Online Collaborative whiteboard

CE301 – capstone project

Joshua Bouvier

2021

# 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 for example with myself, right at home! They are a great tool to express information onto a canvas and to encourage collaborative learning. In an online context this increase the possibilities and potential to allow for multiple people work on the same whiteboard, to view the whiteboard in remote locations and to allow the ability of extra actions like undo, redo and rescaling.

The purpose of this project was to create a whiteboard that could 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 with everyone who is using this. It's also compatible with mobiles as well as desktops allowing people with varying technology to use the whiteboard.

The whiteboard allows users to draw lines and objects, images, text, undo, redo, create and join whiteboard sessions, set permissions, resize objects and the canvas itself all with a clean and simplistic UI.

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

Table of contents

[Acknowledgements 1](#_Toc70508003)

[Abstract 2](#_Toc70508004)

[1 Main text 4](#_Toc70508005)

[1.1 Literature review 4](#_Toc70508006)

[1.1.1 Introduction 4](#_Toc70508007)

[1.1.2 Context 4](#_Toc70508008)

[1.1.3 Use of interactive whiteboards 4](#_Toc70508009)

[1.1.4 Use of collaborative online tools 4](#_Toc70508010)

[1.1.5 Online collaborative whiteboards 5](#_Toc70508011)

[1.1.6 Conclusion 6](#_Toc70508012)

[1.2 Project goals 6](#_Toc70508013)

[1.2.1 Main goals 6](#_Toc70508014)

[1.2.2 Stretch goals 7](#_Toc70508015)

[1.3 Design style 7](#_Toc70508016)

[1.4 Technical documentation 8](#_Toc70508017)

[1.5 Predevelopment 8](#_Toc70508018)

[1.6 Development narrative 10](#_Toc70508019)

[1.7 Testing 15](#_Toc70508020)

[1.7.1 General test for errors 15](#_Toc70508021)

[1.7.2 Max users test 16](#_Toc70508022)

[1.7.3 Max objects test 16](#_Toc70508023)

[1.8 Struggles 16](#_Toc70508024)

[1.9 Things I’ve learnt 17](#_Toc70508025)

[1.9.1 Technical achievements 17](#_Toc70508026)

[2 Project planning 18](#_Toc70508027)

[2.1 Git 18](#_Toc70508028)

[2.2 Jira 20](#_Toc70508029)

[2.3 Timeline 21](#_Toc70508030)

[3 Conclusions 22](#_Toc70508031)

[3.1 Objectives achieved 22](#_Toc70508032)

[3.2 Objectives not achieved 23](#_Toc70508033)

[3.3 As a whole 23](#_Toc70508034)

[4 References 24](#_Toc70508035)

[5 Appendices 25](#_Toc70508036)

# Main text

## Literature review

### Introduction

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

### 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 are looking for solutions that enable them to work collaboratively online.

Whiteboards can be seen in many contexts, such as in education, business and personal situations. They can often be seen as tools to provide information with as seen by presenters or also as learning tools to allow group collaborations.

Personally, I own a wall-mounted whiteboard which I frequently use to scribble out information 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 on. Personally, I’m also an interactive learner. Being presented with a slideshow on a unused interactive whiteboard feels like such a wasted opportunity. With the potential to extended directly to a student via provided equipment or their own electronic devices, I believe offers huge potential for improvement.

### Use of interactive whiteboards

Whilst whiteboards can be used in multiple environments, such as in personal use and business use, most studies focus on it in an education environment. This is backed up by the statement in [SOURCE 1] “While there is a great deal of research available on the effects of IWB use in the classroom”. They are commonly used by teachers, especially in the UK due to a largescale push by the government but also in Europe. There was a £15 billion pound scheme to install them in all primary and secondary schools [1], which nowadays it’s an important part of the classroom.

Whilst the £15 billion pound drive was to install wall mounted whiteboards – this technology can be repurposed for interactive whiteboards too allowing students to access the whiteboard on laptops or tablets allowing for better interaction. Refer to study where they say about mobile phone [<https://dl.acm.org/doi/pdf/10.1145/3395245.3396433>]]

### 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 provide education using collaboration tools such as Google’s docs, sheets and slides [2]. In a survey carried out on Google docs in particular, it was shown that the majority of students gave it a rating of 4 or 5 out of 5 when asked about there experience using google docs and there opinions of it [3]. This shows that collaboration tools can be 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 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 [4] [5]. 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.’ [6]

Joe O’Halloran, Computer Weekly, 11 Feb 2021 [7] states “For collaboration software, 2020 was a year like no other”. His claims are based on research from Aternity [8] 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.”

### Online collaborative whiteboards

There are multiple products already available which provide online whiteboards, them being AWW App (<https://awwapp.com/>) (however this is going defunct at the end of July, 2021), Miro (<https://miro.com/app/dashboard/>) (which AWW App is being merged with), Ziteboard (<https://app.ziteboard.com/>) and Whiteboard Fox (<https://whiteboardfox.com/>) among others.

