2021

Online Collaborative whiteboard

CE301 – CAPSTONE PROJECT

AUTHOR: JOSHUA BOUVIER | REG NO: 1805981

SUPERVISOR: RICCARDO POLI

SECOND ASSESSOR: MORTEZA VARASTEH

Acknowledgements

Special thanks go to my supervisor, Riccardo Poli for his weekly meetings and feedback as well as guidance when asked.

Abstract

Whiteboards are used in classrooms, businesses and, to use myself as an example, right at home. They are an invaluable tool when it comes to presenting new information and developing new ideas. In addition, they offer the opportunity to encourage collaboration and collaborative learning. Moving whiteboards online opens up numerous possibilities, including the option for multiple people to work on the same whiteboard simultaneously, access in remote locations and the flexibility to add extra functions/actions such as undo, redo and rescaling.

The purpose of this project was to create a whiteboard that could be accessed over the internet delivering real-time drawing of lines and objects. With the extension of a permission system, making it view-only, local-edit or fully interactive for everyone who is using this. It is also compatible with mobiles as well as desktops, enabling use across different technologies.

The whiteboard enables users to draw lines and objects, add images and text, undo, redo, create and join whiteboard sessions, set permissions, resize objects. In addition, the canvas itself is designed with a clean and simplistic user interface (UI).

The whiteboard uses a variety of the latest libraries and frameworks such as React, node.js, MongoDB, socket.io and Konva. These have allowed me to have a snappy frontend and modern backend, which can handle a significant amount of users viewing a whiteboard at once.

Table of contents

Table of tables

Table of figures

1 Main text

1.1 Literature review

1.1.1 Introduction

This literature review will look at the use of interactive whiteboards and the growing use of collaborative online tools as a support to learning and working. It will also consider the online collaborative interactive whiteboards currently available.

1.1.2 Context

This project focuses on the development of an effective online collaborative whiteboard. With a move towards online working and learning, educators and businesses may be starting to rethink how they work and be looking for solutions that enable them to work collaboratively online.

Whiteboards can be found in many contexts, such as in education, business and for personal use. They can perform the function of providing information delivered by presenters and can also be viewed as learning tools which offer the opportunity for group collaborations.

Personally, I own a wall-mounted whiteboard which I frequently use to scribble on to help me problem solve. Whilst this has been incredibly helpful, in an online context there is the ability to save and return to a whiteboard later. Personally, I'm also an interactive learner. Being presented with a slideshow on an interactive whiteboard I feel results in a missed opportunity. With the potential to deliver directly to a student, via provided equipment or their own electronic devices, I believe offers huge potential for improvement. I also believe they offer opportunities for the workplace.

1.1.3 Use of interactive whiteboards

Whilst whiteboards can be used in multiple environments, such as for personal use and business use, most studies focus on their use in an education environment. This is backed up by D. G. &. D. M. &. D. A. &. V. Door in the statement "While there is a great deal of research available on the effects of IWB use in the classroom a parsimonious model of the environmental or contextual factors will help to enhance researchers' understanding of the results." [1]. They are commonly used by teachers, especially in the UK, due to a largescale push by the government. There was a £15 billion pound scheme to install interactive whiteboards in all primary and secondary schools [2], which has resulted in them now being an important part of the classroom.

Whilst the £15 billion drive was to install wall-mounted interactive whiteboards – this technology can be repurposed for online collaborative use too, allowing students to access the whiteboard on laptops or tablets opening up opportunities for better interaction. As seen in this study when students were allowed to use their own devices, the result was easy access at a low cost. [3]

1.1.4 Use of collaborative online tools

Online collaborative tools are increasingly used in education institutions due to a general reduction of budgets. They are turning to more modern low-cost alternatives to support teaching and learning using collaboration tools such as Google Docs, sheets and slides [4]. In a survey carried out on Google Docs in particular, it was shown that the majority of students gave Google Docs a rating of 4 or 5 out of 5, when asked about their experience using of the tool [5]. This supports the view that collaboration tools can be very effective in a learning environment.

I was unable to find any significant published research on the extent to which collaborative online tools are being used in business. I have therefore added a number of online articles to my literature review.

Even before the impact of the Coronavirus Pandemic, it has been suggested that the use of collaborative tools in the workplace was already significant. There are a number of online sites citing a survey conducted by PGi in 2017 which indicated that 88% of industry professionals were using online collaboration tools at least once a week [6] [7]. More recently, TrustRadius reported on a survey conducted in April 2020 which showed that out of 'over 2,000 members of the TrustRadius community, 15% planned to increase their software spending as a result of COVID-19. Nearly 60% of those businesses planned to purchase collaboration software.' [8]

To further support the suggestion that 2020 has been a particularly good year for collaboration software, Joe O'Halloran, Computer Weekly, 11 Feb 2021 [9] states "For collaboration software, 2020 was a year like no other". His claims are based on research from Aternity [10] which demonstrates a significant increase in the use of Microsoft Teams and Zoom over the past year. He also suggests that "a new normal of the hybrid workplace is now establishing itself." So, it does appear that collaboration tools will increasingly have a place in business.

1.1.5 Online collaborative whiteboards

There are a number of products already available which provide online whiteboards. I selected the following to review. AWW App (https://awwapp.com/) [11] (This will be merged with Miro however at the end of July, 2021), Miro (https://miro.com/app/dashboard/) [12], Ziteboard (https://app.ziteboard.com/) [13] and Whiteboard Fox (https://whiteboardfox.com/) [14] among others.

Here is a top level comparison of them:

Provider	Live drawing of objects	Sharing behind login/pay wall	Advanced permissions	Intrusive popups asking for login when accessing website
My capstone	Yes	No	Yes	No
AWW App	No	No	No	Yes
Miro	No	No	No	Yes
Ziteboard	No	Yes	Yes	No
Whiteboard Fox	No	No	No	No

Real-time streaming

As seen above, one of the main limitations with the selected products is that they don't offer real-time streaming of drawing. The image only updates, once the drawing has finished. This results in the object suddenly popping onto the screen, something that in my opinion can feel quite jarring. It's also more engaging to see content illustrated in real time rather than waiting for it to pop into the screen.

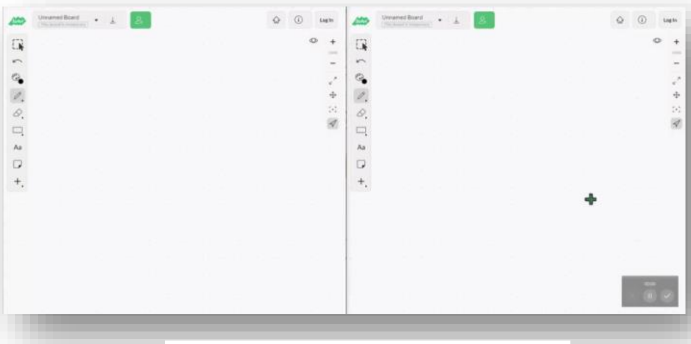
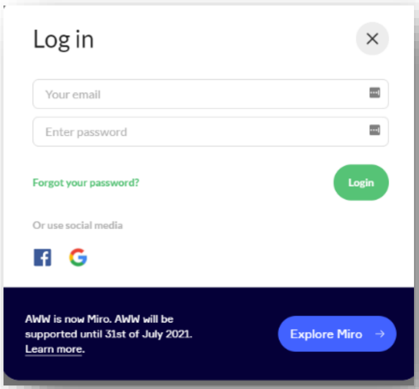


Figure 1 Drawing on 2 whiteboards over the internet with AWW APP

Login/pay walls

Others have login or payment walls which are required to access important features. This can be frustrating as they require every viewer to also either create an account or make unnecessary clicks to clear the popup.



AWW is now Miro. AWW will be supported until 31st of July 2021. Learn more.

Figure 2 Login promp from AWW APP

Permissions

All whiteboards have the common ability to be shared with another user, where the main user can give additional users permission to just view or edit it if they log in. One particular drawback with can be once permissions have been set, these cannot later be changed. This could prove frustrating, as it doesn't allow for change of approach during the presentation. For example, you might suddenly decide you want to let someone write an answer on the whiteboard. This is a particular issue with Ziteboard, where the whiteboard creator also has to log in to be able to share the whiteboard. This could result in a less pleasant experience for a user, as they may not be willing to hand over their data or may not have the time to faff around making an account to scribble some notes.

1.1.6 Conclusion

This literature review suggests that collaboration tools are in demand across multiple sectors. It also suggests that demand is continuing to grow, as we change the way we work and learn.

Whilst online collaborative whiteboards are emerging it can be seen there is still room for improvement. My project will focus on achieving an effective online collaborative whiteboard which in addition addresses the issues raised, offering real-time streaming, no login/paywall and advanced permissions.

1.2 Project goals

1.2.1 Main goals

The main goals of my project are the following:

- **Usability**: Most critically, to design a blank canvas which is interactive, responsive and efficient. This is important because the canvas needs to be suitable for all environments, including scenarios where the user's machine may be quite old or slow, or potentially even a mobile device such as a phone or tablet.
- **Functionality**: The ability to draw lines, circles, squares, images and text onto a canvas.
- **Functionality**: Objects to be editable. To elaborate, editable with regard to their position, size, colour and content. The object should also be able to be deleted.
- **Functionality**: The ability to rescale and move the whiteboard, allowing for an unlimited size for the whiteboard.
- **Storage**: The whiteboard must be stored in a database, so that it can be saved and loaded at a later date and across server restarts. This also allows the whiteboard to have an owner.
- **Permissions**: To include a permissions system, allowing the owner of a whiteboard to make it view-only by other users or editable by all. An owner should also be able to set individual permissions of users that are logged in and change these permissions at any point.
- **Authentication**: To include a user authentication system. Whilst this project focuses on an online collaborative whiteboard to demonstrate permissions and ownership of whiteboards, a basic authentication system is still required. However, this will be without a need for an email verification system or encryption as it's a proof-of-concept.
- **Live streaming**: A networked ability. This encompasses all actions such as drawing, clearing, undo and redo being actioned across all users who are currently viewing the whiteboard at the same time. This will allow other users to see objects being drawn in real-time (specifically not after completion, but as they are being drawn).
- **Design**: An elegant, accessible UI.

1.2.2 Stretch goals

- **Functionality**: Live viewer list to allow you to see who is currently viewing the whiteboard
- **Mobile support**: This would allow for more accessibility for scenarios where a computer might not be available to everyone who wants to view a whiteboard.

1.3 Legal and Ethical

1.3.1 Legal

The only key potential concern is the lack of encryption, making passwords vulnerable. The password would be at risk of being leaked which wouldn’t make the product GDPR compliant. More security-related work would be required for it to be secure for production.

