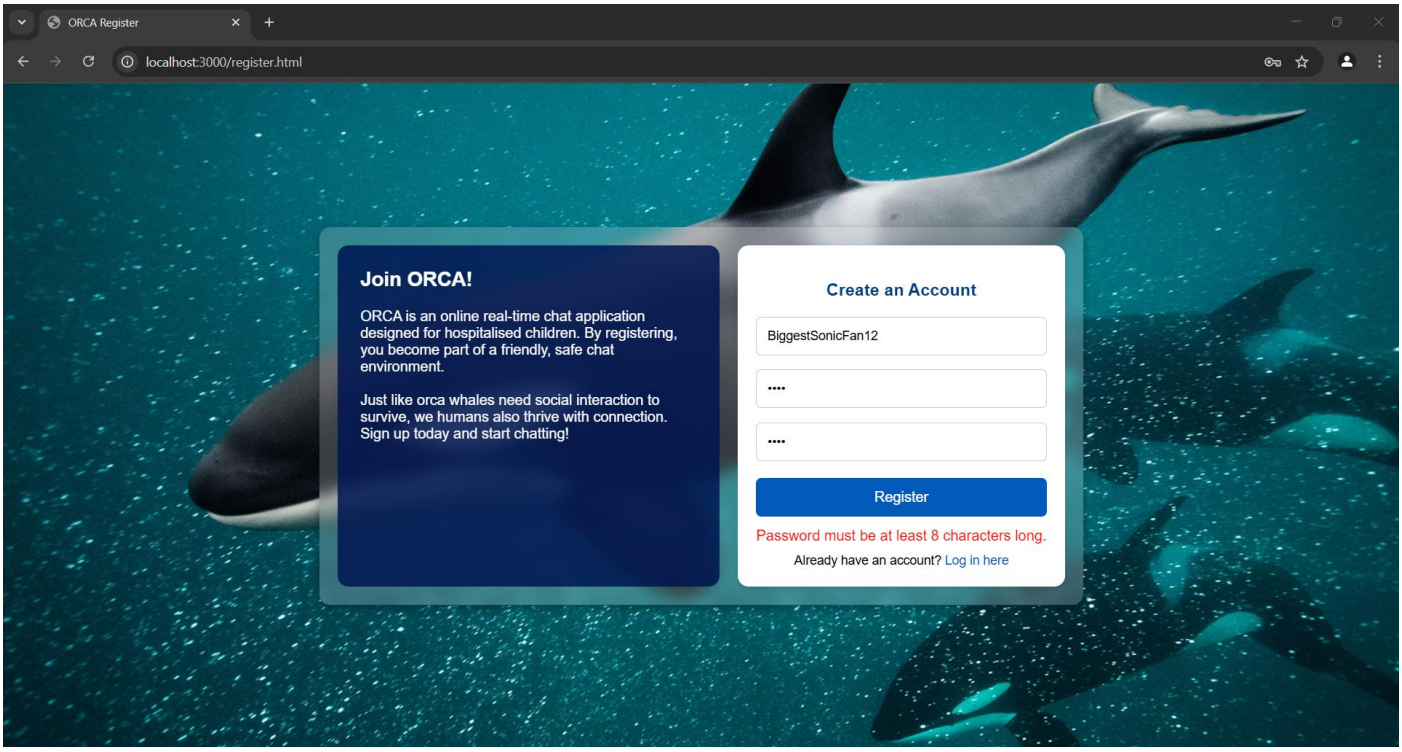


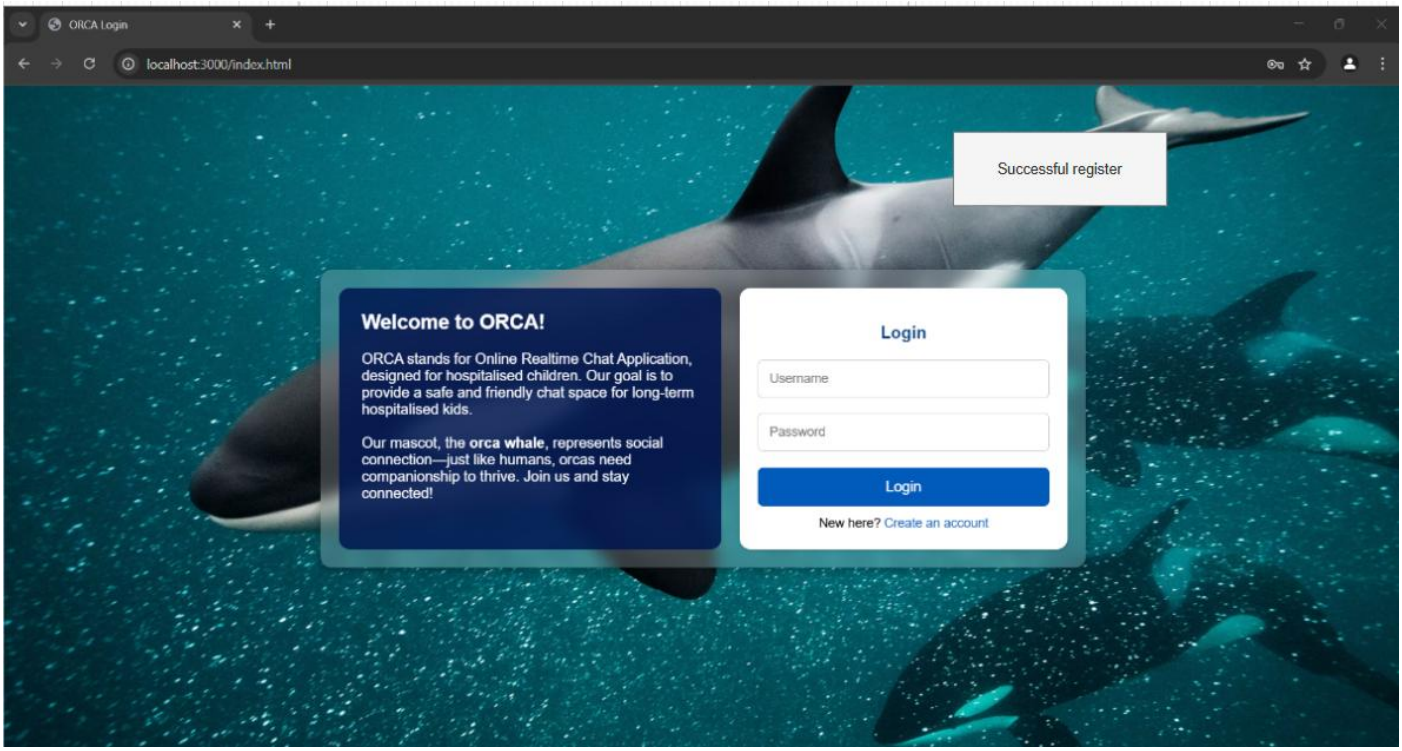
He first encountered the error that his username was too short and that it needs to be at least 12 characters long. He then changed his username to ‘BiggestSonicFan. ’



Now, his username needed two numbers, which he added. His username is now BiggestSonicFan12.

However, he encountered another error when creating an account; his password needed to be at least 8 characters long. The user then changed his password to be 8 characters long and then was able to log in successfully.