Here is a 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 |

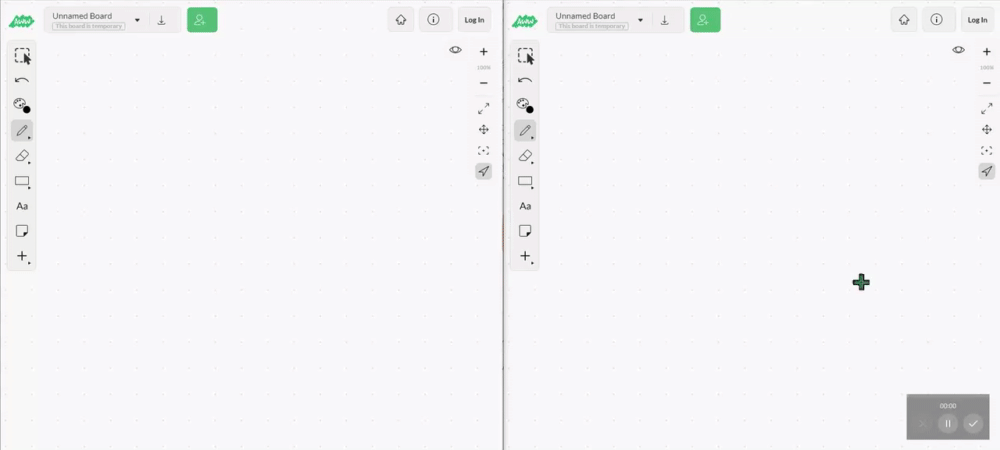
As seen by my comparisons, websites don’t have real-time streaming of drawing (as in it often only updates after the drawing has been finished). In my opinion, I find objects popping onto the screen quite jarring. It’s also more engaging to see content illustrated in real time rather than waiting for it to pop into the screen.

Figure Drawing on 2 whiteboards over the internet with AWW APP

Others have log-in or payment walls which are required to access important features. This can be incredibly annoying and requires every viewer to also either make an account or make unnecessary clicks to clear the popup.

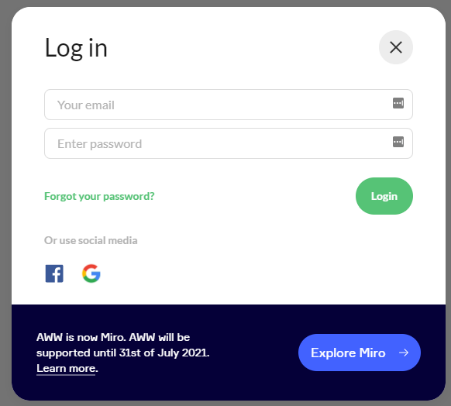
All whiteboards have the common ability to share it to another user, which will give them permission to just view it or edit it if they log in, but once someone has been given permissions that cannot be changed. This gives very little manoeuvring for ways you can present which is a huge drawback. For example you might want to let someone right an answer on the whiteboard for a brief period of time. In particular with Ziteboard, the whiteboard owner also has to login to be able to share the whiteboard. This leads to an unpleasant experience for a user as they may not be willing to hand over their data or don’t have the time to faff around making an account to scribble some notes.

Figure Login promp from AWW APP

### 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 incorporates the suggested improvements.

## Project goals

### Main goals

The main goals of my project are the following:

* Most critically, a blank canvas. The canvas must be interactable, responsive and light-weight. This is because the canvas is designed for all environments, including in scenarios where the user’s machine could be quite old or slow, or potentially even a mobile device such as a phone or tablet.
* The ability to draw lines, circles, squares, images and text onto a canvas, with the circles, squares and text.
* Objects being editable. To elaborate on editable, I mean for their position, size, colour and content to be changeable. The object should also be able to be deleted.
* The ability to rescale and move the whiteboard, allowing for an unlimited size for the whiteboard and complex things to be displayed.
* The storage of whiteboards in a database, so whiteboards can be saved and loaded at a later date and across server restarts. This also allows the ability of a whiteboard having an owner.
* 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
* 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 required, however there is no need for an email verification system or encryption as it’s a proof-of-concept.
* Live viewer list, showing usernames of users that are logged in
* 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 and allowing other users to see objects being drawn in real-time (specifically not after completion, but as they are being drawn).
* A good looking, accessible UI.

### Stretch goals

* 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.

## Design style

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

Figure Topbar of whiteboard

After looking over all the designs, I found my personal preference to be material design. It also presented to be the clearest out of all of them.

I decided I wanted it to be minimalistic yet fully accessible at the same time. 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 on the top left side. All icons also have tooltips to prevent for any confusion about what they do.

Figure Sidebar of whiteboard

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 had to represent a good button for actions like loading saved whiteboards and creating a copy of a whiteboard. To cut down on the amount of buttons seen 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. In line with the other buttons.

