Online Collaborative whiteboard

CE301 – capstone project

Joshua Bouvier

2021

# Acknowledgements

# 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](#_Toc70296683)

[Abstract 1](#_Toc70296684)

[Main text 3](#_Toc70296685)

[Context, motivation, why, why now 3](#_Toc70296686)

[Literature review 3](#_Toc70296687)

[Technical documentation 3](#_Toc70296688)

[Struggles 3](#_Toc70296689)

[Things I’ve learnt 3](#_Toc70296690)

[Project goals 3](#_Toc70296691)

[Main goals 3](#_Toc70296692)

[Stretch goals 4](#_Toc70296693)

[Design style 4](#_Toc70296694)

[Pre-development planning/Project planning 4](#_Toc70296695)

[Timeline 5](#_Toc70296696)

[Conclusions 6](#_Toc70296697)

[References 7](#_Toc70296698)

[Appendices 7](#_Toc70296699)

# Main text

## Context, motivation, why, why now

Whiteboards can be seen in all contexts, in a educations, business and even personal scenarios. They can often be seen as tools to procide information withas seen by presenters or also as learning tools to allow group collaborations. With an extension, we have seen interactive whiteboards in the class using software to be displayed on and draw onto with a project and whiteboard or embedded within a presentation itself.

Personally I own a wall-mounted whiteboard myself which I frequently use to scribble out information that comes into my head to help me problem solve. Whilst this has been incredibly helpful, a huge extension onto this, in an online context, is the ability to save and return back to a whiteboard later on. Personally, im also an interactive learner. Being presented by a slideshow on a unused whiteboard feels like such a wasted opportunity, and with the potential of it being able to extended onto student’s electronic devices says to me there is a huge gap for improvement.

## Literature review

## Technical documentation

https://cseegit.essex.ac.uk/ce301\_2020/ce301\_bouvier\_joshua\_l\_j/-/wikis/home

## Struggles

## Things I’ve learnt

## 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 being editable. To elobarabte on editable I mean for there position, size and in the case of text, content, to be changed.

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.

A user authentication system. Whilst this project centers 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

A networked ability. This encompasses actions such as clearing the whiteboard, undo, 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). This is because […..].

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.

Permissions on a per-user basis to allow individuals to edit the whiteboard

Live viewer list, to allow the owner of a whiteboard to see every logged-in user who is viewing the whiteboard

### Design style

I wanted it to be minimalistic yet fully accessible at the same time. I also wanted to free up as much screen space as possible for the whiteboard itself. I decided for an icon-only approach for the tools to manipulate the whiteboard on the top left side. All icons have tooltips however for any confusion.

For the user actions at the top they are all text buttons (or dropdown in case of permissions). This gives the whiteboard a clean look with obvious meaning to what the buttons do.

# Pre-development planning/Project planning

Pre development 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.

Next I decided to choose the back-end server. My 3 main options were PHP, node.js and go. 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.

I then had to choose which database to use. I have an array of options being MySQL, Orcal,e Microsoft SQL Server, PostgreSQL, MongoDB, MariaDB. Here was my comparison of the databases.

|  |  |  |
| --- | --- | --- |
| **Database management software** | **Pros** | **Cons** |
| MySQL | Very standard, lots of documentation. Have experience with this before. | Not particularly suitable for my purpose not for a dynamic website. |
| Orcale | Nothing, it’s awful! | It’s slow & costs money. |
| Microsoft SQL Server | Developed by Microsoft, fast and stable. | Suitable for a business environment, not for a dynamic website like the whiteboard. Also costs money! |
| PostgreSQL | Incredibly scalable for if there was a surge on hits to the website | Limited documentation |
| MongoDB | Incredibly versatile, fast | Bit of hassle to setup, doesn’t use SQL so little bit of a learning curve |
| MariaDB | Fast & stable | Quite a new SQL type, but a better version of MySQL essentially. |

My final choice of software to use pre-development was deciding how to communicate clients to the server. I choose socket.io.

# 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

# References

# Appendices

**Development**

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’, also called the drawLine function after every movement of the mouse whilst my variable ‘isDrawing’ was true. I then had a further even 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 was drawing lines ass it has not been pressed. Whilst it was affective in making a line, it also looked quite jagged due to it being many small liens rather than a single large one. This will be addressed later on.

I then added 7 colour buttons, which stored hex values of there respective color. When a color was pressed which would update a variable that was used in the creation of lines set its color, ‘strokeStyle’. 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 listened to 3 netmessages. 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 make a netmessage every time its called, labelled ‘drawing’, which passed the data of a line being drawn. This involved is 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 id’s 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.

My next step was to implemented 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 arrays purpose was to store lines that were being drawn, and the 2nd array was to store lines that were completed. For the former array I added the data of the line for every time drawLine, or on the server the drawing netmessage was run. I then created an addition netmessage that was called when an object had finished drawing, ‘lineCompleted’, which pushed the in-progress object to the completed object arrays. 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 at this early stage. Another issue being that it only allowed for one object to be drawn at once as I wasn’t giving an ID to each line (whilst it was technically possible all drawings all coordinates of any shapes made whilst the initial object was being drawn would be included in this object’s data.