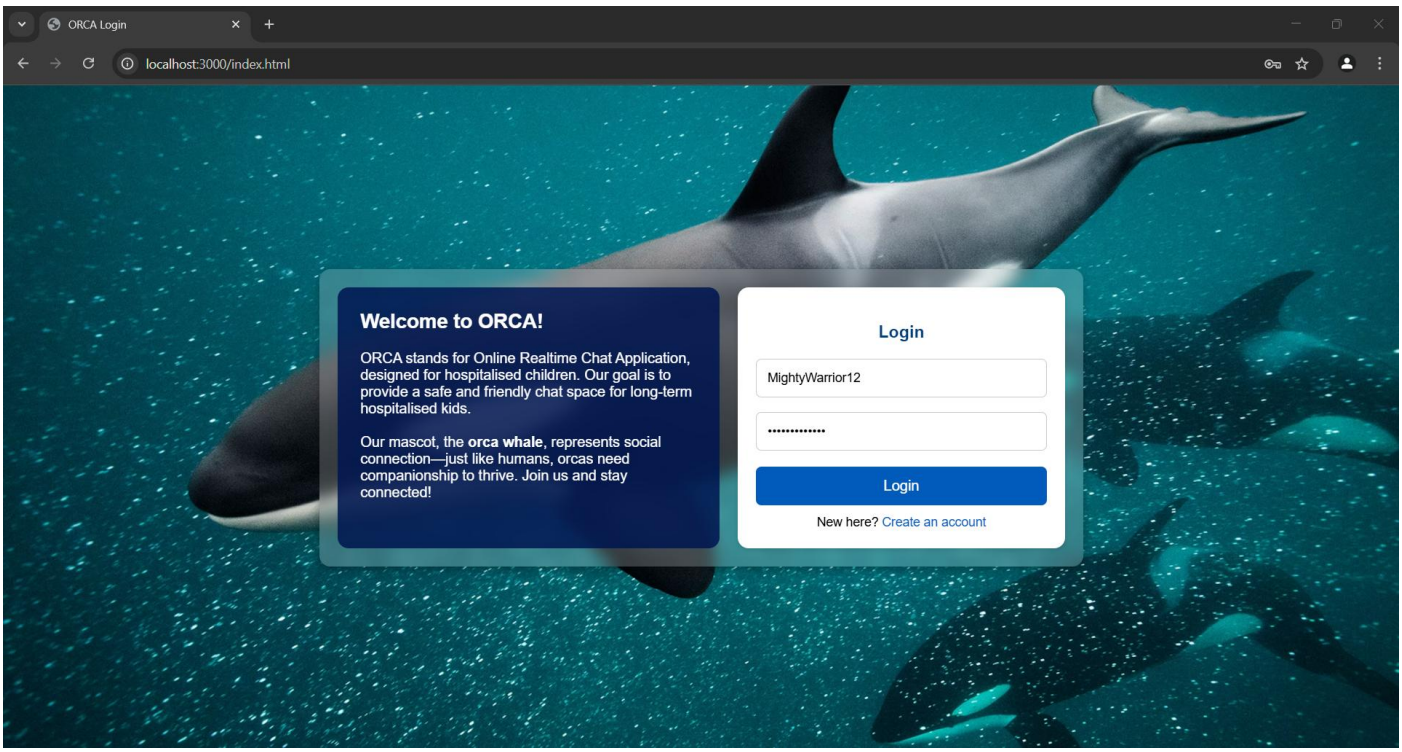


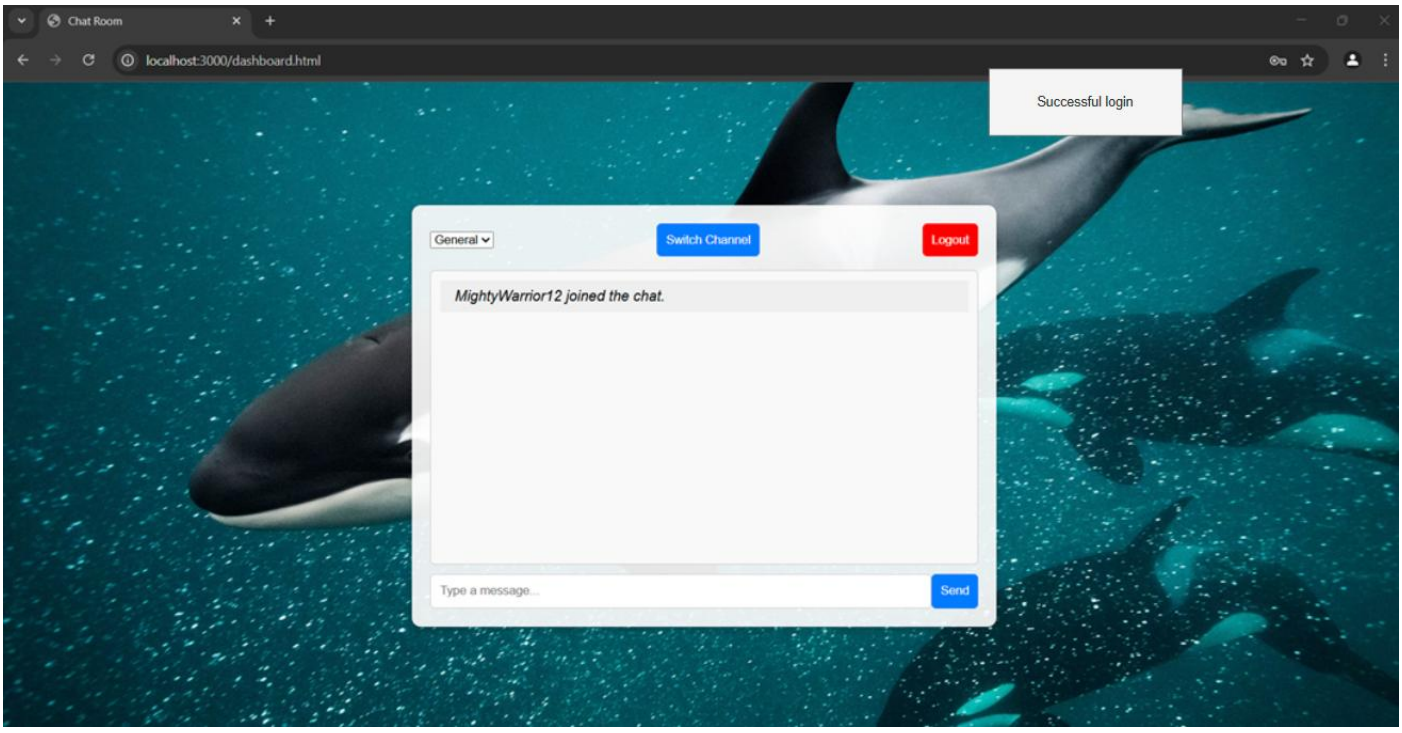


This test was successful because both users created accounts successfully. Furthermore, the users were directed on how to create valid accounts by being notified with specific error messages to advise them. This means that, even on an initial failure to create a valid account, the users can successfully register into the website after multiple tries. The accounts that the users create must be valid, as this will maintain the integrity of the user database. This leads to improved maintenance of the database as entries are validated before they are inserted, preventing inconsistencies.