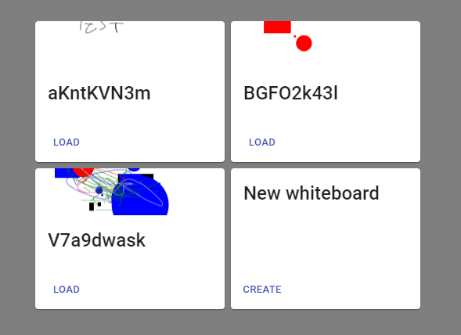
The next important UI component was how I displayed saved whiteboards. Off the bat I knew I wanted a preview image of the whiteboard to be displayed with the name below. Other than that I didn’t want any extra UI pieces or unnecessary descriptions.

Figure Load whiteboards menu

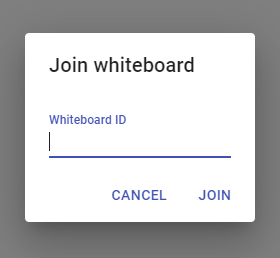
Join whiteboards, login and signup all share the exact same design. When you press on the button a popup shows asking you to input information, which is laid out in the same manner across all the popups.

Figure Join whiteboards menu

In conclusion, I believe this gives the whiteboard a clean look with little room for confusion regarding what each button does.

## Technical documentation

<https://cseegit.essex.ac.uk/ce301_2020/ce301_bouvier_joshua_l_j/-/wikis/home>

## Predevelopment

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, Vue.js and Angular. I decided to go with React as I had some experience with it before and I knew that it is incredibly popular with very good documentation. Below is my comparison of my options.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

Next, I decided to choose the back-end server. My 3 main options were PHP, node.js and golang. In comparison between PHP and node.js, node.js comes out the clear winner. Node.js is faster than PHP and in my opinion is much cleaner. Comparing node.js to go I wasn’t too comfortable with the documentation for go so I decided with node.js.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

I then had to choose which database to use. I have a large array of options being MySQL, Orcale Microsoft SQL Server, PostgreSQL, MongoDB, MariaDB. In the end I choose mongoDB for the fact it used noSQL, 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 radius, and a circle wouldn’t have a XY size. Here was my comparison of the databases. See below my comparison between them.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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. | Have 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 | No 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 |

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

## Development narrative

**October**

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 stuff you can do with it, <https://www.w3schools.com/html/html5_canvas.asp>. The canvas element, as implied by the name, provides a blank canvas to which I can call functions on to draw lines and objects. The next step was to be able to draw in the canvas. I decided to start with simply drawing a line.

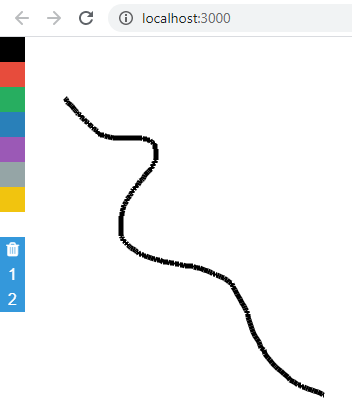
To do this I added an event listener which detected when the mouse was pressed, ‘mousedown’. I made a variable, ‘isDrawing’, to true to signify that the mouse is being pressed, and then I got the coordinates of the current position of the mouse and called a function I made called ‘drawLine’ that drew 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 before 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 also looked quite jagged of you moved the cursor slowly due to it being many small lines rather than a single large one. This will be addressed later on.

Figure 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 is set, ‘line.strokeStyle = stroke’. I also added a clear button that clears the whiteboard.

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. Them being ‘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’.

I then added a net message to my clear whiteboard function labelled clear, that triggered a function on all other clients to trigger the clear function on there’s too.

I then added a basic system, to simulate what it would be like to have multiple sessions at once on a different whiteboard. 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 ids previously mentioned. That variable was then also sent in the ‘drawLine’ function with the data of the line, and once received by the server it used that data to send the data to all users who were are that room.

This completed my MVP.

**November**

My next step was to implement a temporary storage of the whiteboard for whilst the server was running (note that it’s 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. 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).

This did allow me to implement and undo feature however, which worked by quite simply popping the latest inserted line into the completed objects, and redraw the whiteboard using the new table of data. It was also applied to the server meaning undo would also have affect for it’s copy of the whiteboard.

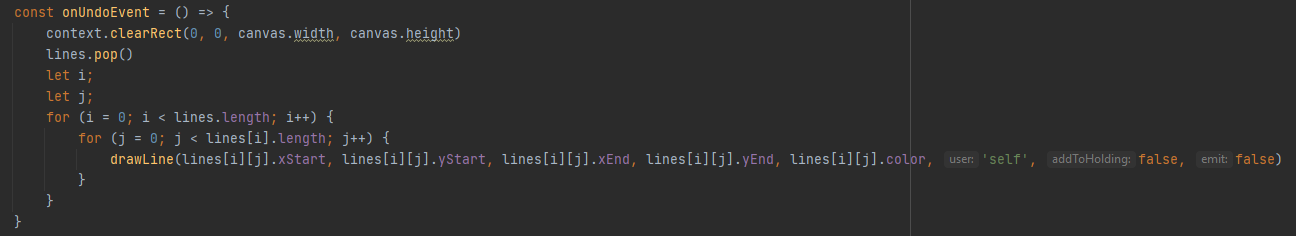


Figure Original undo function

The reason why I didn’t jump straight into a database to store whiteboards was because I had chosen to use an unfamiliar database system to me, mongoDB, which required some time researching and reading documentation to 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.