This did allow me to implenet 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.

The reason why I didn’t jump straight into a database to store whiteboards was because I decided to use an unfamiliar database system to me, mongoDB, which required some time researching and reading docs 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 the created of mongoDB themselves, and using a tool called MongoDBCompass I connected to the database and create 2 databases. One for the whiteboards and for the users. I deleted the completedObjects array on the server created which stored completed objects but kept my array for partially completed objects. There wasn’t any purpose for this however as it didn’t do anything other than provide an overhead, but the old system still remained in place on the client.

When a line was completed it triggered a function on the server called ‘addLineToBoard’ which quite simply inserted the line provided into the database for the board provided. The board was defined by the ‘room’ varariable 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 to previously where I deleted the latest item inserted into the database.

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 with the ‘room’ defined on the client, and the server sends back a net message for each object in the board with the data of the line, to avoid sending huge amounts of data one one chunk. A function was made on the client was made to handle the netmessage and simply add it the completed objects table, and then refresh the canvas.

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 a very basic form of text. To do this I added some extra buttons each symbalising a different object to draw, ie click on on the square means you are drawing a square and when the mouse is pressed using the event listener mentioned previously, it would call a different function depending on which tools is selected. For example if square was selected, it would call a function I made called ‘drawSquare’. In the same style as drawLine, it drawed a square based on the current position of the mouse minus the position of where initially clicked. A key difference between a shape and line drawing 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, 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 inprogress objects array mentioned before. The exact same methodology was applied to drawing circles. Text was similar however it was hardcoded to be “testing123” as I had not made a UI component for an alternative for changing the 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 send the data from the database when a user loads a whiteboard. On the client in similar fashion to the drawLine function, I added netmessaes that triggered the drawing function.

At this point I relaised I had reached a limitation of the HTML canvas, and I needed to reassess my project. The issues that was presented was the inability of being able to edit shapes without complicated overhead and with a further inability to zoomed/move around the whiteboard. I explored different canvas framworks, such as Konva and Fabric.js and in the end I choose konva.

I deleted the vast majority of the client code base, and reimplemented drawin lines, squares, circles and added the ability to draw an eraser. Networking, undo, redo, clear, joining rooms all had to be removed however. This is because The structure of the project changed, aswell 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 rerender 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 syntax of how objects were stored, for the better. Previously I used to have duplicated data such as the color stored for each individual point of a line. In the new iteration I took the opportunity to be more data efficient.

I next moved into reimplementing networking, this task wasn’t too much of a feat as I was able to reuse the server-side code completely and I knew the structure of my networking order. I did run into a technical issue which took a few days to solve. Essentially performance was severely degreded on both the client and sevrer the more that was drawn. It transpired to be that 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 that socket from the orinigally socket that is broadcasting the drawing that is being created. Essentially causing an infinite loop of new sockets and sending the same data and infinly increasing amount of times. 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 rerendered, so I made it close the old connection but keep the socket open to the browser. It then makes a new connection after the rerender.

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

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

I re-added undo, but this time also added for the first time redo. These proved a bit complicated however due to the nature of rerenders and edge cases. An issue I originally ran into was that I had a state for both the counting of steps in the whiteboard and for the storage of a step for later reference. Having these after eachother caused excessive re-redners 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-redners, but don’t cause re-renders when the value is set.

I then re-added undo into the database, but also added redo. To do this I use the history step made by the undo and reinsert it as if it was drawling anew line. Any history going further than this step is then destroyed to conserve memory.

I then re added object updating over the the network, which I do by triggering a netmessage ‘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.

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 woudn’t be particularly relevant to my project, so I decided to use Material UI. I decided my UI for the whiteboard 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 buttons. All the items are contained within a container, all in a list. I then use a popover component to display the extra buttons for the drawing tools and colours.

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 nesscacry 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 serverices would do was handle the entire authentication system where you add a login button and it takes you to there 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 there sveres every X amount of seconds to see if any users had signed up, which woudn’t work for my in my scenario. This is because when you sign up and immediately make a new whiteboard, the system would error as it woudn’t be able to assign you owner of the whiteboard as my sevrers woudn’t know you existed yet. 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 and the data about that user on my database.

So in the end I opted for making my own system, however I didn’t add any email authetntication 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 in text format, rather than buttons. This is because I wanted these actions to be explicitly clear (ie preventing people from accidently signing out/making a new whiteboard). I 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 generaged by mongoDB is then used on the client and stored in its local storage, which means the browser stays logged until signed out.

I then implanted an ownership system for whiteboards which mean on the creation of a whiteboard, I inserted into a new collection labelled ‘permissions’ the whiteboard’s ID, it’s owner (if it has one), the global permission, an array for permissions for individual users and snapshot. The snapshot is used to store an image of the whiteboard which is used for loading whiteboards.