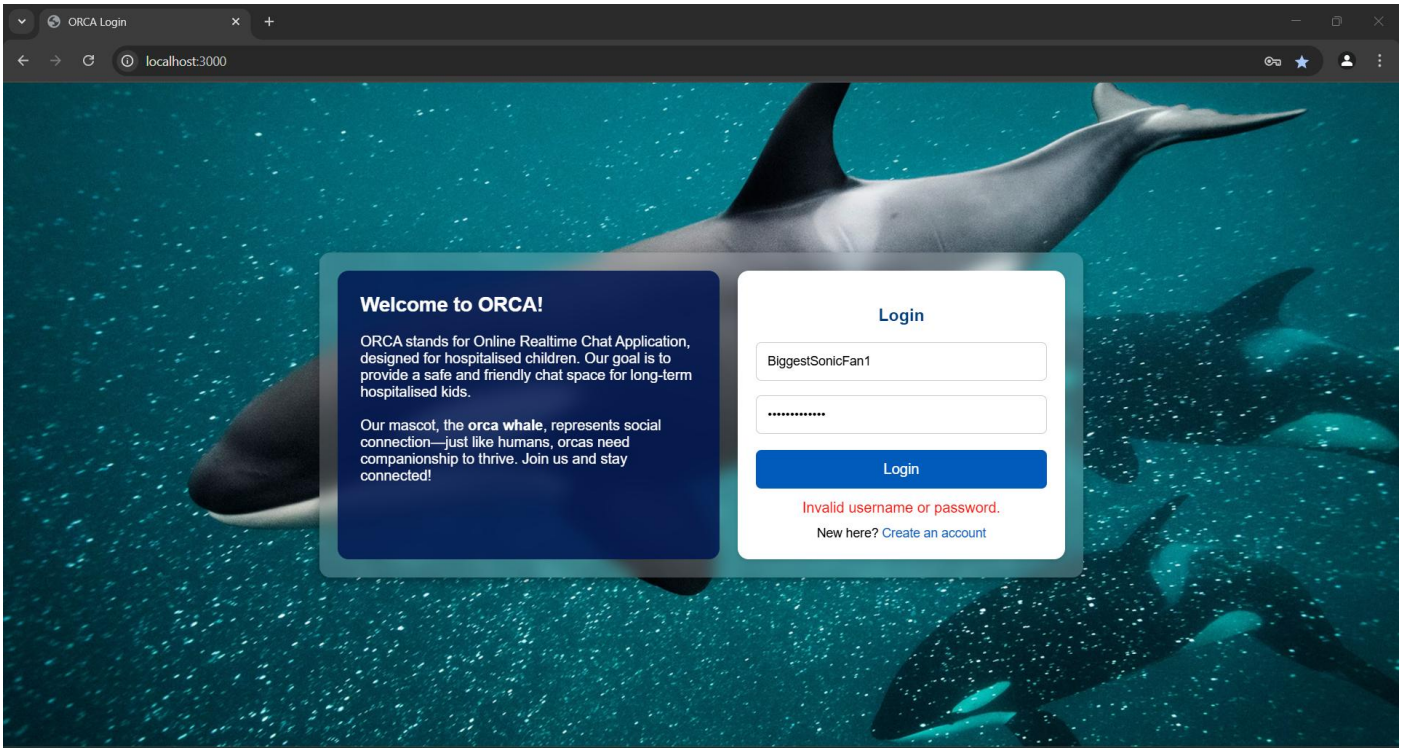
Test 22

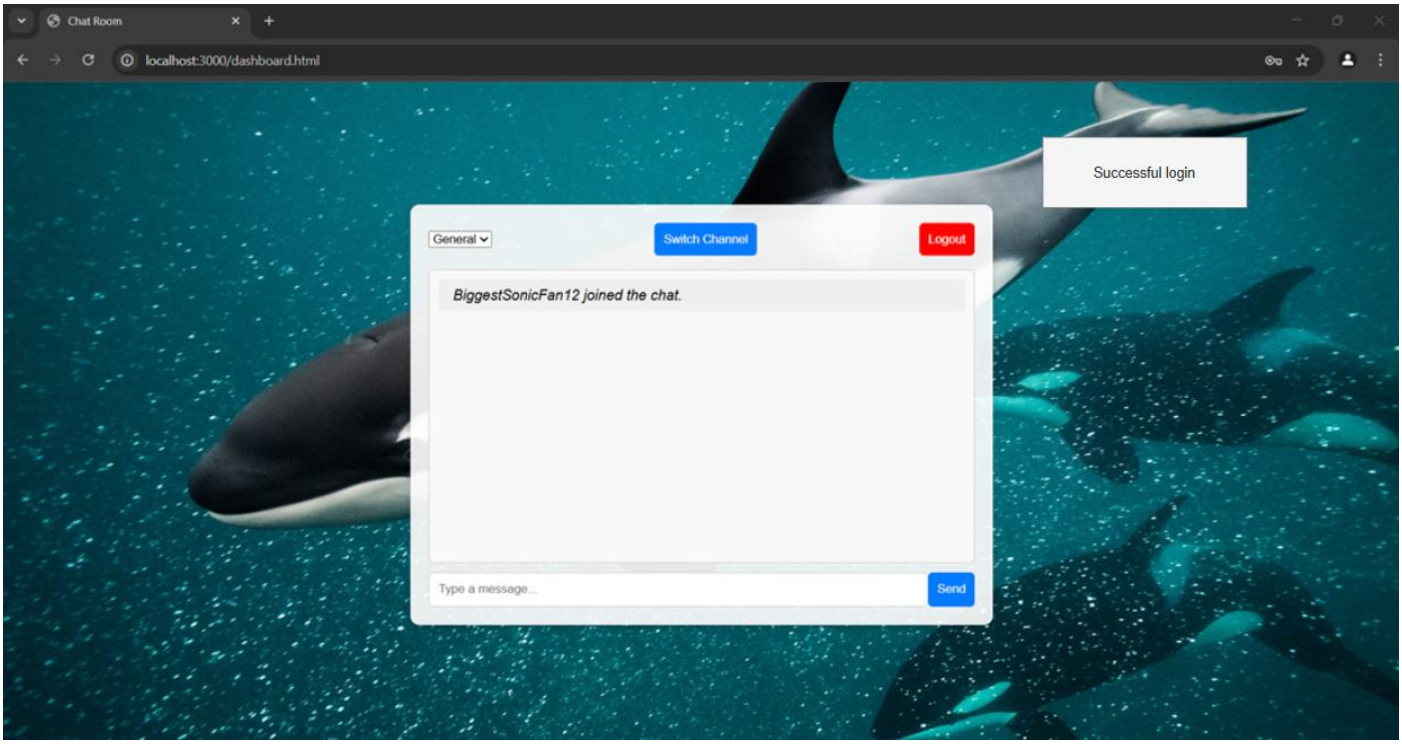
Once both the younger child and the older child reach the login page after registering their accounts, the older child logs into the website successfully on his first try.





Now, the younger child initially struggles when entering his credentials – He received an error warning that his password or username was incorrect. However, after being notified, he edited his inputs and logged into the website successfully.

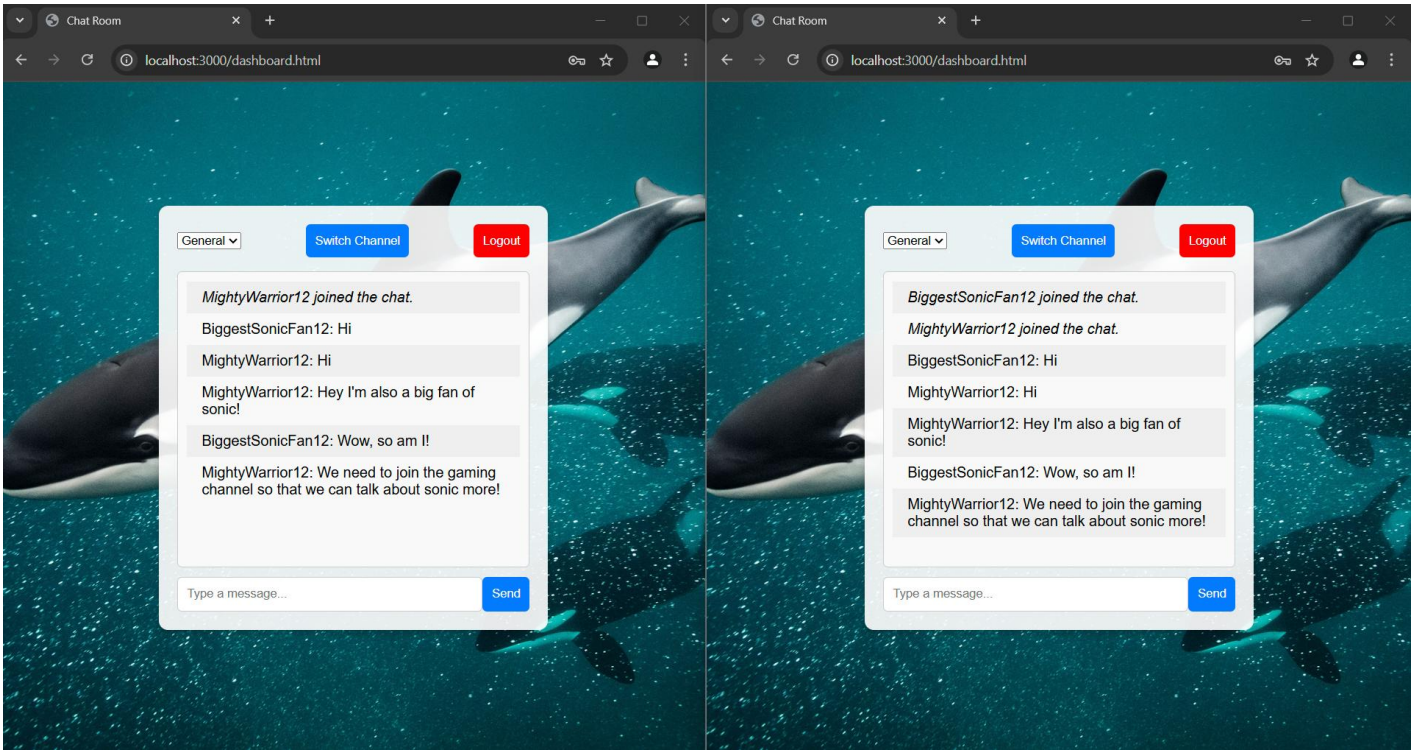




Overall, this test was a success because both users could login into the website with minimal difficulty. The error messages were useful in notifying the user if there had been a failed login due to incorrect credentials. This makes the website user friendly as users are notified on the status of their authentication, whether is was successful or not and why.