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 use 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.

Figure 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 as before, where I deleted the latest item inserted into the database.

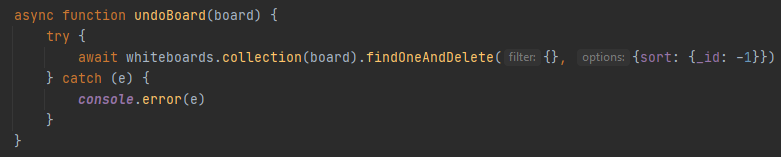


Figure Server-side undo function

**December**

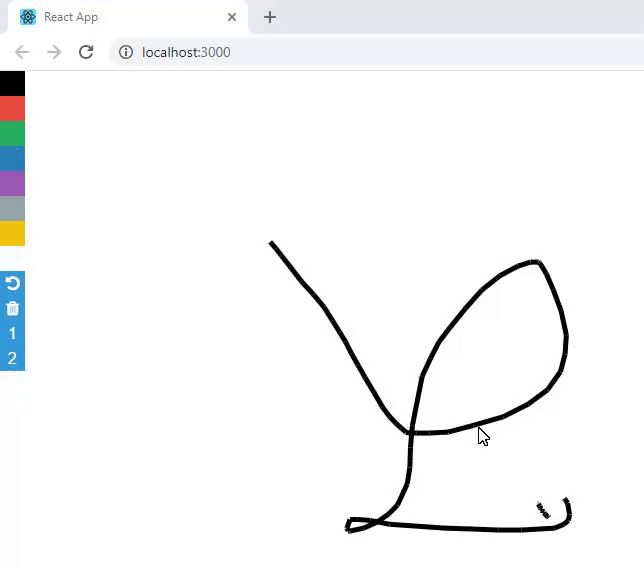
I also added the ability to load the whiteboard from the database when a user views that whiteboard. It would work by when the client loads the page it sends a net message with the whiteboard ID defined on the client, and the server sends back a net message for each object in the board with the data 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 was made to handle the net message which would simply add it the completed objects table, and then order a re-render of the canvas.

Figure Whiteboards loading after refresh from the database

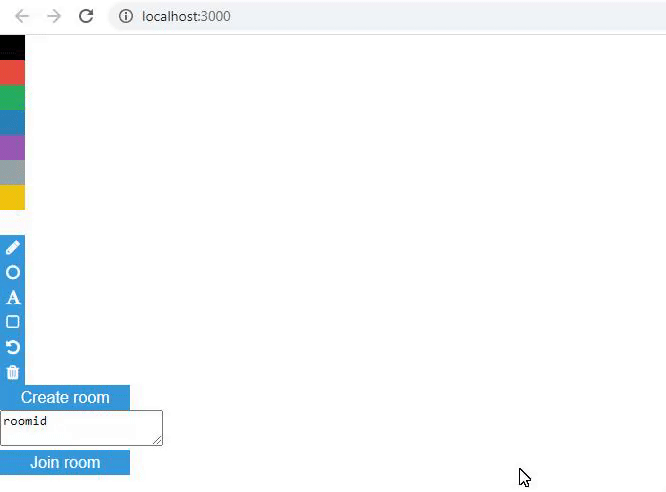
At this point I looked to expand from just being able to draw lines, so I added a very basic version of the ability to draw squares, circles and text. To do this I added some extra buttons each symbolising a different object to draw. For example if the square was selected, it would set a variable which 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 exact 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 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 similar fashion to the ‘drawLine’ function, I added net messages that triggered the relevant drawing function.

**January**

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 were presented was the inability of being able 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 choose Konva based off some research 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 used 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 feat 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 to be that a re-render of the whiteboard (caused by drawing) was causing a new socket connection to be opened, which also meant 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 a challenge to solve. To counter this I had to do further research into react and found I should be defining clean-up behaviour for 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 form the database again for loading a whiteboard. This was an easy one-line implantation.

**February**

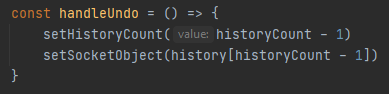
I re-added undo, but I also added for the first time redo. These proved a bit complicated however due to the nature of re-renders and edge cases. An issue I originally ran into was that I had a state for both the counting of steps made in a whiteboard and an array for a copy of all objects on the canvas at that step for later reference. A step goes forward by one every time something is drawn. It can travers 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 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.

Figure Latest undo function

I then re-added undo into the database, but also added redo. To do this I use the history array made by the undo function and using the current step + 1 of the whiteboard I reinsert 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 do 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.

**March**

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 tacky 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 with the drawing options and colours expand out into further side buttons. All the items are contained within a container, in a list. I then use a popover component to display the extra buttons for the drawing tools and colours.