1.3.2 Ethical

I have tried to make sure the product is accessible as possible by keeping the interface simple. The idea being that the product is still accessible to those who may have a less powerful device

1.4 Design style

UI

I was aware of a few popular design styles when designing my UI. They were:

- Material design, a scheme followed by Google
- Flat design, a scheme followed by apple
- Bootstrap, a fairly basic open-source scheme

After looking over all the designs, my personal preference was material design. This was due to the fact I felt it offered the clearest presentation out of all of them.

I decided I wanted it to be minimalistic yet fully accessible at the same time.

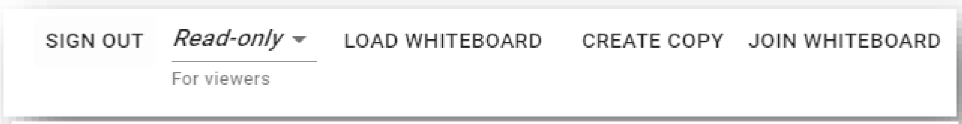


Figure 3 Topbar of whiteboard

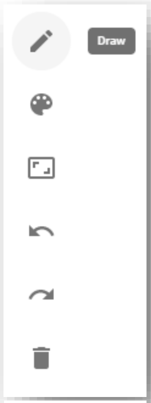


Figure 4 Sidebar of whiteboard

Whiteboard actions

Another crucial part of the design was to free-up as much screen space as possible for the whiteboard itself. After some initial designs I choose an icon-only approach for the tools to manipulate the whiteboard, positioned on the top left side. All icons also have tooltips to avoid any confusion regarding their function.

User actions

For the user actions, I decided to place them at the top right of the whiteboard. After initially trialling them as icon buttons, I changed them to text buttons. This was because it was difficult to design a clear button to represent actions like loading saved whiteboards and creating a copy of a whiteboard. To keep the number of visible buttons to a minimum on the user actions tab, I also moved 'new Whiteboard' to the load whiteboards UI. All of them are text buttons other than the global permissions, which is a dropdown. All buttons are positioned in line with eachother.

Displaying saved whiteboards

The next important UI component was how I displayed saved whiteboards. Off the bat I knew I wanted a preview image of the whiteboard to be displayed with the name below. Other than that I didn't want any extra UI pieces or unnecessary descriptions.

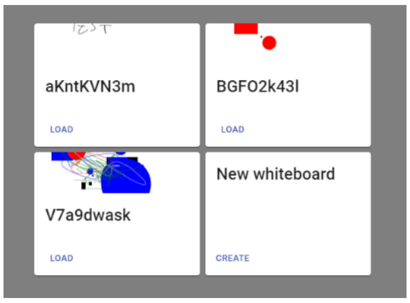


Figure 5 Load whiteboards menu

Join, login and signup

Join whiteboards, login and signup all share the same design. When you press on the button a popup appears, asking you to input information, which is laid out in the same manner across all the popups.

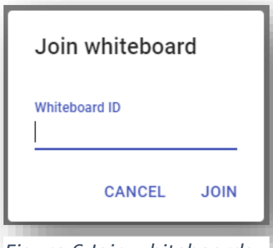


Figure 6 Join whiteboards menu

Conclusion

I believe the result is a clean look with little room for confusion regarding what each button does.

1.5 Technical documentation

Technical documentation can be found here:
https://cseegit.essex.ac.uk/ce301_2020/ce301_bouvier_joshua_l_j/-/wikis/home

A video of the project being demonstrated can be found here:
https://www.youtube.com/watch?v=J1ZBVgXbgzo

1.6 Predevelopment

Frontend

Predevelopment involved researching what tools would be the best. I knew I wanted a modern and interactive frontend. So, I explored the big 3 frontend frameworks, React [15], Vue.js [16] and Angular [17]. I decided to go with React as I had some previous experience with it and I knew that it is incredibly popular [18] with very good documentation. Below is my comparison of my options.

Table 1 Frontend software comparison

Frontend software	Popularity	Speed	Documentation	Syntax	Familiarity
React	Huge	Decent	Lots of documentation	Good	Some
Angular	Huge	Decent	Lots of documentation	Good	Never used before
Vue	Small, but emerging	Quick	Some documentation	Good	Never used before

Backend

Next, I chose the back-end server. My 3 main options were PHP [19], node.js [20] and golang [21]. Making a comparison between PHP and node.js, node.js comes out the clear winner. Node.js is faster than PHP [22] and in my opinion is much cleaner. When comparing node.js to golang, I wasn't too comfortable with the documentation for golang so I decided with node.js.

Table 2 Backend software comparison

Backend software	Popularity	Speed	Documentation	Syntax	Familiarity
PHP	Huge	Slow	Lots of documentation	Ugly	Some
node.js	Huge	Decent	Lots of documentation	Clean	Knowledge of JavaScript
golang	Less popular	Quick	Some documentation	Clean	Never used before

Database

I then chose which database to use. I had a large array of options, these being MySQL [23], Oracle [24] Microsoft SQL Server [25], PostgreSQL [26], MongoDB [27], MariaDB [28]. In the end I choose mongoDB for the fact it used noSQL [29], meaning how I inserted the data of an object into the database didn't have to be the same for all objects. For example, text wouldn't have a radius, and a circle wouldn't have a XY size. See below my comparison between them.

Table 3 Database software comparison

Database management software	Popularity	Speed	Documentation	Relevancy	Familiarity	Price

MySQL	Very standard	Slow	Lots of documentation	Not particularly suitable for my purpose not for a dynamic website.	Some experience with this before	Free
Oracle	Popular across businesses	Slow	Lots of documentation	Not particularly suitable for my purpose not for a dynamic website	No experience with this before	Costs money
Microsoft SQL Server	Developed by Microsoft, fast and stable.	Fast and stable	Lots of documentation	Not for a dynamic website like the whiteboard	Limited experience with this before	Costs money
PostgreSQL	Semi popular, emerging database tool	Fast	Limited documentation	Incredibly scalable for if there was a surge on hits to the website	No experience with this before	Free
MongoDB	Semi popular, emerging database tool	Fast	Decent documentation	Incredibly versatile, bit of hassle to setup	No experience with this before	Free
MariaDB	Semi popular, emerging database tool	Fast & stable	Limited documentation	Quite a new SQL type, but a better version of MySQL essentially.	No experience with this before	Free

Sockets

My final choice of software to use was deciding how clients communicate to the server. Some brief research went into this, but I quickly settled upon socket.io [30] based on its great documentation and high popularity. I could have implemented raw web sockets which would have been faster [31], but this would have been considerably more complicated which I decided against.

1.7 Development narrative

1.7.1 October

1.7.1.1 The canvas

Development started initially with the creation of the MVP. This had a core element being the canvas. Initially I started my project by using the basic HTML canvas element. Here is a link to some documentation about the element, and the kind of things you can do with it, https://www.w3schools.com/html/html5_canvas.asp [32]. The canvas element, as implied by the name, provides a blank canvas to which I could call functions on to draw lines and objects.

Line drawing and colour

The next step was to be able to draw in the canvas. I decided to start with simply drawing a line.

To do this I added an event listener which detected when the mouse was pressed, 'mousedown'. I made a variable, 'isDrawing', to true to signify that the mouse is being pressed, and then I got the coordinates of the current position of the mouse and created a function which I called 'drawLine' that drew a tiny line in that position. I had a further event listener for mouse movement, 'mousemove', which also called the drawLine function after every movement of the mouse whilst my variable 'isDrawing' was true. I then had a further event listener for if the mouse is released, 'mouseup', which would set the 'isDrawing', mentioned from previously, to false. This would mean when the 'mousemove' event was being triggered by mouse movement, it would not draw lines as the mouse has not been pressed. Whilst it was effective in making a line, it looked quite jagged if you moved the cursor slowly. This was due to it being many small lines rather than a single large one. This is addressed later on in development.

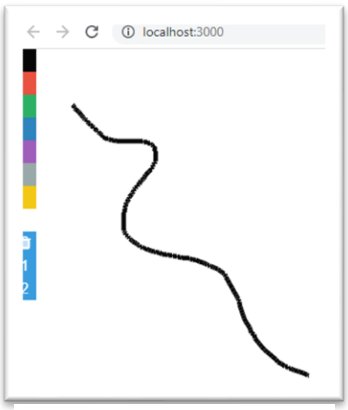


Figure 7 First version of whiteboard

I then added 7 colour buttons, which stored hex values of their respective colour. When a colour was pressed, I would update a variable which stored the current stroke. This variable was used in the creation of lines when its colour was set, 'line.strokeStyle = stroke'. I also added a clear button to clear the whiteboard.

1.7.1.2 Networking

My next step was to implement networking, so changes made to the whiteboard could be seen across different devices. To do this I made a JavaScript file that is run by node.js, which simply starts a socket server (using socket.io) that listens to 3 net messages. The net messages are 'drawing', 'clear' and 'joinRoom'. Going back to the client files, I made the client connect to my socket server when the page is loaded and I amended my 'drawLine' function to make a net message every time it's called, labelled 'drawing', which passed the data of a line being drawn. This data being the cursor's coordinates and colour. It would then send this back to every other client connected and call the 'drawLine' function with that data. I now had my very first implementation of an 'online whiteboard'.

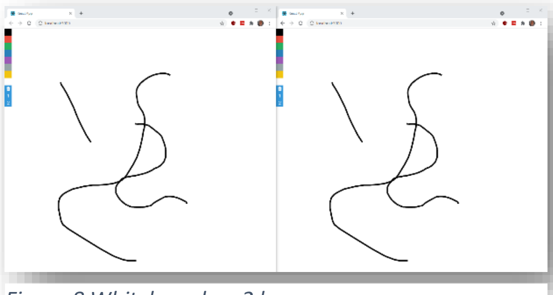


Figure 8 Whiteboard on 2 browsers

I then added a net message to my clear whiteboard function labelled ‘clear’, that triggered a function on all other clients to clear the whiteboard for there screen.

1.7.1.3 Multiple whiteboards

I then added a basic system, to simulate what it would be like to have multiple sessions at once on a different whiteboards. The id’s of the whiteboards were hard coded to “room1” and “room2”. I added 2 buttons labelled 1 and 2, which when you joined it set a variable to the id’s previously mentioned. That variable was then also sent in the ‘drawLine’ function with the data of the line. Once received by the server, it uses that data to send the line to all users who were that room.

This completed my MVP.