Test 23

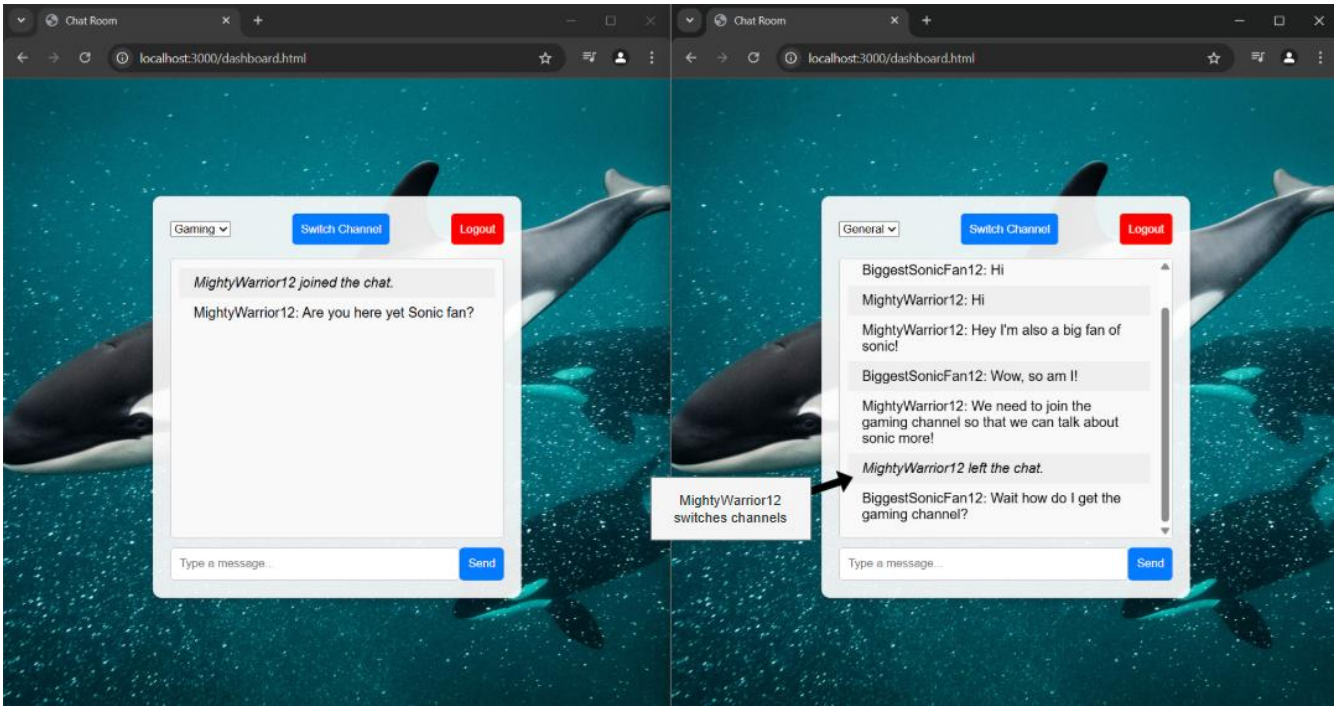
Both users were on the chat panel of the website and communicated with one another. The two found common interests which they did while in the general channel. Below is the chat that they had after join the chat panel.

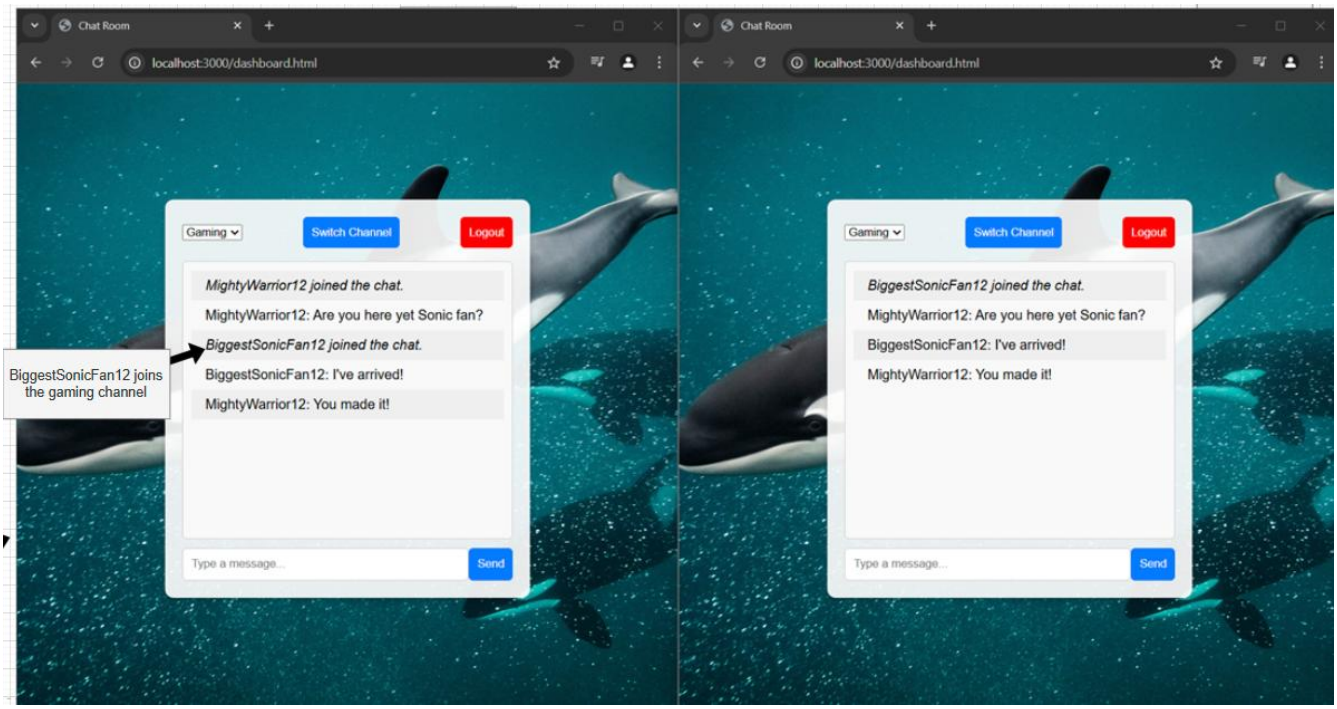


This test was successful because the two users communicated naturally due to the responsiveness of the real-time chat functionality. The barrier of distance was overcome because they could receive and send messages instantly. Moreover, this solves my stakeholder's problem of bringing children together even when separated by distance or because of medical conditions. Furthermore, this leads to my next test of the channel's functionality.

Test 24

The older child first switches channels to the gaming channel, but the younger child struggles to get to the gaming channel initially. However, he is able to join the older child in the gaming channel after searching the page for the drop down menu then clicking the switch channel button.





The test was a success because the older child and the younger child were able to switch channels to continue their conversation in a more relevant channel. This fulfils my stakeholder's requirements because the children can find people with similar interests, forming a type of community. Furthermore, the separate channels allow children with specific interests to meet others with similar interests.

Unmet success criteria

I did not meet this success criterion to its fullest extent.

1) **Moderation and user safety** – Ensuring security and safety on the website is especially important for my stakeholders since they are responsible for the care of the children and young people who will use the website.

The two failed tests below describe the uncompleted aspects of this success criteria and how they could be implemented in the future. However, this success criteria was not fully unmet because I implemented the chat filter to censor inappropriate words on the chat panel. Evidence of this can be seen in Test 10 and Test 19.

Test 4 - I failed to implement the feature of different access levels because of time constraints. On page 97, I describe how I would implement this feature in future development. Essentially, I would store the users' access level in the user database; from this access level, I can send the user relevant user interfaces. For example, the older child access level would grant the users access to more adult topic channels like sex education, while the younger child access level would not have access to this top channel. Furthermore, the admin access level would grant access to the admin panel. Overall, this feature would be useful in moderating and keeping users safe because younger children would be protected from inappropriate content. Moreover, the admin access level, granting access to the admin panel, would be used to moderate users and messages sent on the website – protecting all users from anti-social behaviour and inappropriate content (safeguarding the community).

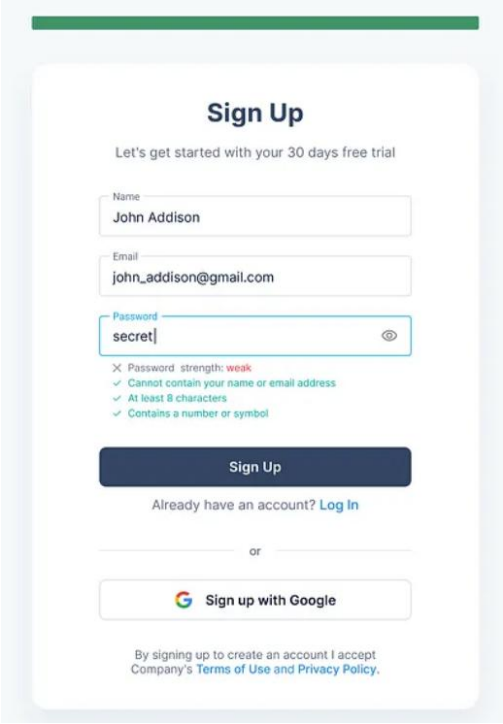
Test 11 – The next feature that I failed to implement was the admin panel and its function of altering, deleting and creating users. I failed to implement this feature because of time constraints; however, without the access levels feature, I would not have been able to prevent unauthorized access to the admin panel. This would have been a security risk if I had implemented this feature without the authenticated access levels. However, on page 105, I described how I would implement this feature, given that I had implemented the access level feature. Essentially, when a user logs in to the website, they are authenticated, and then the server will send a different user interface depending on their access level. If the user is an admin, they are sent the admin panel, where they can make changes to the user database and the chat database. This will allow admins to delete messages and alter, create, and delete users. The usefulness of this feature would be additional safeguarding of the website as admins can delete problematic users or delete offensive messages. However, an alternative to the admin panel could be that developers can make changes to the user database during runtime of the website, but that could risk the integrity of the database.