Figure The sidebar UI

**April**

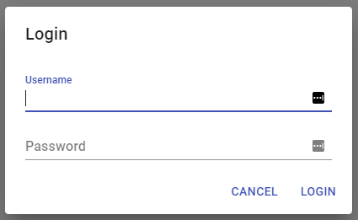
At this point I decided to implement the authentication system. I didn’t want to dedicate too much time to it as it wasn’t a core part of my project, but is necessary for me to be able to build a permission system. I looked online and found some services that handled this for me, being Auth0 and Amazon’s Cognito. What both services would do was handle the entire authentication system where you add a login button and it takes you to their website, they 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 to see if any users had signed up, which wouldn’t work in my scenario. This is 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 my servers wouldn’t know you existed yet. The same issues would occur when its detecting if a logged in user is viewing a whiteboard, as it wouldn’t think this user is valid. Whilst this could be handled it would take extra code nor did I want to be keeping track of 2 sets of data, being the data on Auth0/Amazon Cognito and the data on my database.

Figure The login UI

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, so I decided to have user action buttons displayed on the top right of the whiteboard as text buttons, rather than icon buttons. This is because I wanted these actions to be explicitly clear (ie 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, where 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 which you type in the details and it just checks your username and password. The unique key for a client is generated by mongoDB and is then stored on the client in its local storage. The client has a check 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 The load whiteboards menu

I then made an ownership system for whiteboards. On the creation of a whiteboard, I inserted 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.

## Testing

### General test for errors

For the entirety of the development cycle, I have always been testing my code usually with myself but sometimes with friends to catch stability issues and more edge-case issues which I wouldn’t particularly 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.

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

MULTIPLE BROWSERS – MOBILE?

The website has been tested on chrome and firefox. On both it performs with no issues.

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

* CPU: Intel® Xeon® Gold 2 cores processor
* RAM: 4GB
* Storage: 40GB SSD

### Max users test

### Max objects test

## Struggles

I had 2 main struggles.

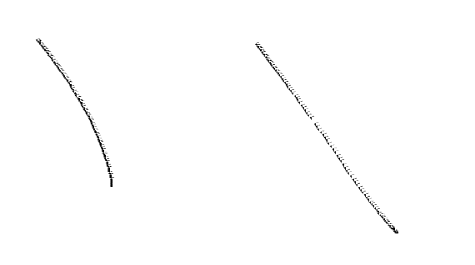
The first main struggle was undo. This struggle continued to follow me throughout a long period of my project until eventually getting solved for good. Originally, I didn’t want to re-render every object on the screen when undo was ran. I experimented with an approach which rubbed out every piece of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

Figure : Jagged effect after undoing

After this I decided to redraw each object on the board excluding the one in question after an undo/redo. Further issues were presented however when it came to storage a history of a state of a whiteboard when flicking between undo and redo. An issue presented itself as because when using states to set data, the functions weren’t being called in order so it sometimes would skip over a state which would cascade into further errors down the line as it thinks it has all the versions.

Another struggle was 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 best suited for being used within react. The issues that was presented was the inability of being able 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.

## Things I’ve learnt

Ive learnt a bunch of new technologies and methodologies in my time working on my project. Some technologies I had never used before and other I had light experience, but now I feel I can comfortable say that I understand and could make further products using these technologies.

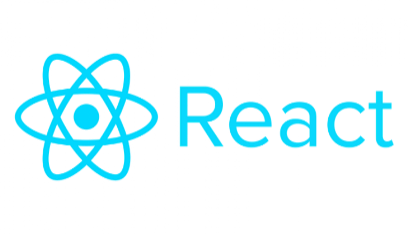
Starting with React, it’s a JavaScript library used for building complex, interactive and responsive websites. It allows me to cleanly update components on my webpage and split the website down into smaller components. It’s incredibly popular among developers, with it having the most watchers, the most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

Figure The React logo

I had a very small amount past experience with react before starting my project, so most of it was knew 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. Because of 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 I 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 easily to grapple. The socket server accepts a connection request from the requirement, and are given a unique ID. Then from the client I easily just send data to the server with an identifier for which net message it was, and vica versa for sending data back. I had to learn however had 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 about my main issue.

Figure The socket.io logo

React-konva is another technology I hadn’t used before. React-konva is based of 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, ie objects from the database, too quick for the client to handle causing it to override itself, I implemented a timer on how often objects are added. Acknowledgments to my supervisor for guiding my with this issue.

Figure The KONVA logo

I was familiar with JavaScript, however I had never used node.js before. 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, yet it was a learning experience nonetheless.

Figure The node.js logo

### Technical achievements

My 3 biggest technical achievements are:

* Client-Server network traffic, including 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’s 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 something at the heart of a modern and responsive website.

Asynchronous functions can be quite tricky, as it means 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 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 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.

# Project planning

## Git

I made in total 58 commits to the project through its life-span as of time of writing. The commits are consistently made throughout it life-span project, with odd gaps in between. Usually these are accounted for times when I had 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’s being changed without requiring further examination of code. No branches other than the main 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.