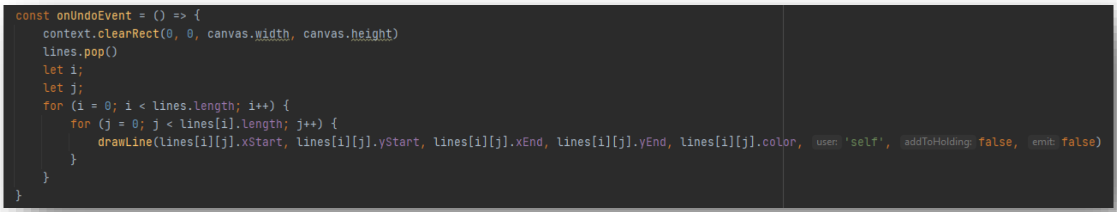
1.7.2 November

1.7.2.1 Server storage

My next step was to implement a temporary storage of the whiteboard for while the server was running (note that it is not into a database, yet). To do this I created 2 arrays, both on the server and client. The first array’s purpose was to store lines that were being drawn, and the 2nd array was to store lines that were completed. For the first array I added the data of the line for every time ‘drawLine’ was run. I then created an addition net message that was called when an object had finished drawing, ‘lineCompleted’, which pushed the in-progress object to the completed object array. This design had a few short-falls though and it also did break the room system, as it was only storing this data for the first whiteboard for simplicity. It was improved later on in development. Another issue being that it only allowed for one object to be drawn at once for all clients viewing the whiteboard, as I wasn’t giving an ID to each line (whilst it was technically possible for multiple users to draw at once, any drawings made after the initial object was started would overwrite each other and merge data, causing a mess).

1.7.2.2 Undo

This did allow me to implement an undo feature however, which worked by quite simply popping the latest inserted line into the completed objects, and redrawing the whiteboard using the new table of data. It was also applied to the server meaning undo would also have an effect on its copy of the whiteboard.



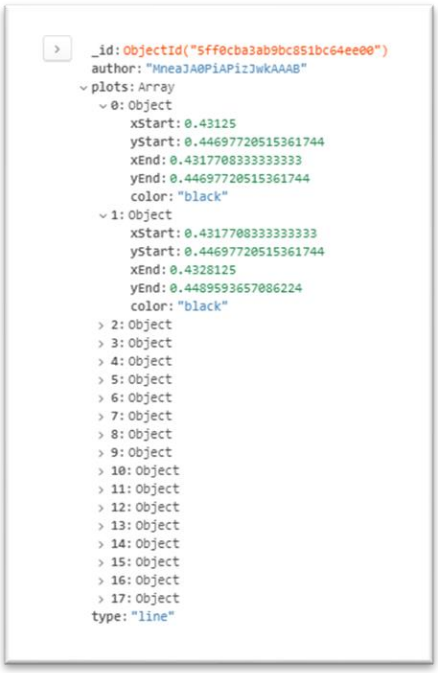
```
const onUndoEvent = () => {
    context.clearRect(0, 0, canvas.width, canvas.height)
    lines.pop()
    let i;
    let j;
    for (i = 0; i < lines.length; i++) {
        for (j = 0; j < lines[i].length; j++) {
            drawLine(lines[i][j].xStart, lines[i][j].yStart, lines[i][j].xEnd, lines[i][j].yEnd, lines[i][j].color, user: 'self', addToHolding: false, emit: false)
        }
    }
}
```

Figure 9 Original undo function

The reason why I didn’t jump straight into a database to store whiteboards, was because I had chosen a database system I was unfamiliar with, mongoDB, which required some time researching and reading documentation [33] to fully understand how to use it. I also wanted to see how I would go about implementing an undo system before the more time-consuming task of connecting it to a database, so I would know what the best way to store the data would be.

1.7.2.3 Database storage

Now I had undo completed, I connected my server to a mongoDB server hosted by mongoDB themselves, and using a tool called MongoDBCompass I connected to the database and created 2 databases. One for the whiteboards and one for the users. I deleted the completedObjects array on the server, which previously was used to store completed objects but kept my array for in-complete objects. There wasn't any purpose for this however on the server as it didn't do anything other than provide an overhead, but the old system still remained in place on the client.

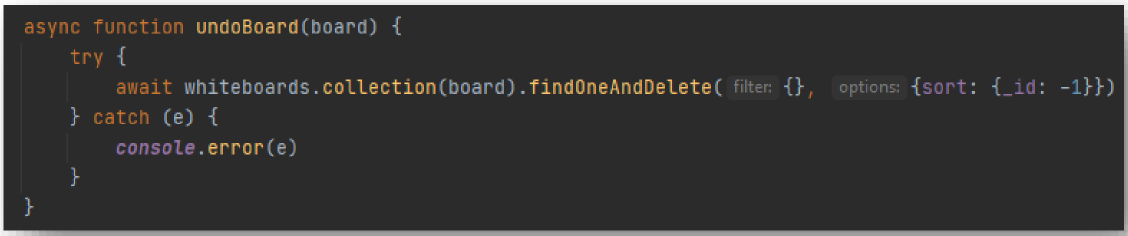


```
_id: ObjectId("5ff0cba3ab9bc851bc64ee00")
author: "MneaJA0PiAPizJwkAAAB"
plots: Array
  0: Object
     xStart: 0.43125
     yStart: 0.44697720515361744
     xEnd: 0.43177083333333333
     yEnd: 0.44697720515361744
     color: "black"
  1: Object
     xStart: 0.43177083333333333
     yStart: 0.44697720515361744
     xEnd: 0.4328125
     yEnd: 0.4489593657086224
     color: "black"
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
  8: Object
  9: Object
  10: Object
  11: Object
  12: Object
  13: Object
  14: Object
  15: Object
  16: Object
  17: Object
type: "line"
```

Figure 10 First version of a database entry for an object

When a line was completed, it triggered a function on the server called 'addLineToBoard' which quite simply inserted the line provided into the relevant whiteboard collection. The whiteboard ID was defined by the 'room' variable previously mentioned on the client, which was also sent with the data about the line.

My next step was to reimplement undo, and to do this I achieved it in a similar way to before, where I deleted the latest item inserted into the database.



```
async function undoBoard(board) {
    try {
        await whiteboards.collection(board).findOneAndDelete( filter: {},  options: {sort: {_id: -1}})
    } catch (e) {
        console.error(e)
    }
}
```

Figure 11 Server-side undo function

1.7.3 December

1.7.3.1 Load whiteboard

I also added the ability to load the whiteboard from the database when a user views that whiteboard. It would work when the client access the website for the first time or refreshes. The server sends a net message with the whiteboard ID defined on the client, and the server sends back a net message for each object in the board, with the data of the line. It was done like this to avoid sending huge amounts of data in one chunk. A function was made on the client to handle the net message, which would simply add it the completed objects table, and then order a re-render of the canvas.

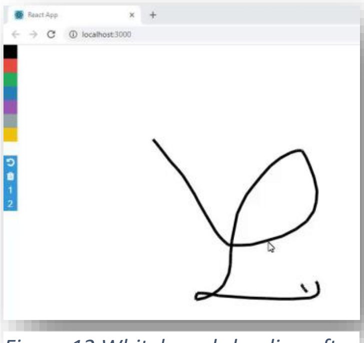


Figure 12 Whiteboards loading after refresh from the database

1.7.3.2 Squares, circles and text

At this point I looked to expand from just being able to draw lines. So, I added a very basic version of the ability to draw squares, circles and text. To do this I added some extra buttons each symbolising a different object to draw. For example, if the square was selected, it would set a variable which stored the current tool to "square". In the same style as 'drawLine', it drew a square based on the current position of the mouse minus the position of where initially clicked. A key difference between the way a shape and line is calculated is that with a shape, when you are resizing it you don't want to see every stage of the size as you are sizing it (as it would turn into a big blob), where with a line you want to see every stage your cursor has been. I had to modify how I stored the shape objects to make it override the values of the old object in the in-complete objects array mentioned before. The same methodology was applied to drawing circles. Text was similar, however the data was hardcoded to be "testing123", as I had not made a UI component for changing the text.

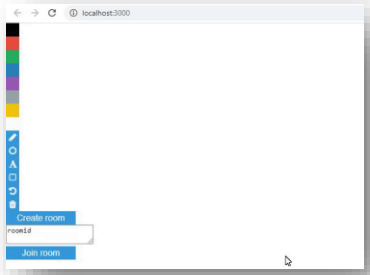


Figure 13 Drawing of objects other than lines (squares, circles and text)

The next step was to make this work over the network. This only required some minor tweaking on the server, to the function that sent the data from the database when a user loads a whiteboard. On the client, in a similar fashion to the 'drawLine' function, I added net messages that triggered the relevant drawing function.

1.7.4 January

1.7.4.1 Project review and significant redo

At this point I realised I had reached a limitation of the HTML canvas, and I needed to reassess my project. A main issue that presented itself was the inability to edit shapes without complicated mathematical overhead and deeply complex element manipulation. Another issue which I felt was important was the inability to zoom/move around the whiteboard. I decided to explore different canvas frameworks, such as Konva and Fabric.js. In the end I chose Konva based on some research [34] and its better compatibility with React.

I deleted the vast majority of the client code base, and reimplemented drawing lines, squares, circles and added an eraser. Networking, undo, redo, clear and joining whiteboards all had to be temporarily removed, however. This is because the structure and syntax of the project had significantly changed, as well as the way I stored objects in the client for later reference. Previously I used the function .push to add objects into the array, but with konva I needed to cause a re-render after every change made. This meant I had to use a core feature of react, states. How states work is that you define a state, for example [value, setValue], which you call 'setValue' with your new value to change the 'value' variable. Calling 'setValue' also triggers a re-render.

Konva also brought a change in the layout of how objects were stored, for the better. Previously I had to have duplicated data, such as the colour, stored for each individual point of a line. In the new iteration I took the opportunity to be more data efficient.

My next move was to reimplement networking. This task wasn't too much of a challenge as I was able to reuse the server-side code completely and I knew how I was going to structure my net messages from the client. At this point, I did run into a major technical issue which took a few weeks to solve. Essentially, performance was severely degraded on both the client and server the more that was drawn. It transpired that a re-render of the whiteboard (caused by drawing) was causing a new socket connection to be opened. This also meant that when a new socket is opened it sends data to all the previously established sockets with the lines you are currently drawing, causing another re-render etc. Essentially this caused memory leak, which was hard to pin down, and even when I had pinned it down proved to be a challenge to solve. To counter this, I had to do further research into react and found I should be defining clean-up behaviour if the page is getting re-rendered. So, I made it close the old connection and make a new one but keep the socket open to the browser after every re-render.

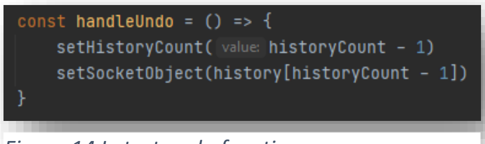
Next, I re-added clearing the whiteboard and the basic ability to join a pre-defined whiteboard session. Both of these were easy implementations largely using previous code, other than the clearing which I made a new function for.