Future Improvements (Usability Features)

In this section, I will review my black box tests, specifically the three tests below. The test below contributes to the success of the success criteria below.

1) **Website functionality and user interface** – This is not as complex as real-time chat capabilities, but it will pose a challenge as it can be very time-consuming to create a user-friendly website with topic channel capabilities. Furthermore, I must create a user-friendly website since young people of all ages will be using the website. However, I plan to use pre-made assets and designs to help create a good user interface to minimise time spent on graphical improvements.

Overall, the black box tests for usability were a success because the users were able to use the website as intended with few hindrances. This success fulfils my stakeholder's requirement for the website to be user-friendly as users can understand how to interact with the website naturally without external assistance. This is important because the users of the website vary in ages therefore, the website needs to be suitable for younger children and older children.



Test 21 – This test was an overall success; however, both users initially struggled to register an account because the credentials that they used did not fulfil the requirements. An improvement I could make in the future is to display the register credential requirements so that users know how they should enter their credentials. Furthermore, a dynamic requirement list (as shown on the left) could be used to notify the user once they have reached one of the requirements. This would make it easier for users to register on the website as the requirements are clearer and known beforehand.

Test 22 – Likewise, this test was also overall a success; however, an improvement to the login system could be that instead of sending the user the error message 'Invalid username or password, ' the system should tell the user specifically if the username or password is incorrect. This would improve clarity and make it easier for users to log in to the website.

Test 24 – Similar to the rest of the previous tests, this test was an overall success; however, the switching channel feature could be communicated clearer to the users as the younger child initially struggled to switch channels. A future improvement that I would make will be to create a channel board where users can choose the channel they want to go to by clicking on the name of the channel. The names of the channels are linked, which takes the user to the corresponding topic channel. This would, overall, allow users to avoid having to use the drop-down menu; rather, they would only need to click on the name of the channel. Furthermore, this would allow users to join topic channels on topics they are most interested in upon entering the website. Overall, it improves the user experience on the website because of the improved ease of use.

Maintenance

There being no rate limiting preventing users from spamming messages on the chat panel is a potential maintenance problem. This is because the users could spam the chat panel, slowing down the performance of the server. This will affect all users connected to the server, therefore, there is a chance that it could crash the server. Furthermore, spamming the chat panel can be considered a DDoS attack since the client can use the messages to overwhelm the server. This would affect my stakeholder greatly since spamming the chat panel could bring down the whole server, preventing other users

from using and accessing the website – essentially killing the website. However, in future development, I could add rate limiting to the https requests to prevent DDoS attacks. I would implement this by checking if the time between the client's messages is less than one second. If they are, then I will time out or disconnect the user from the server, preventing any further messages from being sent.

Another potential maintenance issue would be the database growing infinitely large since there is no cleanup policy. This means that as more messages are sent on the website, the larger the chat database will grow; theoretically, it could grow infinitely. This would affect my stakeholder greatly because the host device's memory will fill up quickly, which means that it will need to be replaced or upgraded to keep up with the website's demand. To mitigate this effect, in future development, I could have a cleanup policy where messages older than 30 days are wiped from the database. This clears up space for new messages but also prevents the database from becoming too large, taking up all the storage space on the host computer.

Sensitive information like the key to access the user's authentication password is hard-coded into the program instead of being stored in an env file (environment variable), which manages the key securely. The problem with having hardcoded keys is that keys could be leaked, therefore, unauthorised users could gain access to the website and the database. Another issue is that hardcoded keys will not scale with the website, meaning that if future developers want to add to the website, they will need to manually find the key and write it for each case as it's needed. This is bad for my stakeholder because this could be a security risk as if the key is leaked, the user database will be freely accessible to anyone. Given that the website's users are children, this could potentially pose a safeguarding threat as well. To prevent this security risk, I would use an env file to store all my keys and have the program request the keys securely from the env file. The upsides of this are that keys are scalable with the website as developers only need to request the key from the env file to receive the key.

Limitations

The project is still limited in scalability because too many users joining the server at once could overwhelm the system. This is because the website's server-side processing is limited to the processing power of the host server. Therefore, the website could crash if too many users were to use it at once due to there not being enough processing power. This could be an issue in the future because, depending on the host device of the stakeholder, the website could have too many users beyond its processing capacity. By implementing load balancing to the server, users can be distributed equally across multiple servers, increasing the website's capacity.

Next, the retrieval of data from the databases is limited in speed and will degrade over time because the records in the database are not indexed. Indexing it makes retrieval and storing of data quicker, improving the performance of the server. This is important for my stakeholder because there will be many users using the website, therefore, there will be many instances of retrieval and storing of records. A bottleneck in this system will slow down the performance of the server, meaning messages will take longer to load. In future development, I would index all the records I insert into the database because indexing allows for faster searching and retrieval.

Finally, there are no admin roles for the website to moderate the website and safeguard the children. Furthermore, there is no front-end feature to ban users or delete messages. These tools would be useful for admins to moderate the server as they can remove problematic messages or users. Moreover, admins can dynamically moderate the server better than a chat filter since a human is better at detecting and understanding inappropriate messages than a chat filter. Essentially, an admin could catch bypasses better than a chat filter. This role not being implemented affects my stakeholder because the users of the website are children, therefore, they need to use a website that is secure and age appropriate. However, in future development, I would add the admin role and a report system where users can report other users for inappropriate behaviour. The admins would then be able to see the reports and deal with them accordingly.

SOURCE CODE

index.js > configureSession

```
const express = require('express'); // Importing the Express framework
const { createServer } = require('node:http'); // Importing the createServer function from the http module
const { join } = require('node:path'); // Importing join function from the path module
const { Server } = require('socket.io'); // Importing the Server function from socket.io module
const sqlite3 = require('sqlite3'); // Importing SQlite which is a database engine (Database manager)
const { open } = require('sqlite'); // Importing the open function from SQLite to open the database
const { availableParallelism } = require('node:os'); // This function will allow for parallel proccessing on the CPU's cores
const cluster = require('node:cluster'); // This module allows us to run multiple processes (workers) in parallel.
const { createAdapter, setupPrimary } = require('@socket.io/cluster-adapter');
// This will import a function the will be used to connect worker (servers) together
const path = require('path'); // Allows the document to access paths to get other files
const authRoutes = require('./routes/auth'); // Imports the authenication routes
require('dotenv').config(); // Will store sensitive information outside the code securely
const session = require('express-session'); // Manages user sessions (keeps the users logged in)
const passport = require('passport'); // This will handle authentication
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcryptjs'); // Will hash our client's password
const userDatabase = require('./db'); // Loads the database (USERDATA)

// if this is the primary process (the first time this code has run)
if (cluster.isPrimary) {
    const numCPUs = availableParallelism(); // Returns the number of cores of the CPU
    // create one worker per available core
    for (let i = 0; i < numCPUs; i++) {
        cluster.fork({ PORT: 3000 + i });
    }
    // set up the adapter on the primary thread (Connects all workers together)
    return setupPrimary();
}


function setUpServer() {
    const app = express(); // Creates and instance of Express
    const server = createServer(app); // Creates a HTTP server
    // Express (app) will handel all incoming HTTP requests
    const io = new Server(server, { // Creates an instance of socket.io
```

```
        connectionStateRecovery: {}, // Will save the state of the server if a user disconnects
        // set up the adapter on each worker thread
        adapter: createAdapter()
        // syncs messages across all workers
    });

    return { app, server, io };
}


async function createDatabase() {
    const database = await open({
        filename: 'chat.db', // It will create a new database with this name is none is found
        driver: sqlite3.Database // Tells SQLite which drives to use when interacting with the database
    });
    // This function will wait till the database chat.db is open

    await database.exec(`
        CREATE TABLE IF NOT EXISTS messages (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            client_offset TEXT UNIQUE,
            sender TEXT,
            content TEXT,
            channel TEXT
        );
    `);
    // This function will execute the SQL commands inside the function
    /* The SQL will create the fields:
    id - the primary key that's an integer that will autoincrement
    client_offset - is a key that is unique for each client (used for searching messages for certain users)
    content - Here the message sent by the client is stored
    channel - This is the channel that the message was sent in
    */