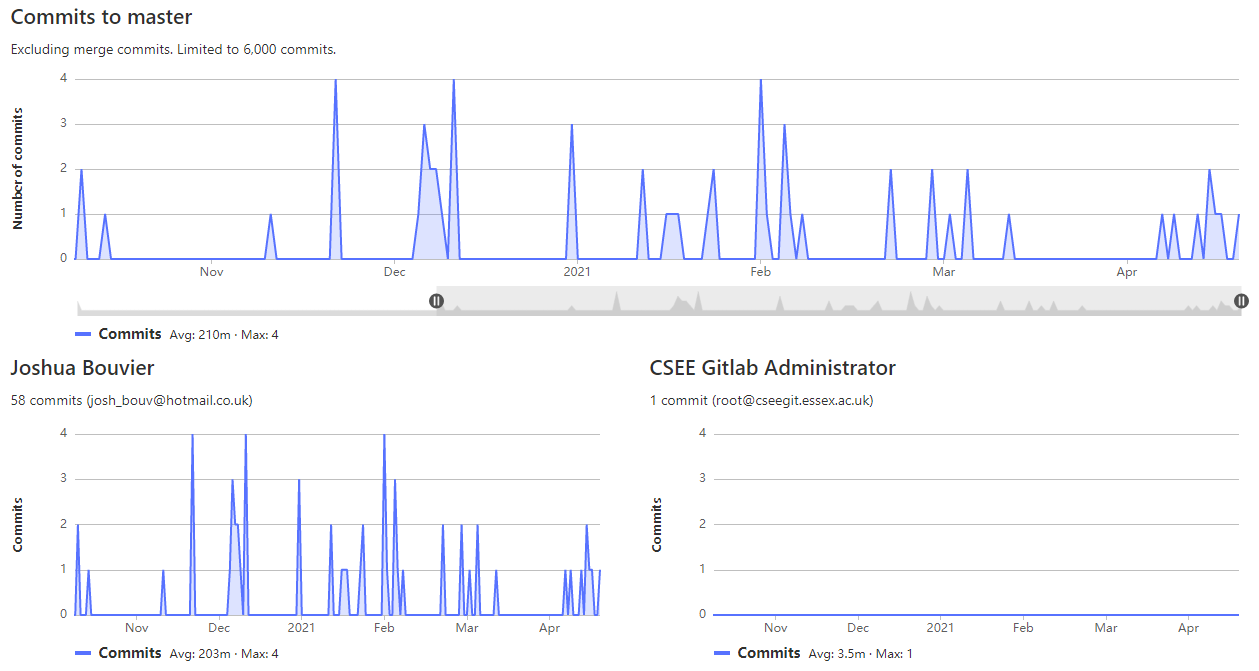


Figure GitLab commit chart, taken on 27/04/2021

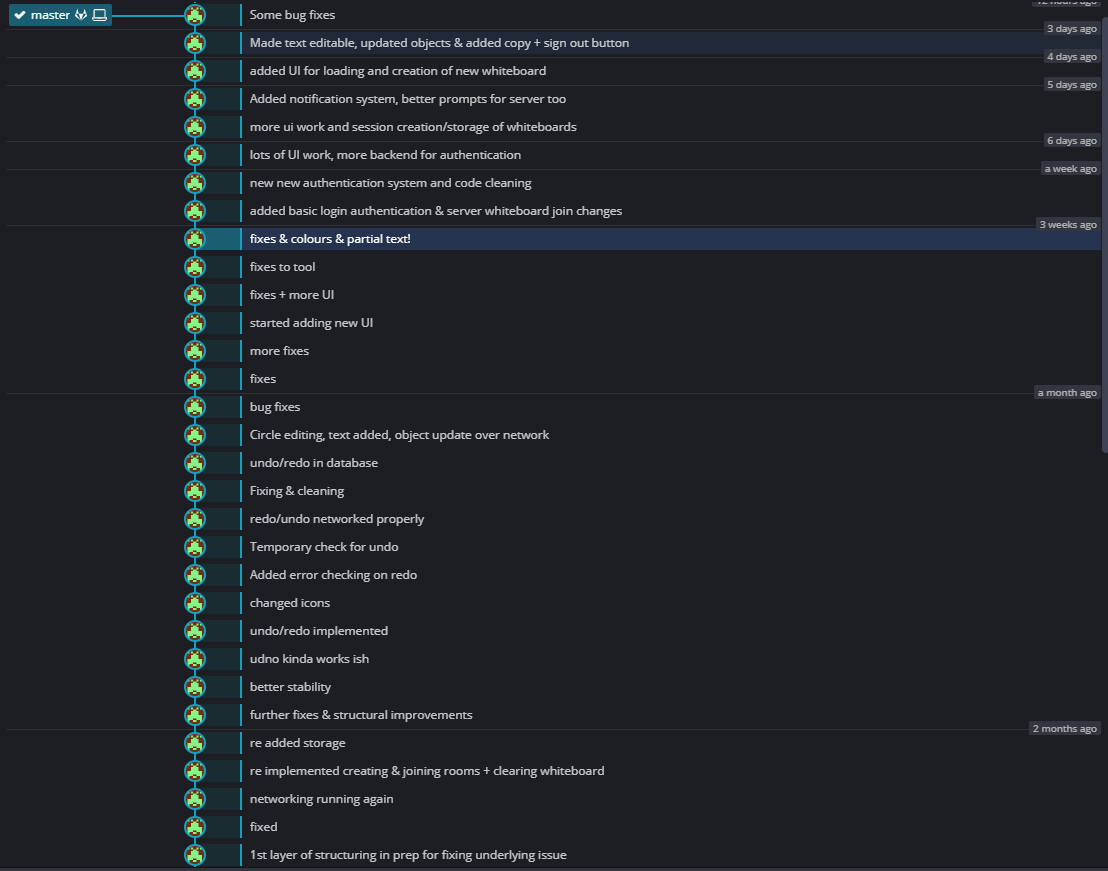


Figure Commit messages

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.

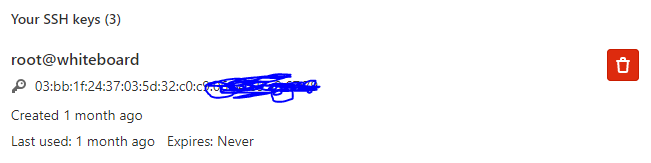


Figure Example of SSH key, actual key is hidden however for security