Then I re-added streaming from the database again for loading a whiteboard. This was an easy one-line implantation.

1.7.5 February

1.7.5.1 Adding undo and redo

I re-added undo, but I also added, for the first time, redo. These proved fairly complicated however, due to the nature of re-renders and edge cases. An issue I initially experienced was that I had a state for both the counting of steps made in a whiteboard and an array for a copy of all objects on the canvas at that step for later reference. A step goes forward by one every time something is drawn. It can traverse forwards and backwards, if you press undo and redo. Having these after each other caused excessive re-renders, consequently causing performance issues and sometimes data loss. After some more researching of react [35] I found I had to use Ref's instead. These are similar to states, as in they store data across re-renders, but don't cause re-renders when the value is set.



```
const handleUndo = () => {
    setHistoryCount( value: historyCount - 1)
    setSocketObject(history[historyCount - 1])
}
```

Figure 14 Latest undo function

I then re-added undo into the database, but also added redo. To do this I used the history array made by the undo function and using the current step + 1 of the whiteboard, I reinserted the newest line as if it was drawling a new line. Any history going further than this step is then destroyed to conserve memory.

I then added object updating over the network, which I achieved by triggering a net message 'updateObject', which has the ID of the object and the new data, which is then just handled by a simple findOneAndUpdate query on the server to the database. The client then receives this data too and searches for this object in the completed objects table. Once it's found the object, it updates its data.

1.7.6 March

1.7.6.1 Adding the final UI

At this point I am nearing completion of all core aspects of the whiteboard and I felt comfortable making a proper UI, rather than the basic one I had temporarily made. I didn't want to build an entire UI framework on my own, as that would have been extremely time consuming and wouldn't be particularly relevant to my project. So, I decided to use Material UI. I decided my UI for the whiteboard actions would be on the left-hand side, like the temporary UI, but with exclusively icon buttons, and expanded the drawing options and colours into further side buttons. All the items are contained within a container, in a list. I then used a popover component to display the extra buttons for the drawing tools and colours.

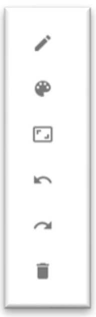


Figure 15 The sidebar UI

1.7.7 April

1.7.7.1 Authentication system

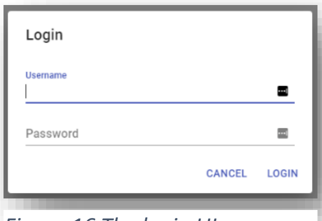


Figure 16 The login UI

At this point I decided to implement the authentication system. I didn't want to dedicate too much time to it, as it wasn't a core part of my project, but was necessary for me to be able to build a permission system. I looked online and found some services that handled this for me, being Auth0 [36] and Amazon's Cognito [37]. What both services would do is handle the entire authentication system, where you add a login button and it takes you to their website. They then take your username/password and encrypt it and give you a cookie. However, I found I had to continuously make an API request to their servers every X amount of seconds [38] to see if any users had signed up, which didn't offer the experience I wished to achieve. This was because when you sign up and immediately make a new whiteboard, the system would error, as it wouldn't be able to assign you owner of the whiteboard, as the servers wouldn't know you existed yet. The same issues would occur when detecting if a logged in user is viewing a whiteboard, as it wouldn't think this user was valid. Whilst this could be handled, it would have required extra code. Plus, I didn't want to be keeping track of 2 sets of data, being the data on Auth0/Amazon Cognito and the data on my database.

So, in the end I opted for making my own system. However, I didn't add any email authentication or encryption as it wasn't necessary, nor is the website encrypted (I haven't set it up for HTTPS) meaning encrypting the password would have been pointless, as it could have been intercepted already.

Once I had an authentication system, I needed to give it a UI. I decided to display the user action buttons displayed on the top right of the whiteboard as text buttons, rather than icon buttons. This was because I wanted these actions to be explicitly clear (i.e. preventing people from accidently signing out/making a new whiteboard). I then also made a login, signup and sign out button.

How this works is incredibly simple, when you enter a username and password, it checks if the username already exists. If it does not exist, it enters your details into the database. You can then login by typing in your details and the server checks your username and password. The unique key for a client is generated by mongoDB and is then stored on the client's browsers local storage. The client checks to see if a uniqueID is stored, and if so you are signed in. If you don't have this key you are not signed in. Signing out deletes this key from your local storage.

Figure 17 The load whiteboards menu

1.7.7.2 Ownership

I then made an ownership system for whiteboards. This meant that on the creation of a whiteboard, the server inserts key information about whiteboard into a new collection labelled 'permissions'. The whiteboard's ID, it's owner (if the user is signed in, else it does not have an owner), the global permission for the whiteboard (read or write), an array for permissions for individual users and a snapshot. The snapshot is used to store an image of the whiteboard, which is used when viewing saved whiteboards.

1.7.7.3 Loading whiteboards

Setting up ownership led to the ability to load whiteboards. To do this was actually quite simple. All I had to I do was make a query to the 'permissions' collections, and check for whiteboards with the owern's uniqueID. The server then returns all the data, mentioned in the previous paragraph, to the client, which I then loop through each whiteboard, which is then displayed by name and snapshot.

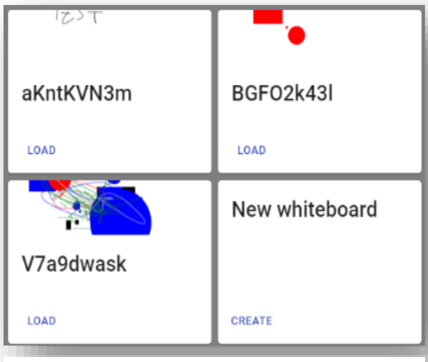


Figure 18 Load whiteboards UI

Although I had storage for snapshots at this point, I wasn't actually making any snapshots. To do this I considered a number of ways to accomplish this. I didn't want to be streaming large images from client to server and vice versa, so I knew whatever solution I had, required a reduction in the quality of the image. I thought about having a secondary server, which would process taking an image of a whiteboard, but this would have been quite excessive and taken unknown amounts of time to generate an image. I also thought about how I could store an image, as sending a PNG over a socket isn't realistic.

I browsed through Konva's documentation [39] and found the perfect function, toDataURL [40]. This allows a client to generate a string which is an image of the canvas in base64 standard. This was incredibly size efficient and was easy to store in the database. I did however have to generate this image on the client and send it over, which I decided to implement every time an object has been finished drawing.

1.7.7.4 User permissions

To accomplish this, I needed to keep track of users that are viewing the whiteboard. To explain how this works, I need to go into a little bit of detail as to how sockets work. Once a connection is made, it forms a socket. Sockets are disconnected when the page is refreshed, but not when the page is re-rendered. Whenever a user requests a new whiteboard or joins a whiteboard, a function called 'addUser' is called, which retrieves all the data from the database regarding that whiteboard, and adds that socket to the whiteboard room. Only people in that whiteboard room see messages from others in the same whiteboard room. Each whiteboard has its own room.

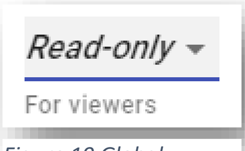


Figure 19 Global Permissions UI

In my 'addUser' function, firstly I insert the whiteboard ID into an object array with its global permissions, owner and user permissions. I then insert into another object array the socketID with its data as the whiteboard ID and username. When a user's socket disconnects or changes room, this data is then either removed or overwritten. When the global permissions is changed or a user is given or blocked permission, the object array storing this data is updated. When the data is sent to the client telling them the whiteboard is ready for them, in a net message called 'setupWhiteboard', it also passes on the global permission and their individual permission. The client then uses this to decipher if it will allow them to draw or display a notification. Every time the global permission is changed or a specific user permissions is changed, an update is sent to all clients with the new permissions. On the client it then re calculates if they are allowed to edit the whiteboard.

1.7.7.5 Viewer list

The permission system keeps track of the permissions people have in real-time. The viewer list piggybacks off the permission system, where it loops through the object array, tracking active sockets and checks to see how many clients are viewing the whiteboard in question. Once it has that tally, the server sends it off to the client in the 'setupWhitebord' net message. When someone joins, leaves or disconnects from a whiteboard a net message is sent with the updated viewer list to all clients viewing that whiteboard. For users that are not logged in, they are given the pseudo-name "guest". The data simply contains an array of every viewers username, which is passed to the UI component that displays the viewers. It loops through every user and displays the information seen in figure 20 Along side each user on the UI, I also added a block access and allow access button.

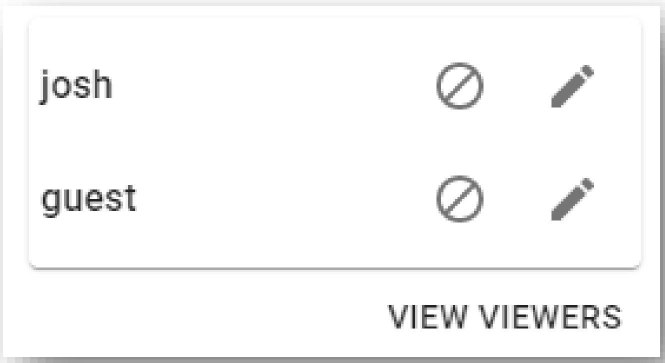
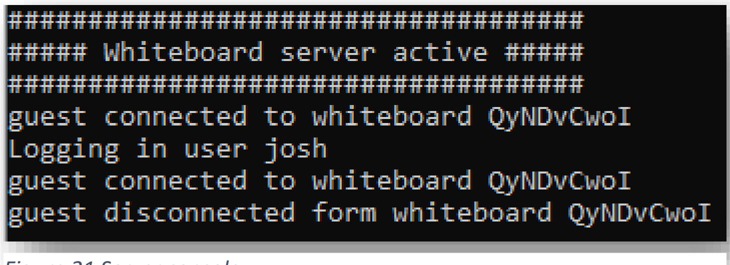


Figure 20 Whiteboard Viewers UI

1.8 Testing

1.8.1 General test for errors

For the entirety of the development cycle, I have continuously tested my code usually by myself, but sometimes with others, to catch stability issues and more edge-case issues, which I wouldn't necessarily notice when testing on my own. The server code also has comprehensive logs for if an error is detected, which are outputted to the console.



```
########################################
##### Whiteboard server active #####
########################################
guest connected to whiteboard QyNDvCwoI
Logging in user josh
guest connected to whiteboard QyNDvCwoI
guest disconnected form whiteboard QyNDvCwoI
```

Figure 21 Server console

I also conducted some extra tests to determine the quality of my project, and how it handles under load.

The website has been tested simultaneously on chrome and Firefox, opera and Edge. On all, it performs with no issues.