```

```
    return database;
}


function configureSession(app) {
    // The session helps keep the user logged in
    app.use(session({
        secret: process.env.SECRET_KEY || 'your_secret_key', // secret key to protect data
        resave: false,
        saveUninitialized: false
        // Prevents unnecessary session saving
    }));

    // Initialize Passport
    app.use(passport.initialize());
    // Start the node.js passport
    app.use(passport.session());
    // Links passport to session so logged in users are remembered
}

function handleFileConnection(app) {

    // Serve static files (Give all public files to the client)
    app.use(express.static(path.join(__dirname)));
    app.use(express.static(path.join(__dirname, 'public')));

    // On request of the server, the server will responds with the HTML file (index.html) as default
    app.get('/', (req, res) => {
        res.sendFile(join(__dirname, 'public', 'index.html'));
    });

    // Middleware
    app.use(express.json());
    // Allows the server to understand JASON data
```

```
105     app.use(express.urlencoded({ extended: false }));
106     // Allows the server to read HTML form data
107
108     configureSession(app);
109
110     // Load routes
111     app.use(authRoutes);
112     app.use('/auth', require('./routes/auth'));
113 }
114
115
116 async function checkLoginCredentials(database) {
117     // The passport will recieve the username and password from the client when the user tries to login
118     passport.use(new LocalStrategy(
119         { usernameField: 'username' }, // Use username instead of default username
120         (username, password, callback) => {
121             const user = database.prepare("SELECT * FROM users WHERE username = ?").get(username);
122             // Here it searches the database for the username to get the record
123
124             if (!user) { // If username is not in the database an error message is output
125                 callback(null, false, { message: "User not found" });
126                 return
127             }
128
129             // This hashes the password and compares it with the hashed password in the database
130             bcrypt.compare(password, user.password, (err, isMatch) => {
131                 if (err) {
132                     return callback(err);
133                 }
134                 if (!isMatch){ // If the hashes don't match then an error message is output
135                     return callback(null, false, { message: "Incorrect password" });
136                 }
137                 return callback(null, user);
138             });
```

```
        }
    ));
}

async function handleClientSession(database) {
    // The sever will now remeber the use by saving their ID in the session
    passport.serializeUser((user, callback) => callback(null, user.id));
    // If they decide to change pages the ID is used to get their full details
    passport.deserializeUser((id, callback) => {
    const user = database.prepare("SELECT * FROM users WHERE id = ?").get(id);
        callback(null, user);
});
}

async function authenticateClient(database) {

    await checkLoginCredentials(database)

    await handleClientSession(database);

}


const channels = {};
// container for the channels
function removeUserFromChannel(io, socket){
    for (let channel_ID in channels) {
        if (channels[channel_ID].users[socket.id]) {
            let username = channels[channel_ID].users[socket.id];
            delete channels[channel_ID].users[socket.id];
            socket.leave(channel_ID);
            io.to(channel_ID).emit('user-left', username);
```

```
        }
    }
}

// This function is the main part of our program, it has one entry point and is asynchronous
async function main() {

    const { app, server, io } = setUpServer();

    const messageDatabase = await createDatabase();

    handleFileConnection(app);

    await authenticateClient(userDatabase);

    // When there is a connection to the server this event will fire
    io.on('connection', async (socket) => {
        console.log('A user connected');

        socket.on('join channel', async (channel_ID, username, callback) => {

            removeUserFromChannel(io, socket);

            socket.join(channel_ID); // client joins the channel
            if (!channels[channel_ID]) {
                channels[channel_ID] = { name: `Room ${channel_ID}`, users: {} };
                // If the channel doesn't exist then a new one is created
            }
            channels[channel_ID].users[socket.id] = username;
            // client's username store in corresponding channel

            // The username is sent to all the client in the channel - (updating the clients on who joined)
            io.to(channel_ID).emit('user-joined', username, Object.keys(channels[channel_ID].users));
```

```
        console.log(`${username} joined channel ${channel_ID}`);
        callback(`Joined channel ${channel_ID}`);

        // Fetch past messages for this channel
        try {
            const messages = await messageDatabase.all('SELECT id, content, sender FROM messages WHERE channel = ? ORDER BY id ASC', channel_ID);
            messages.forEach((row) => {
                socket.emit('chat message', row.content, row.id, row.sender);
            });
        } catch (e) {
            console.error('Error fetching messages:', e);
        }
    });

```

```
// This function will run once a 'chat message' is received by the server
socket.on('chat message', async (msg, clientOffset, channel_ID, sender, callback) => {
    let result;
    try {
        result = await messageDatabase.run('INSERT INTO messages (content, client_offset, channel, sender) VALUES (?, ?, ?, ?)', msg, clientOffset, channel_ID, sender);
        // Will insert the message into the database
    } catch (e) {
        if (e.error === 19) {
            console.error('Error inserting messages:', e);
            //callback(e); // Will notifiy the client when there is a duplicate message
        }
        return;
    }
    // Will send messages to all client in the corresponding channel
    io.to(channel_ID).emit('chat message', msg, result.lastID, sender);
    callback(`message has been sent in ${channel_ID}`); // acknowledge the event
});

```

```
236         // this will remove the client from the channel once the client disconnects
237         socket.on('disconnect', () => {
238             console.log('User disconnected');
239             removeUserFromChannel(io, socket);
240         });
241
242     });
243
244     // each worker will listen on a distinct portSS
245     const port = process.env.PORT;
246     server.listen(port, () => {
247         console.log(`Server running at http://localhost:${port}`);
248     });
249 }
250
251 // Calling the main block of our program to start/run the server
252 main();
253
```

JS db.js > ...

```
1  const Database = require('better-sqlite3');
2  const db = new Database('chat_auth.db');
3
4  // Create Users table if not exists
5  db.prepare(`
6    CREATE TABLE IF NOT EXISTS users (
7      id INTEGER PRIMARY KEY AUTOINCREMENT,
8      username TEXT UNIQUE NOT NULL,
9      password TEXT NOT NULL
10   )
11 `).run();
12
13 // exports database so that index.js can access it
14 module.exports = db;
15
```

routes > JS auth.js > validateUserInput

```
const express = require('express');
const bcrypt = require('bcryptjs');
const passport = require('passport');
const db = require('../db');

const router = express.Router();

// Helper function to validate user input
const validateUserInput = (username, password) => {
    if (!username || !password) {
        return "Username and password are required.";
    }
    if (username.length < 12) {
        return "Username must be at least 12 characters long.";
    }
    if ((username.match(/\d/g) || []).length < 2) {
        return "Username must contain at least two numbers.";
    }
    if (password.length < 8) {
        return "Password must be at least 8 characters long.";
    }
    return null; // No validation errors
};

// Register Route
router.post('/register', async (req, res) => {
    try {
        const { username, password } = req.body;

        // Validate user input
        const validationError = validateUserInput(username, password);
        if (validationError) {
            return res.status(400).json({ message: validationError });
        }

```

```
        // Check if username already exists
        const userExists = db.prepare("SELECT * FROM users WHERE username = ?").get(username);
        if (userExists) {
            return res.status(400).json({ message: "Username already exists" });
        }

        // Hash password and insert user
        const hashedPassword = await bcrypt.hash(password, 10);
        db.prepare("INSERT INTO users (username, password) VALUES (?, ?)").run(username, hashedPassword);

        return res.status(200).json({ message: "User registered successfully!" });
    } catch (err) {
        console.error("Error during registration:", err.message);
        return res.status(500).json({ message: "Server error. Please try again later." });
    }
});

// Login Route
router.post('/login', (req, res, next) => {
    passport.authenticate('local', (err, user, info) => {
        if (err) {
            console.error("Authentication error:", err.message);
            return res.status(500).json({ message: "Internal server error" });
        }
        if (!user) {
            return res.status(400).json({ message: "Invalid username or password." });
        }

        req.logIn(user, (err) => {
            if (err) {
                console.error("Login error:", err.message);
                return res.status(500).json({ message: "Internal server error" });
            }
```

```
            res.redirect("/dashboard.html");
            return
        });
    })(req, res, next);
});

// Logout Route
router.get('/logout', (req, res) => {
    req.logout((err) => {
        if (err) {
            console.error("Logout error:", err.message);
            return res.status(500).json({ message: "Logout error. Please try again." });
        }
        res.redirect('/index.html');
    });
});

module.exports = router;
```

public > # style.css > .input-box

```css
body {
    font-family: Arial, sans-serif;
    background: url(src/orca_image.avif) no-repeat center center/cover;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    flex-direction: column;
}
.chat-container {
    background: rgba(255, 255, 255, 0.9);
    padding: 20px;
    border-radius: 10px;
    width: 50%;
    max-width: 600px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}
.header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 15px;
}
.chat-box {
    height: 300px;
    border: 1px solid #ccc;
    padding: 10px;
    overflow-y: auto;
    background: #f9f9f9;
    border-radius: 5px;
}
.input-box {
    display: flex;
    margin-top: 10px;
}
```

```
}
input {
    flex: 1;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
button {
    padding: 10px;
    border: none;
    background: #007BFF;
    color: white;
    border-radius: 5px;
    cursor: pointer;
}
.logout {
    background: red;
}
 /* Style for message list */
 #messages { list-style-type: none; margin: 0; padding: 0; }
#messages > li { padding: 0.5rem 1rem; }
#messages > li:nth-child(odd) { background: #efefef; }

/* Background Styling */
body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background: url(src/orca_image.avif) no-repeat center center/cover;
}