## Jira

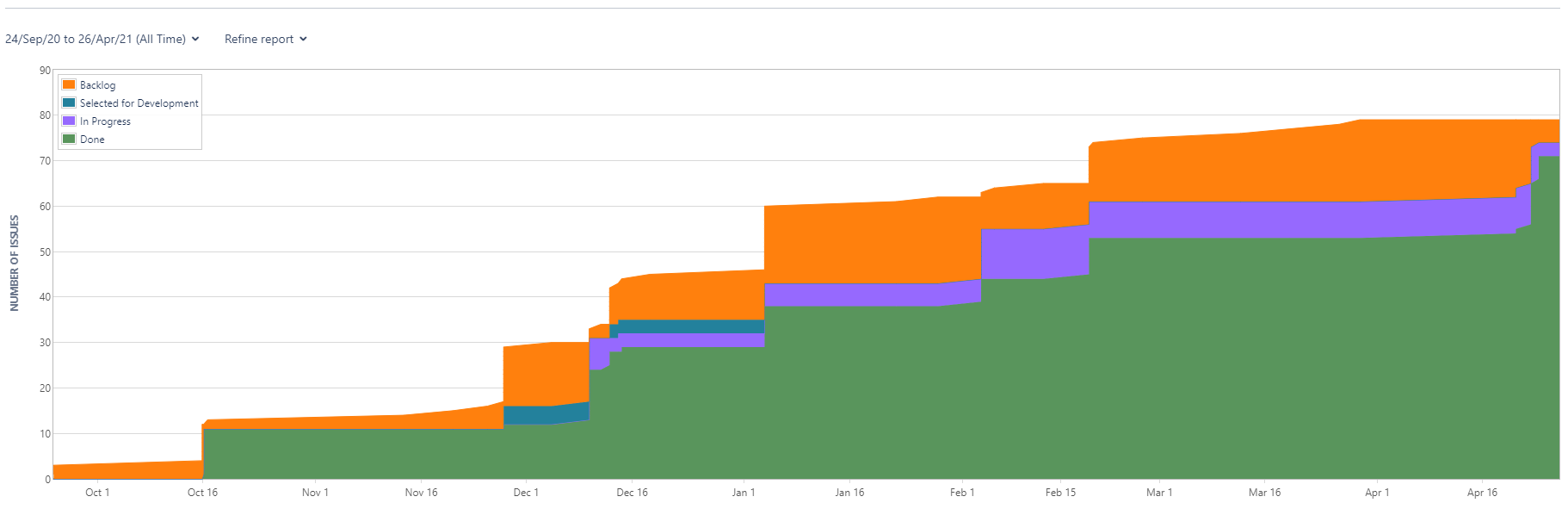


Figure Jira activity, images taken on 27/04/2021

On reflection I do wish I used Jira a bit more frequently, however as seen in figure 25 it was used through the project around every month where 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.