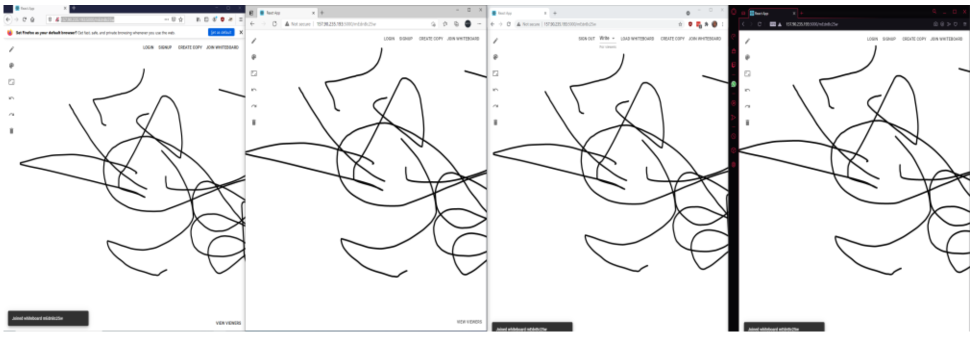


Figure 22 Images of a whiteboard being loaded on multiple browsers. From L-R (Firefox, Edge, Chrome, Opera)

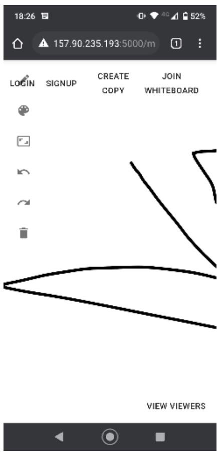


Figure 23 Screenshot of whiteboard on mobile (Android - Chrome)

Whilst the whiteboard can be viewed on mobiles, as seen in figure 23, the UI isn't optimized it nor does it allow you to draw.

All performance based testing was carried out on a virtualised server to ensure relative consistency with the following specifications:

- CPU: Intel® Xeon® Gold 2 core processor
- RAM: 4GB
- Storage: 40GB SSD

1.8.2 Max users test

I setup a test to simulate the average size of a classroom, which in an archived department of education official statistic from 2007 show the average size of a secondary school classroom was 21 [41]. I was able to sustain 30 active tabs to my whiteboard, from my computer. I therefore decided this would be a good baseline. I drew on the screen for about 30 seconds with no issues. This can be seen in the graphs in figure 24 which show a CPU spike to 8% usage at peak. If this was multiplied by 10, to make roughly 100%, this demonstrates that it could handle anywhere up to 300 people. For an incredibly low spec server like this, that's quite impressive. You can also see incredibly low network usage, peaking at just under 400 KBps showing there is no bottleneck there. It would be impossible to make a truly accurate test without actually gathering 30 people but this test serves as a rough estimate to what it might look like.

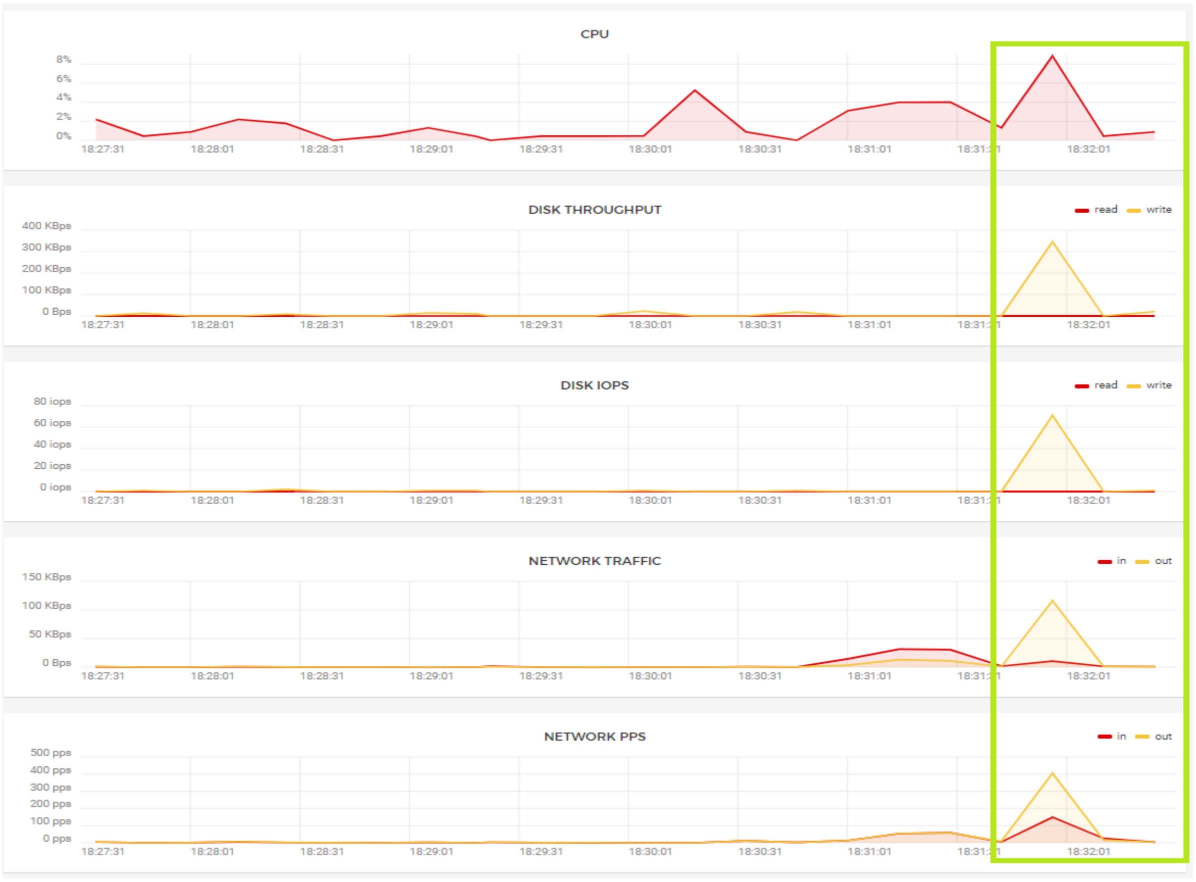


Figure 24 Virtual Server Usage Statistics

In another smaller test between 4 devices all drawing at once, no issues presented themselves.

1.8.3 Max objects test

Konva as a canvas framework shows it can handle thousands of shapes with relative ease, as shown in the documentation [42].

For my test I made a whiteboard with over 200 objects and no performance issues were presented. See figure 25. This test also included loading all the objects from the database, which occurs when you access a whiteboard for the first time or refresh the page.

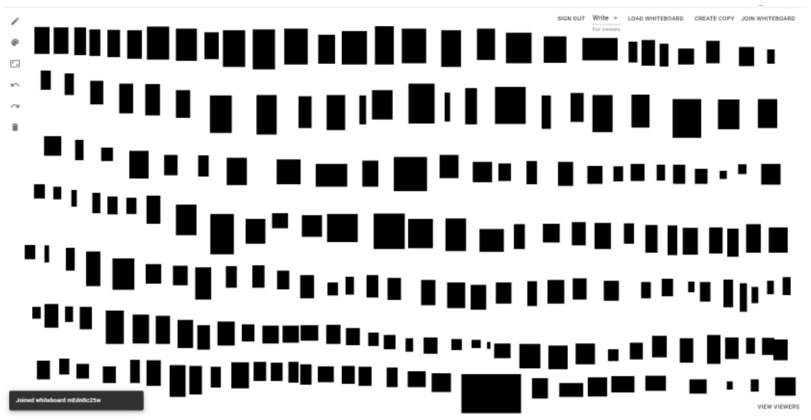


Figure 25 Whiteboard with over 200 objects on with no performance issues. Images taken after a refresh

1.9 Challenges

I experienced 2 main issues.

Undo

The first main challenge was undo. This struggle continued to follow me throughout a long period of my project, until eventually solved it for good. Originally, I didn't want to re-render every object on the screen when undo was run. I experimented with an approach which rubbed out every piece of a line draw point by point. However, this resulted in 2 unwanted side effects. The first being that it would also wipe out any drawing underneath and above the object I wanted to undo. The other issue was rather more critical. When a line is drawn it takes into account the direction you are moving the mouse, but the eraser didn't. This led to an awful jagged effect when undoing.

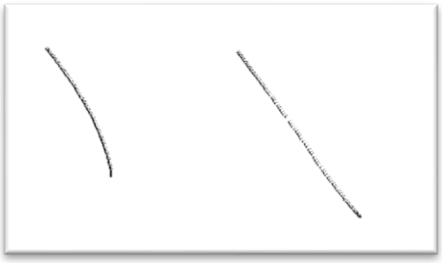


Figure 26: Jagged effect after undoing

After this, I decided to redraw each object on the board, excluding the object that was undone, after an undo/redo. Further issues were presented however, when undo would randomly undo 2 objects, and you wouldn't be able to redo. This was because when using states to set data, the functions were being too frequently, so it sometimes would skip over a state which would cascade into further errors down the line, as the client thought it had all the versions.

Canvas technology changeover

Another challenge arose with my original canvas implantation. Before using react-konva, I used the default HTML canvas element. This presented a couple issues. The first being it wasn't suitable to be used within react. The issues that this presented were the inability to edit shapes without complicated overhead and with a further inability to zoom/move around the whiteboard. At this point I took the difficult decision to restart my project on a different framework.

1.10 Things I've learnt

I have learnt a host of new technologies and methodologies in my time working on my project. Some technologies I had never used before like socket.io, node.js and konva and others I had limited experience with like react and javascript, but now I feel I can comfortably say that I understand and could make further products using these technologies.

Starting with React. It is a JavaScript library used for building complex, interactive and responsive websites. It allows me to cleanly update components on my webpage and split the website down into smaller components. It is incredibly popular among developers, with it having the most watchers, the most forks and most contributors compared to angular and Vue.

Figure 27 The React logo

I had a very small amount past experience with react before starting my project, so most of it was new to me. I understand how components work and how to lay them out, but didn't have any knowledge or understanding of the deeper technologies such as react hooks, refs and states. Due to the nature of my project, being all one page and requiring many re-renders, it required me to dig much deeper into react than I was anticipating. Through my project, it can be seen I use states for storing objects that are drawn, refs for storage of objects over re-renders, forward refs for more complicated buttons.

I had never used socket.io before, but the concept seemed quite easy to grasp. The socket server accepts a connection request from the requirement, and is given a unique ID. Then from the client, I easily send data to the server with an identifier for the relevant net message, and vice versa for sending data back. However, I had to learn how to optimize this connection, as re-renders caused continuous reconnections. After learning more about react and reading into the socket-io documentation I was able to work around my main issue.