```

```
/* Main Container */
.container {
    display: flex;
    max-width: 800px;
    width: 90%;
    background: rgba(255, 255, 255, 0.2);
    backdrop-filter: blur(8px);
    padding: 20px;
    border-radius: 12px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);
    gap: 20px;
}

/* Welcome Box */
.welcome-box {
    flex: 1;
    background: rgba(0, 20, 80, 0.85);
    color: white;
    padding: 25px;
    border-radius: 12px;
    min-width: 280px;
}

.welcome-box h2 {
    margin-top: 0;
    font-size: 22px;
}

/* Login Box */
.login-box {
    background: white;
    padding: 20px;
    border-radius: 12px;
    box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.15);
    width: 320px;
```

```
105     width: 320px;
106     display: flex;
107     flex-direction: column;
108     align-items: center;
109 }
110
111 .login-box h3 {
112     text-align: center;
113     margin-bottom: 10px;
114     color: #003f7f;
115 }
116
117 /* Form Inputs */
118 .login-box form {
119     width: 100%;
120     display: flex;
121     flex-direction: column;
122     align-items: center;
123 }
124
125 .login-box input {
126     width: 100%;
127     padding: 12px;
128     margin: 8px 0;
129     border: 1px solid #ccc;
130     border-radius: 6px;
131     font-size: 14px;
132     box-sizing: border-box;
133 }
134
135 .login-box button {
136     width: 100%;
137     padding: 12px;
138     background: #005bbb;
139     color: white;
```

```
138     background: #005bbb;
139     color: white;
140     border: none;
141     border-radius: 6px;
142     cursor: pointer;
143     font-size: 16px;
144     margin-top: 10px;
145 }
146
147 .login-box button:hover {
148     background: #004080;
149 }
150
151 /* Register Link */
152 .register-link {
153     margin-top: 10px;
154     font-size: 14px;
155 }
156
157 .register-link a {
158     color: #005bbb;
159     text-decoration: none;
160 }
161
162 .register-link a:hover {
163     text-decoration: underline;
164 }
165
166 .error-message {
167     color: red;
168     text-align: center;
169     display: none;
170     margin-top: 12px;
171 }
172
```

```
@media (max-width: 768px) {
    .container {
        flex-direction: column;
        text-align: center;
        align-items: center;
    }
    .welcome-box {
        margin-right: 0;
        margin-bottom: 20px;
    }
}
        /* Background Styling */
        body {
            margin: 0;
            padding: 0;
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background: url(src/orca_image.avif) no-repeat center center/cover;
        }

        /* Main Container */
        .container {
            display: flex;
            max-width: 800px;
            width: 90%;
            background: rgba(255, 255, 255, 0.2);
            backdrop-filter: blur(8px);
            padding: 20px;
            border-radius: 12px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);
            gap: 20px;
        }
```

```
208
209         /* Welcome Box */
210         .welcome-box {
211             flex: 1;
212             background: rgba(0, 20, 80, 0.85);
213             color: white;
214             padding: 25px;
215             border-radius: 12px;
216             min-width: 280px;
217         }
218
219         .welcome-box h2 {
220             margin-top: 0;
221             font-size: 22px;
222         }
223
224         /* Register Box */
225         .register-box {
226             background: white;
227             padding: 20px;
228             border-radius: 12px;
229             box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.15);
230             width: 320px;
231             display: flex;
232             flex-direction: column;
233             align-items: center;
234         }
235
236         .register-box h3 {
237             text-align: center;
238             margin-bottom: 10px;
239             color: #003f7f;
240         }
241
242         /* Form Inputs */
```

```
242         /* Form Inputs */
243         .register-box form {
244             width: 100%;
245             display: flex;
246             flex-direction: column;
247             align-items: center;
248         }
249
250         .register-box input {
251             width: 100%;
252             padding: 12px;
253             margin: 8px 0;
254             border: 1px solid #ccc;
255             border-radius: 6px;
256             font-size: 14px;
257             box-sizing: border-box;
258         }
259
260         .register-box button {
261             width: 100%;
262             padding: 12px;
263             background: #005bbb;
264             color: white;
265             border: none;
266             border-radius: 6px;
267             cursor: pointer;
268             font-size: 16px;
269             margin-top: 10px;
270         }
271
272         .register-box button:hover {
273             background: #004080;
274         }
275
276         /* Login Link */
277         .login-link {
```

```
277         .login-link {
278             margin-top: 10px;
279             font-size: 14px;
280         }
281
282         .login-link a {
283             color: #005bbb;
284             text-decoration: none;
285         }
286
287         .login-link a:hover {
288             text-decoration: underline;
289         }
290
291         .error-message {
292             color: red;
293             text-align: center;
294             display: none;
295             margin-top: 12px;
296         }
297
298         @media (max-width: 768px) {
299             .container {
300                 flex-direction: column;
301                 text-align: center;
302                 align-items: center;
303             }
304             .welcome-box {
305                 margin-right: 0;
306                 margin-bottom: 20px;
307             }
308         }
```

public > JS script.js > ...

```
let counter = 0;
// will be incremented so that the clientOffset is always unique

// This creates a reference of the server on the client side
let socket = io({
    auth: {
        serverOffset: 0
    },
    // How long the client will wait for an acknowlegdement from the server
    ackTimeout: 10000,
    // The amount of times the client will try to send messages to the server
    retries: 3,
});

const form = document.getElementById('form');
const input = document.getElementById('input');
const message = document.getElementById('messages'); // Container for the recieved messages
// List of banned words (you can expand this list)
const bannedWords = ["crap", "curse", "offensive"];

let channel_ID = "general"; // the default channel
let username = prompt("Enter your username:") || "Anonymous";
// This will ask the client for their name, if they enter nothing their name is set to Anonymous


// Join a channel
socket.emit('join channel', channel_ID, username, (response) => {
    console.log(response);
});



// Function to censor words in a message
function censorMessage(message) {
    let words = message.split(/\b/); // Split by word boundaries

```

```
    return words.map(word => {
        let lowerCaseWord = word.toLowerCase(); // Puts the word into lowercase
        if (bannedWords.includes(lowerCaseWord)) {
            return "#".repeat(word.length); // Replace with hashtags
        }
        return word; // Keep non-banned words unchanged
    }).join(""); // Reassemble the sentence
}

// Once the submit button is pressed, the client chencks if the message is not empty
// Then the client sends this message to the client
form.addEventListener('submit', (e) => {
    e.preventDefault();
    if (input.value) {
        // Creates an unique identifier for each message
        let clientOffset = `${socket.id}-${Date.now()}`;
        socket.emit('chat message', censorMessage(input.value), clientOffset, channel_ID, username);
        input.value = '';
        input.placeholder = "Type a message...";
    } else {
        input.placeholder = "Invalid input";
    }
});

socket.on('chat message', function(msg, serverOffset, sender) { // the message and the state of the server sent from the server
    // Creates a list element that will contain the message
    const item = document.createElement('li');
    item.textContent = `${sender}: ${msg}`;
    message.appendChild(item);
    window.scrollTo(0, document.body.scrollHeight); // Moves down the pages to fit the message
    socket.auth.serverOffset = serverOffset;
    // This will update the state of the client to be synced with the state of the server
});
// Once the message is recieved, it will be displayed to the client

```

```

    // This will run when user joins the server
    socket.on('user-joined', (username, users) => {
        // A join message message is displayed to let clients know that a user has join
        const item = document.createElement('li');
        item.textContent = `${username} joined the chat.`;
        item.style.fontStyle = "italic";
        message.appendChild(item);
    });

    // Once a client disconncets a message is sent to the clients
    // Notifying them that someone has left
    socket.on('user-left', (username) => {
        const item = document.createElement('li');
        item.textContent = `${username} left the chat.`;
        item.style.fontStyle = "italic";
        message.appendChild(item);
    });
```

public > register.html > html > body > div.container

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ORCA Register</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
    <!-- Welcome Message -->
    <div class="welcome-box">
        <h2>Join ORCA!</h2>
        <p>
            ORCA is an online real-time chat application designed for hospitalised children.
            By registering, you become part of a friendly, safe chat environment.
            <br><br>
            Just like orca whales need social interaction to survive, we humans also thrive with connection.
            Sign up today and start chatting!
        </p>
    </div>