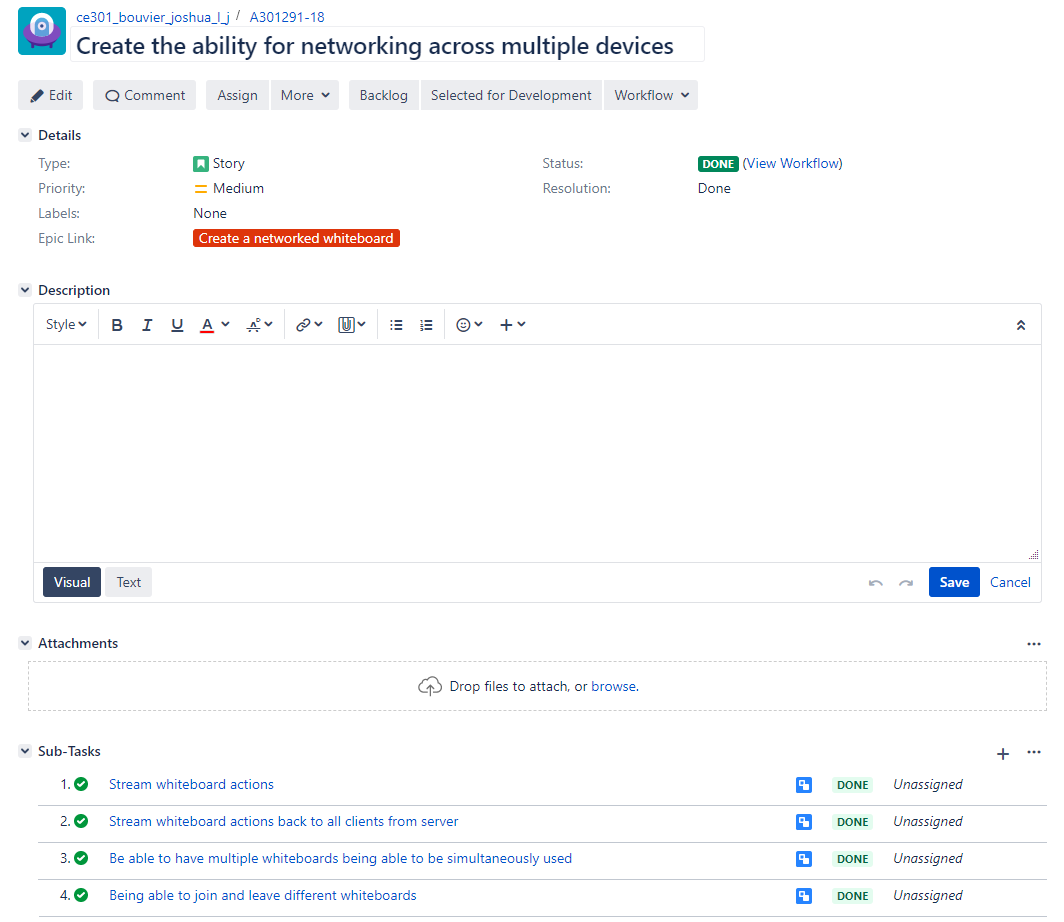


Figure Jira Story with subtasks

Some comments were made to tasks were relevant too, as seen in figure [X]

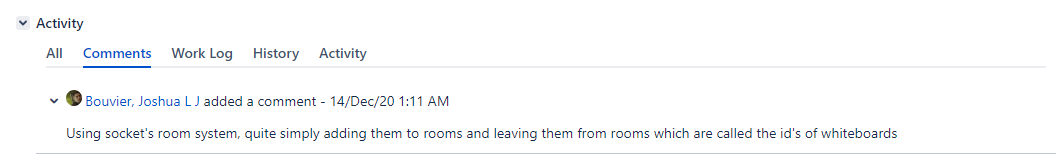


Figure Jira comment on a task

## Timeline

**October**

* MVP was made, which consisted of a HTML canvas which you could draw lines on, the lines were also drawn 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 are drawn after the joining of the room, but not data was stored on the server so any previous work could not be viewed.

**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

**December**

* Random generation of whiteboard names was added
* The addition of squares, circles and text to a local client (not networked). You 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.

**January**

* New implementation of a different canvas technology was implemented. It uses Konva instead of a HTML canvas element. It added the eraser 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.

**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

**March**

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

**April**

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

# Conclusions

## Objectives achieved

All my main objectives were achieved. I was successfully able to make a canvas, allow users to draw lines and objects onto it all over a networked environment. I was also able to add a permissions system, authentication system, the ability to save and load whiteboards and some extra small goals.

There are some things I could go onto further improve though. The first being allowing the user to choose there colour rather than using one of a pre-defined colour. I did make multiple attempts at getting this to work, but UI issues kept presenting themselves causing errors and instability meaning I wasn’t able to further pursue a better solution. Another thing I could improve is the context menu for editing objects. It’s currently quite basic and only has limited options, with more time it could be improved further.

Overall however, the project was a success, leading to a result I am very happy with.

## Objectives not achieved

I wasn’t able to add [ADD HERE]

## As a whole

[DO CONCLUSION]

CHECK PEORMSSIOSN IS WORKING BUG TESTING

# References

|  |  |
| --- | --- |
| [1] | 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. |
| [2] | 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. |
| [3] | 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. |
| [4] | 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/. |
| [5] | 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/. |
| [6] | S. Huisache, “Collaboration Software Statistics and Trends,” TrustRadius, 5 June 2020. [Online]. Available: https://www.trustradius.com/vendor-blog/collaboration-statistics-and-trends. |
| [7] | 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. |
| [8] | 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. |

# Appendices