Figure 28The socket.io logo

React-konva is another technology I hadn't used before. React-konva is based on the plain HTML version konva, which is a custom version of the HTML canvas library. I had to learn to be memory efficient, handling data over multiple tickets. For example, if I set a variable, sometimes it will be set until the next frame. Another example is if I am receiving data and objects are being sent from the database too quickly for the client to handle it can cause it to override itself. To resolve this I implement a timer on how often objects are added. Acknowledgments to my supervisor for guiding me through this issue.

Figure 29 The KONVA logo

I was familiar with JavaScript but had never used node.js. Node.js is an open source back-end server which runs JavaScript code. I was familiar with a front-end and back-end scenario though, so I found it incredibly easy to implement, but it was a learning experience, nonetheless.

Figure 30The node.js logo

1.10.1 Technical achievements

My 3 biggest technical achievements are:

- Client-Server network traffic, with lots of data in a short amount of time
- Handling asynchronous functions through both server and client
- Being able to manipulate pre-existing objects draw onto a screen

Client-server network traffic is inherently hard, as it is not as simple as a POST/GET request, as seen at the best level of web development. It requires listeners for data that could be sent completely randomly to which I also have to display cleanly, without causing interface issues for HTML which isn't designed to be responsive. This required frequent checking of the documentation provided by socket.io. It was also quite an enjoyable experience learning this kind of technology, as it felt like I was really learning about something at the heart of a modern and responsive website.

Asynchronous functions can be quite tricky, as the code won't wait for the function to complete before continuing. In some scenarios, if not handled properly data can be lost or errors can appear. Most of my server code is asynchronous due to the nature of what it does (usually just database manipulation) but I did run into issues which I overcame. This was a task I was worried about how to complete, as it can be notoriously hard.

Being able to manipulate pre-existing objects was a task that also worried me. Originally, I thought I was going to have to mathematically calculate the position of objects, which gets quite tricky with zooming and moving the canvas. I was able to complete this in the end, by taking advantage of my canvas's framework but this is one of the more complicated systems in the whiteboard.

2 Project planning

2.1 Git

I have made a total of 58 commits to the project, at of time of writing. The commits have been consistently made throughout the life-span project, with odd gaps in between. Usually these are accounted for times by other commitments such as tests and/or multiple assignments.

At the very beginning during the MVP, my commits were mostly major feature implementations. However, later on I would commit more for more smaller changes like minor bug fixes. All my commit messages have been relevant and have adequately explained what has been changed, without requiring further examination of code. No branches other than the master was created, as there was never a need. Arguably, it might have been useful when changing from the HTML canvas to react-konva. However, there wasn't much code that was reusable and on the few small things that I did take from the old version, I just referred back to an older commit.

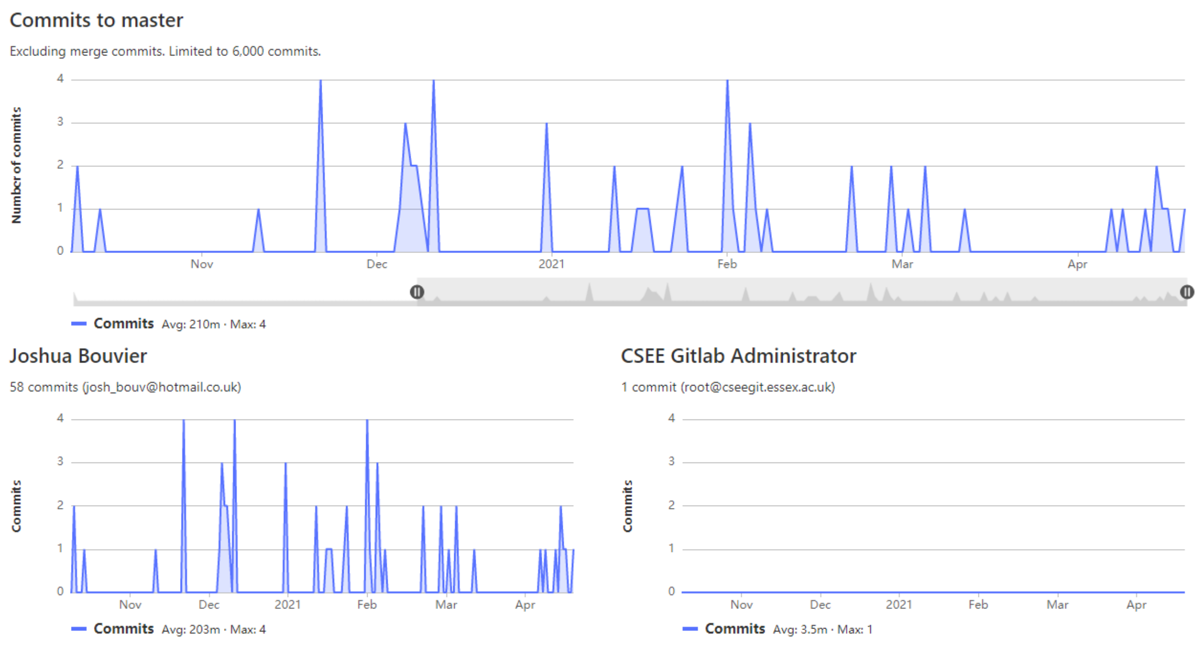
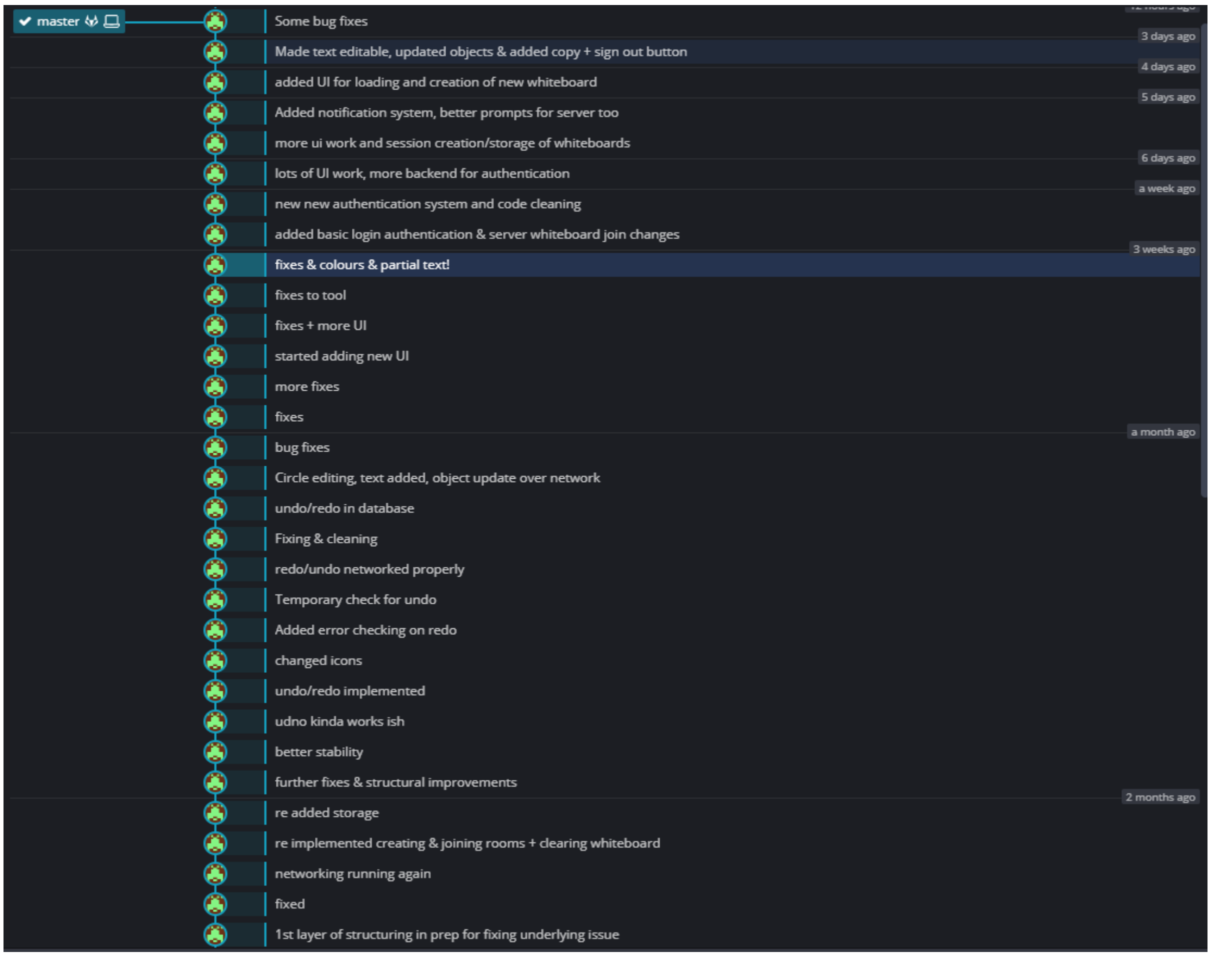


Figure 31 GitLab commit chart, taken on 27/04/2021



master

Some bug fixes

3 days ago

Made text editable, updated objects & added copy + sign out button

4 days ago

added UI for loading and creation of new whiteboard

5 days ago

Added notification system, better prompts for server too

more ui work and session creation/storage of whiteboards

6 days ago

lots of UI work, more backend for authentication

a week ago

new new authentication system and code cleaning

added basic login authentication & server whiteboard join changes

3 weeks ago

fixes & colours & partial text!

fixes to tool

fixes + more UI

started adding new UI

more fixes

fixes

a month ago

bug fixes

Circle editing, text added, object update over network

undo/redo in database

Fixing & cleaning

redo/undo networked properly

Temporary check for undo

Added error checking on redo

changed icons

undo/redo implemented

udno kinda works ish

better stability

further fixes & structural improvements

2 months ago

re added storage

re implemented creating & joining rooms + clearing whiteboard

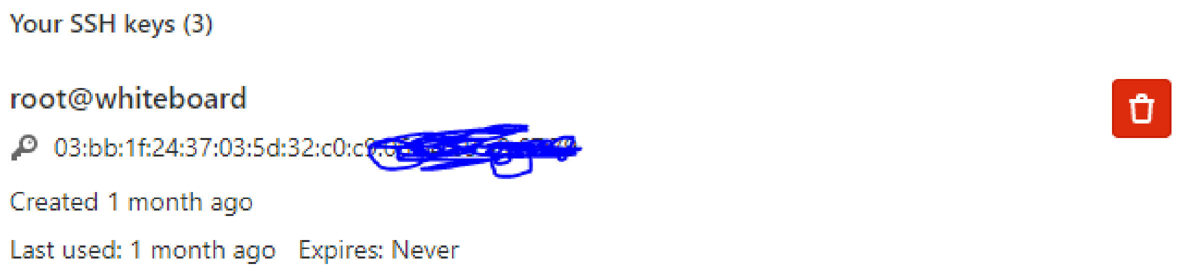
networking running again

fixed

1st layer of structuring in prep for fixing underlying issue

Figure 32 Commit messages

I was already quite familiar with git but to host the website on my server, I had to also generate a new SSH key to be able to pull the latest updates from the repo to my server.