    <!-- Register Form -->
    <div class="register-box">
        <h3>Create an Account</h3>
        <form id="register-form">
            <input id="username" type="text" name="username" placeholder="Username" required>
            <input id="password" type="password" name="password" placeholder="Password" required>
            <input id="confirm-password" type="password" name="confirm-password" placeholder="Confirm Password" required>
            <button type="submit">Register</button>
        </form>
        <div id="error-message" class="error-message"></div>

        <!-- Small Login Link -->
        <div class="login-link">
```

```
35          <div class="login-link">
36              Already have an account? <a href="index.html">Log in here</a>
37          </div>
38      </div>
39  </div>
40
41      <script>
42          const form = document.getElementById("register-form");
43
44          form.addEventListener("submit", async (e) => {
45              e.preventDefault(); // Prevent default form submission
46
47              const username = document.getElementById("username").value.trim();
48              const password = document.getElementById("password").value.trim();
49              let confirmPassword = document.getElementById("confirm-password").value;
50
51              // Password match validation
52              if (password !== confirmPassword) {
53                  showErrorMessage("Passwords do not match.");
54                  return;
55              }
56
57              try {
58                  const response = await fetch("/register", {
59                      method: "POST",
60                      headers: { "Content-Type": "application/json" },
61                      body: JSON.stringify({ username, password })
62                  });
63
64                  const data = await response.json();
```

```
63
64                  const data = await response.json();
65
66                  if (!response.ok) {
67                      showErrorMessage(data.message); // Show error message from server
68                  } else {
69                      window.location.href = "/index.html"; // Redirect on success
70                  }
71              } catch (error) {
72                  showErrorMessage("An error occurred. Please try again.");
73              }
74          });
75
76          function showErrorMessage(message) {
77              const errorContainer = document.getElementById("error-message");
78              errorContainer.textContent = message;
79              errorContainer.style.display = "block"; // Show error message
80          }
81      </script>
82
83  </body>
84  </html>
```

public > index.html > html

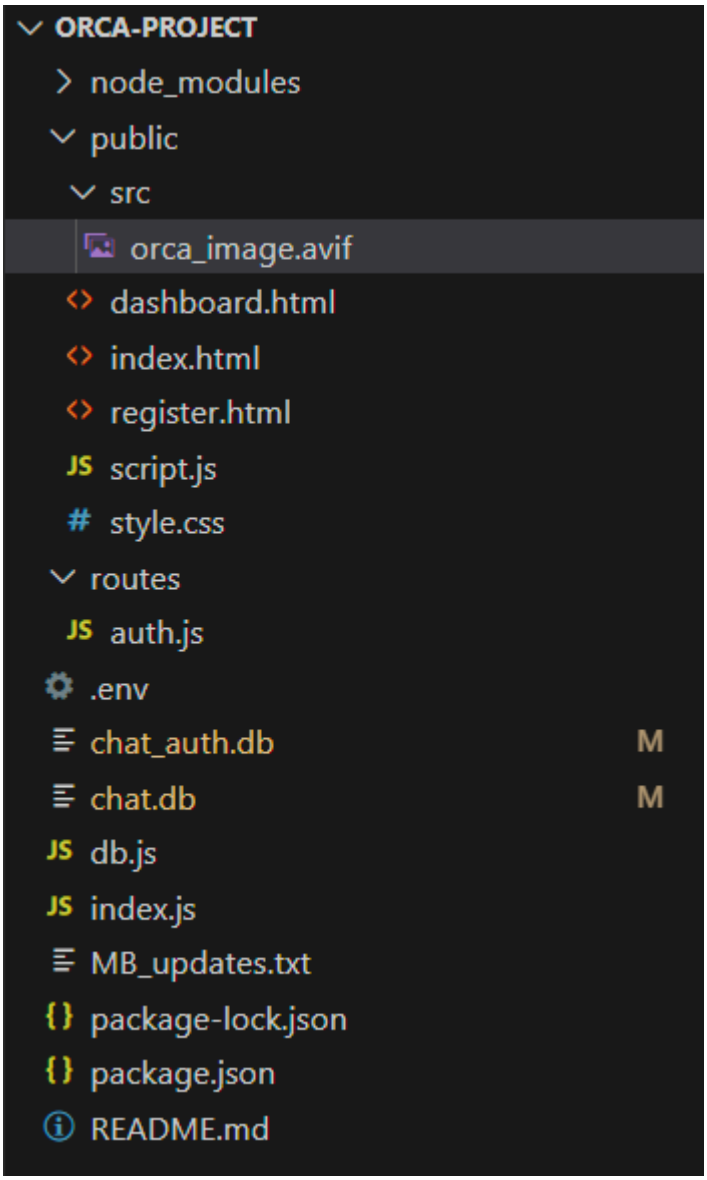
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ORCA Login</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<div class="container">
    <!-- Welcome Message -->
    <div class="welcome-box">
        <h2>Welcome to ORCA!</h2>
        <p>
            ORCA stands for Online Realtime Chat Application, designed for hospitalised children.
            Our goal is to provide a safe and friendly chat space for long-term hospitalised kids.
            <br><br>
            Our mascot, the <strong>orca whale</strong>, represents social connection—just like humans, orcas need
            companionship to thrive. Join us and stay connected!
        </p>
    </div>

    <!-- Login Form -->
    <div class="login-box">
        <h3>Login</h3>
        <form id="login-form">
            <input id="username" type="text" name="username" placeholder="Username" required>
            <input id="password" type="password" name="password" placeholder="Password" required>
            <button type="submit">Login</button>
        </form>
        <div id="error-message" class="error-message">Invalid credentials. Please try again.</div>

        <!-- Small Register Link -->
        <div class="register-link">
            New here? <a href="register.html">Create an account</a>
```

```
        <div class="register-link">
            New here? <a href="register.html">Create an account</a>
        </div>
    </div>
</div>
    <script>
        document.getElementById("login-form").addEventListener("submit", async (e) => {
            e.preventDefault(); // Prevent default form submission

            // Get username and password input values
            const username = document.getElementById("username").value;
            const password = document.getElementById("password").value;

            try {
                const response = await fetch("/auth/login", {
                    method: "POST",
                    headers: { "Content-Type": "application/json" },
                    body: JSON.stringify({ username, password })
                });

                if (!response.ok) {
                    const data = await response.json();
                    showErrorMessage(data.message); // Display error message from server
                    return;
                }

                // Redirect user to dashboard on successful login
                window.location.href = "/dashboard.html";
            } catch (error) {
                console.error("Error during login:", error.message);
                showErrorMessage("An error occurred. Please try again.");
```

```
                showErrorMessage("An error occurred. Please try again.");
            }
        });

        function showErrorMessage(message) {
            const errorContainer = document.getElementById("error-message");
            errorContainer.textContent = message;
            errorContainer.style.display = "block"; // Show error message
        }
    </script>

</body>
</html>
```

public > dashboard.html > html > body > script > switchChannel

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chat Room</title>
    <link rel="stylesheet" href="style.css">
    </style>
</head>
<body>
  <div class="chat-container">
      <div class="header">
          <select id="channel-select">
              <option value="general">General</option>
              <option value="gaming">Gaming</option>
              <option value="movies">Movies</option>
          </select>
          <button onclick="switchChannel()" class="channel-switch" id="switch-channel">Switch Channel</button>
          <button class="logout" onclick="logout()">Logout</button>
        </div>
      <div class="chat-box" id="chat-box">
        <ul id="messages"></ul>
      </div>
      <form id="form">
      <div class="input-box">
          <input type="text" id="input" placeholder="Type a message...">
          <button id="send-button">Send</button>
      </div>
    </form>
  </div>
    <script src="/socket.io/socket.io.js"></script>
    <script src="./script.js"></script>

    <script>
      // When the client switches channels this event will run
      function switchChannel() {
        let newChannel = document.getElementById("channel-select").value;
        socket.emit('join channel', newChannel, username, (response) => { // the client joins the new channel
          console.log(response);
          document.getElementById("messages").innerHTML = ""; // Clear messages when switching
      });
      channel_ID = newChannel;
      }

      function logout() {
    // You may want to clear the user's session or perform any necessary logout actions here
    // For example, if using Passport.js, you might do something like:
    // socket.emit('logout', username); // if you're handling logout via socket

    // Then redirect to the index.html page
    window.location.href = 'index.html';  // Redirect to index.html
  }
    </script>
</body>
</html>
```


ORCA-PROJECT
node_modules
public
src
orca_image.avif
dashboard.html
index.html
register.html
script.js
style.css
routes
auth.js
.env
chat_auth.db
M
chat.db
M
db.js
index.js
MB_updates.txt
package-lock.json
package.json
README.md