Your SSH keys (3)

root@whiteboard

03:bb:1f:24:37:03:5d:32:c0:c[redacted]

Created 1 month ago

Last used: 1 month ago   Expires: Never

Figure 33 Example of SSH key, actual key is hidden however for security

2.2 Jira

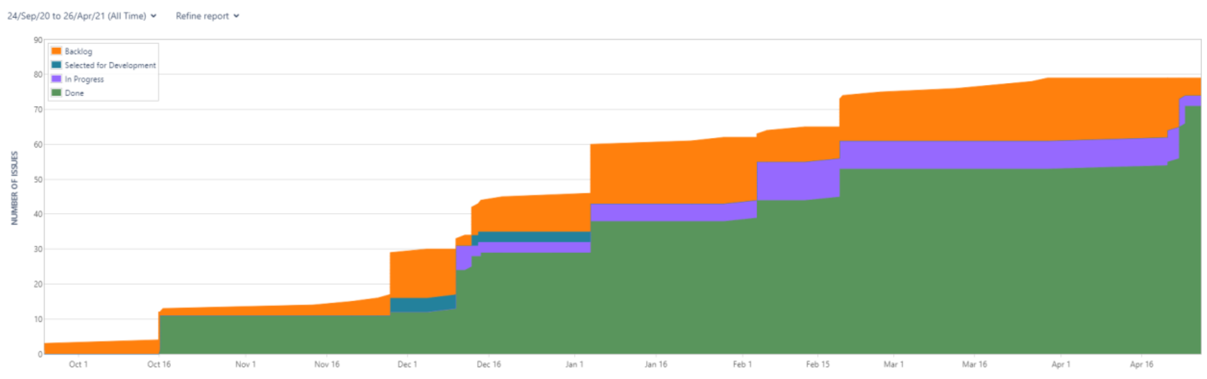
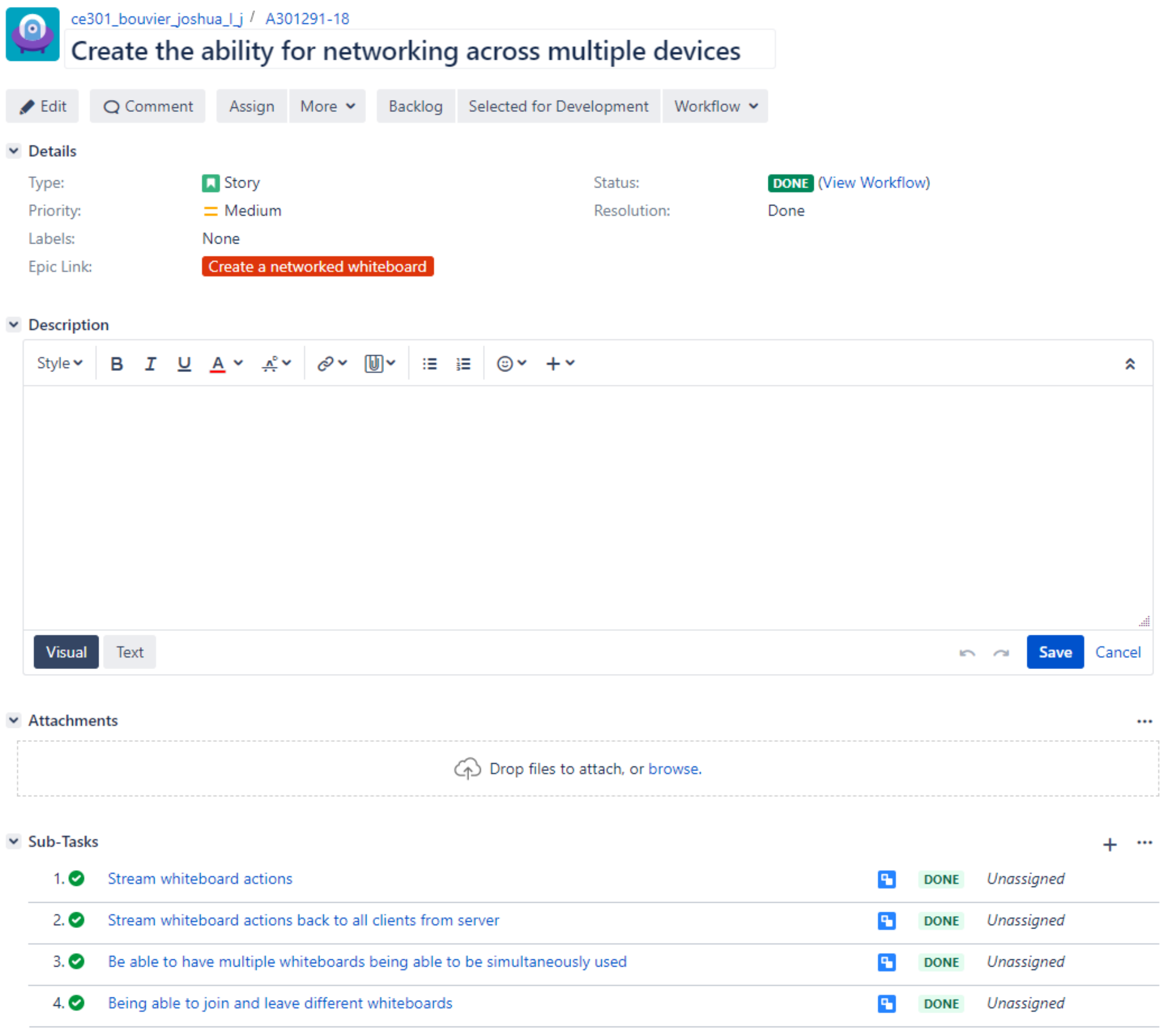


Figure 34 Jira activity, images taken on 27/04/2021

On reflection, I do wish I had used Jira a bit more frequently. However, as seen in figure 34, it was used through the project roughly every month, when a large set of my tasks were updated and, if applicable, new ones were added.

I utilised story's and sub-tasks properly as well, as shown in this screenshot of this story. Here you can see 4 completed subtasks.



ce301_bouvier_joshua_l_j / A301291-18

Create the ability for networking across multiple devices

Details

Type:	Story	Status:	DONE (View Workflow)
Priority:	Medium	Resolution:	Done
Labels:	None		
Epic Link:	Create a networked whiteboard		

Description

Attachments

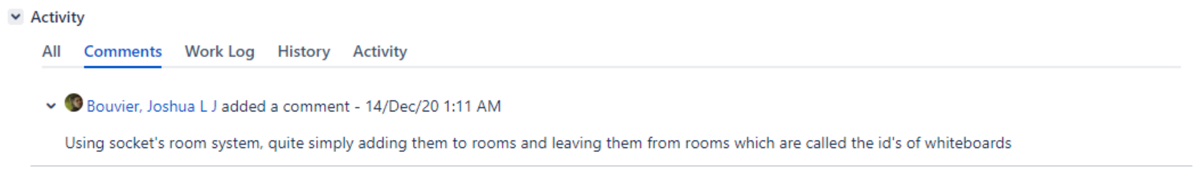
Drop files to attach, or browse.

Sub-Tasks

1. ✓	Stream whiteboard actions	DONE	*Unassigned*
2. ✓	Stream whiteboard actions back to all clients from server	DONE	*Unassigned*
3. ✓	Be able to have multiple whiteboards being able to be simultaneously used	DONE	*Unassigned*
4. ✓	Being able to join and leave different whiteboards	DONE	*Unassigned*

Figure 35 Jira Story with subtasks

Some comments were made to tasks were relevant too, as seen in figure 36



Bouvier, Joshua L J added a comment - 14/Dec/20 1:11 AM

Using socket's room system, quite simply adding them to rooms and leaving them from rooms which are called the id's of whiteboards

Figure 36 Jira comment on a task

2.3 Timeline

2.3.1 October

- MVP was made, which consisted of an HTML canvas which you could draw lines on. The lines could also seen and made across to other devices viewing the whiteboard. It had options to choose between 7 different colours. You could also clear the whiteboard and switch between 2 globally accessible streams. The stream would send lines that were drawn after the joining of the room, but no data was stored on the server, so any previous work could not be viewed.

2.3.2 November

- Undo was added and the persistence of whiteboard 1 was stored across browser refreshes.
- Storage was added to a database so it could be saved across server restarts.

2.3.3 December

- Random generation of whiteboard names was added.
- The addition of squares, circles and text to a local client (not networked). The size of the shapes could not be changed after it was initially drawn, and the text could not be changed from “testing123”.
- Shapes were networked.

2.3.4 January

- New implementation of a different canvas technology was implemented. It used Konva instead of an HTML canvas element. The eraser was added. However, storage, networking, clearing and undo was temporarily removed because of the change in framework, meaning it needed to be recoded from scratch.
- Networking was reimplemented.
- Creating rooms was implemented, joining and clearing was reimplemented.
- Storage of whiteboards was reimplemented.

2.3.5 February

- Reimplementation of undo on the local client. However, this proved to be troublesome for some weeks ahead.
- Implementation of redo on the local client.
- Networked undo and redo.
- Circles became resizable (on local client only), text readded, the movement of objects over network.

2.3.6 March

- Made new UI from scratch for the whiteboard, encompassing buttons for tools, colours, manipulation, undo, redo, clear.

2.3.7 April

- Added custom authentication system & basic permissions.
- More UI work, added buttons and ability to login, signup, sign out and create whiteboards.
- Added notification system.
- Added UI for loading of whiteboards.
- Made text editable and added copy whiteboard button..

3 Conclusion

3.1 Objectives achieved

With regards to the objectives I set out to achieve at the start of the report, all my main objectives were achieved. In addition, I was able to complete some of my stretch goals and made some unexpected additional achievements along the way too. Please see table below.

Table 4 Goal achivment summary table

Objective	Achieved
Usability: Most critically, to design a blank canvas which is interactive, responsive and efficient. This is important because the canvas needs to be suitable for all environments, including scenarios where the user's machine may be quite old or slow, or potentially even a mobile device such as a phone or tablet.	Completely achieved.
Functionality: The ability to draw lines, circles, squares, images and text onto a canvas.	Mostly achieved. Ability to upload Images is not present.
Functionality: Objects to be editable. To elaborate, editable with regard to their position, size, colour and content. The object should also be able to be deleted.	Completely achieved.
Functionality: The ability to rescale and move the whiteboard, allowing for an unlimited size for the whiteboard.	Completely achieved.
Storage: The whiteboard must be stored in a database, so that it can be saved and loaded at a later date and across server restarts. This also allows the whiteboard to have an owner.	Completely achieved.
Permissions: To include a permissions system, allowing the owner of a whiteboard to make it view-only by other users or editable by all. An owner should also be able to set individual permissions of users that are logged in and change these permissions at any point.	Completely achieved.
Authentication: To include a user authentication system. Whilst this project focuses on an online collaborative whiteboard to demonstrate permissions and ownership of whiteboards, a basic authentication system is still required. However, this will be without a need for an email verification system or encryption as it's a proof-of-concept.	Completely achieved.
Live streaming: A networked ability. This encompasses all actions such as drawing, clearing, undo and redo being actioned across all users who are currently viewing the whiteboard at the same time. This will allow other users to see objects being drawn in real-time (specifically not after completion, but as they are being drawn).	Completely achieved.
Design: An elegant, accessible UI.	Completely achieved.
Stretch goals	
Functionality: Live viewer list to allow you to see who is currently viewing the whiteboard	Completely achieved.
Mobile support: This would allow for more accessibility for scenarios where a computer might not be available to everyone who wants to view a whiteboard.	Whiteboards can be viewed on mobile, but you are unable to interact with it

	nor is the UI optimized for it.
Un-project goals	
Ability to delete objects	Completely achieved.
Ability to make copy of whiteboards	Completely achieved.

3.2 Objectives not achieved/Things I would do to improve

There are some things I feel could be improved further though.

The first thing I would like to have taken further, is the ability of the user to choose a custom colour rather be limited to selecting from one of a set of pre-defined colours. I did make multiple attempts at getting this to work, but UI issues kept presenting themselves, causing errors and instability, meaning I was unable to further pursue a better solution.

Another thing I would like to have added is a context menu for objects. which would allow you to change the colour of objects after they are placed. However, I was able to allow users to delete objects by pressing the delete button on the keyboard, after selecting it.

Another task I was unable to complete was adding fully fledged mobile support. It is supported to a degree, in that you can view drawings on the whiteboard, but portions of the UI and drawing cannot be accessed.

Finally, I wasn't able to add images due to the complexity of requiring a file server to store them.

3.3 As a whole

Overall, the project was a success, leading to a result I am very happy with. I was able to achieve practically all my initial goals, with some extra unexpected goals like a viewer list and copying whiteboards too. I am particularly pleased with networking aspect of my design.

Whilst it is a shame I was unable to add mobile support, I had identified this goal from the start as a stretch goal for if I had time. However, after all other tasks were completed, there was not the time to commit to the research required to make a mobile friendly UI, plus I believe I would also have run the risk of breaking my existing UI. As a result, I had to make the decision not to pursue it.

I have enjoyed the project and am appreciative that I have learnt a great deal about various technologies through my time working on it, which I have no doubt I will use in the future.

4 Bibliography

[1] D. G. &. D. M. &. D. A. &. V. Door, “The evolution of an effective pedagogy for teachers using the interactive whiteboard in mathematics and modern languages: an empirical analysis from the secondary sector,” Taylor & Francis Online, 15 Feb 2007. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/17439880601141146?scroll=top&needAccess=true.

[2] S. D. P, ““The Effects of Interactive Whiteboards (IWBs) on Student Performance and Learning: A Literature Review.”,” Journal of Educational Technology Systems, 2010. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.2190/ET.38.3.b.

[3] T. Rojanarata, “How Online Whiteboard Promotes Students’ Collaborative,” Association for Computing Machinery, March 2020. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3395245.3396433.

[4] N. Sultan, “Cloud computing for education: A new dawn?,” International Journal of Information Management, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401209001170.

[5] K.-O. Jeong*, “A Study on the Integration of Google Docs as a Webbased Collaborative Learning Platform in,” Indian Journal of Science and Technology, 2016. [Online]. Available: https://www.researchgate.net/profile/Kyeong-Ouk-Jeong/publication/309752493_A_Study_on_the_Integration_of_Google_Docs_as_a_Web-based_Collaborative_Learning_Platform_in_EFL_Writing_Instruction/links/5851ea9308ae0c0f32200022/A-Study-on-the-Integration-of-Go.

[6] Buisness Matters, “Are you one of the 88% of industry professionals using online collaboration tools?,” Buisness Matters, 2 May 2017. [Online]. Available: https://bmmagazine.co.uk/in-business/industry-professionals-online-collaboration-tools/.

[7] smallbuisness.co.uk, “The rise of online collaboration tools to improve business competitiveness,” Bonhill Group plc,, 21 September 2017. [Online]. Available: https://smallbusiness.co.uk/rise-online-collaboration-tools-improve-business-competitiveness-2540820/.

[8] S. Huisache, “Collaboration Software Statistics and Trends,” TrustRadius, 5 June 2020. [Online]. Available: https://www.trustradius.com/vendor-blog/collaboration-statistics-and-trends.

[9] J. O’Halloran, “Massive uptick in collaboration software usage in 2020,” ComputerWeekly.com, 11 Febuary 2021. [Online]. Available: https://www.computerweekly.com/news/252496232/Massive-uptick-in-collaboration-software-usage-in-2020#:~:text=The%20study%20found%20that%20usage,further%20accelerated%20in%20early%20August.

[10] Aternity, “The Global Remote Work Productivity Tracker: Volume 7: Collaboration app sprawl in the future,” Aternity, 2021. [Online]. Available: https://www.aternity.com/wp-content/uploads/2021/02/Global-Remote-Work-Productivity-Tracker-Vol-7.pdf.

[11] “A Web Whiteboard App,” Miro, [Online]. Available: https://awwapp.com/.

[12] “Online whiteboard for visual collaboration,” Miro, [Online]. Available: https://miro.com/app/dashboard/.

[13] “Online Whiteboard with Realtime Collaboration,” Ziteboard, [Online]. Available: https://app.ziteboard.com/.

[14] “Whiteboard Fox,” Springbok Solutions Ltd, [Online]. Available: https://whiteboardfox.com/.

[15] “A JavaScript library for building user interfaces,” [Online]. Available: https://reactjs.org/.

[16] “The Progressive,” [Online]. Available: https://vuejs.org/.

[17] “The modern web,” [Online]. Available: https://angular.io/.

[18] “npm trends,” [Online]. Available: https://www.npmtrends.com/angular-vs-react-vs-vue.

[19] “PHP: Hypertext Preprocessor,” [Online]. Available: https://www.php.net/.

[20] “Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.,” [Online]. Available: https://nodejs.org/en/.

[21] “The Go Programming Language,” [Online]. Available: https://golang.org/.

[22] S. Martin, “Node.js vs PHP: Which is better for web development?,” hackernoon, 25 July 2019. [Online]. Available: https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp.

[23] MySQL, [Online]. Available: https://www.mysql.com/.

[24] Oracle, [Online]. Available: https://www.oracle.com/database/.

[25] Microsoft, [Online]. Available: https://www.microsoft.com/en-us/sql-server.

[26] “PostgreSQL: The World's Most Advanced Open Source Relational Database,” PostgreSQL Global Development Group, [Online]. Available: https://www.postgresql.org/.

[27] “The database for,” MongoDB, Inc, [Online]. Available: https://www.mongodb.com/.

[28] “MariaDB Server: The open source relational database,” MariaDB Foundation, [Online]. Available: https://mariadb.org/.

[29] MongoDB, Inc., “What is NoSQL?,” MongoDB, Inc, [Online]. Available: https://www.mongodb.com/nosql-explained.

[30] socket.io, [Online]. Available: https://socket.io/.

[31] a. &. sab24, “Benchmark-driven development,” [Online]. Available: https://github.com/uNetworking/uWebSockets/blob/master/misc/websocket_lineup.png.

[32] W3Schools, “HTML Canvas Graphics,” W3Schools, [Online]. Available: https://www.w3schools.com/html/html5_canvas.asp.

[33] “Welcome to the MongoDB Documentation,” MongoDB, Inc, [Online]. Available: https://docs.mongodb.com/.

[34] A. Lavrenov, “react-konva vs vanilla canvas,” [Online]. Available: https://github.com/konvajs/react-konva#react-konva-vs-vanilla-canvas.

[35] Facebook Inc, “Refs and the DOM,” Facebook Inc, [Online]. Available: https://reactjs.org/docs/refs-and-the-dom.html.

[36] “Auth0: Secure access for everyone. But not just anyone.,” Auth0, [Online]. Available: https://auth0.com/.

[37] “Amazon Cognito: Simple and Secure User Sign-Up, Sign-In, and Access Control,” Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/cognito/.

[38] shuo.yang.2006, “How to create a user on the service side whenever a user register to my app through Auth0?,” Auth0, [Online]. Available: https://community.auth0.com/t/how-to-create-a-user-on-the-service-side-whenever-a-user-register-to-my-app-through-auth0/7056/2.

[39] A. Lavrenov, “Docs,” Konva project, [Online]. Available: https://konvajs.org/api/Konva.html.

[40] A. Lavrenov, “toDataURL,” Konva Foundation, [Online]. Available: https://konvajs.org/api/Konva.Canvas.html#toDataURL.

[41] Department Of Education, “Class Size Data, Secondary,” 2007. [Online]. Available: https://webarchive.nationalarchives.gov.uk/20130321115027/https://www.education.gov.uk/researchandstatistics/statistics/allstatistics/a00196874/class-size-data-secondary.

[42] “10,000 Shapes with Tooltips Stress Test with Konva,” Konva, [Online]. Available: https://konvajs.org/docs/sandbox/10000_Shapes_with_Tooltip.html.

[43] D. M. &. D. Glover, “The evolution of an effective pedagogy for teachers using the interactive whiteboard in mathematics and modern languages: an empirical analysis from the secondary sector,” Taylor & Francis Online, 28 May 2010. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/17439880601141146?journalCode=cjem20.

[44] c. &. P. &. И. З. &. TURTLE, “Generate random string/characters in JavaScript,” stackoverflow, 2009-2019. [Online]. Available: https://stackoverflow.com/questions/1349404/generate-random-string-characters-in-javascript.

[45] A. Lavrenov, “How to find relative mouse position?,” Konva Foundation, Unkown. [Online]. Available: https://konvajs.org/docs/sandbox/Relative_Pointer_Position.html#page-title.

[46] A. Lavrenov, "Text editing in HTML5 canvas with Konva," Konva Foundation, Unkown. [Online]. Available: https://konvajs.org/docs/sandbox/Editable_Text.html#page-title.