**ForeverBoard**

**Alex Pipes**

**CSC 521**

**Original Goal of Project:**

My original goal of the project was to have infinite online white boards that are live updating and shareable by url. I wanted users to be able to free hand draw and have other users see that without refreshing the page. I like to think of it as a mix between google docs and Microsoft paint. This project would be mostly used between friends chatting, leaving hidden messages on undiscovered urls, or community drawings.

**Requirements:**

* Free hand drawing
* Live updating
* All URLs valid
* Drawings remain FOREVER unless cleared by the clear button
* Drawings remain FOREVER even if server goes down
* Must be fun!
* Available on the internet

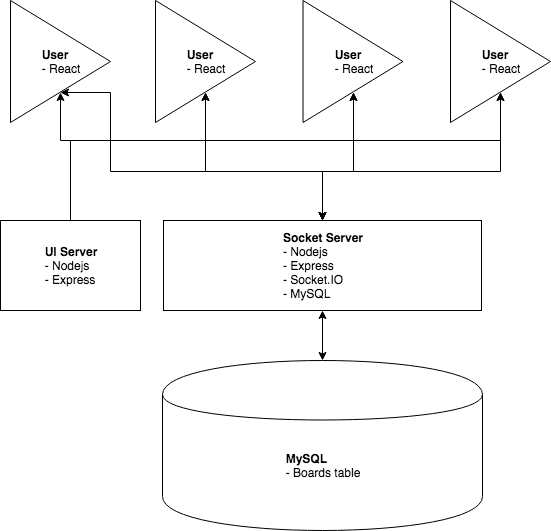
**How To Use:**

Go to foreverboard.eastus.cloudapp.azure.com. You will be presented with a white board you can draw on. To draw on the white board, pick any spot in the white space, left click and drag. Pretty simple! You can add any extension you can think of (less than 255 characters) to that URL and it will bring you to that whiteboard. If someone has been there before you’ll see what they left if they left anything at all. If someone is there now you might catch them drawing.

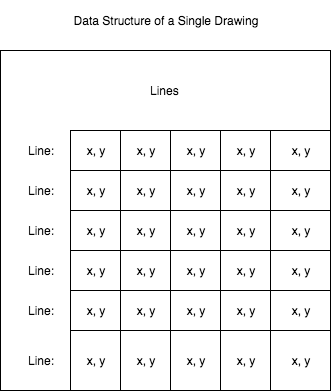
**Technology Stack:**

* React – JavaScript UI library. Used to generate SVGs and create the UI.
* Socket.io – JavaScript library to communicate between user and server both directions in real time. Used for live updating of the drawings.
* Node.JS – Allows me to run JavaScript on my local computer. I used this for my webservers.
* Express – JavaScript library to handle browser requests. This is an alternative to PHP when running on node.
* MySQL – Used to store drawings incase the server goes down.

**Architecture Diagrams:**

****

**Note:** The connection between the UI server is only in one direction but the communication from the Socket Server to the clients is bidirectional. The Socket Server can push messages to the client.

****

**Note:** On the client side the drawing is a list of list of maps but on the server side it is an array of arrays of objects. This is because the socket.io library was converting our data structure. It was not worth it to convert it server side since it will just be converted back to an array when we send it back to the client.

**Project Journal:**

When I first started the project I wanted do the servers first. I believed getting the browsers to live update would be the hardest. Early on I knew I wanted to write the project in one language to keep it simple. Since the browsers can only JavaScript this locked me in to JavaScript for the backend. This meant learning node from the ground up.

It turns out node is a simple that allows to run JavaScript outside of the browser and came pretty naturally. This was also my first exposure to a package manager (npm). I used this to download socket.io for the server, which I knew I needed since I wanted live updating. Once I had node and socket.io working I ran into my first problem. What is a drawing?

I didn’t know how to write functions on sending and receiving drawings when I didn’t even know what the data looked like. This was my first full pivot. I had to change into doing the browser side drawing first, then figure out how to send that server.

This started my first venture into React. I knew I wanted to use that as a front end framework as that is what they used at my internship. I started with a simple react page tutorial with a button on it. It was cool because react watched the files I was changing and automatically updated my browser to show my changes. This was very gratifying even if it was simple.

Now that I had react working on my machine I wanted to make the free hand drawing. I did some searching on how to make free hand drawings in the browser using React. I was able to find a good tutorial on how to generate SVGs in the browser with React. This was a great starting place and after I had finished was able to draw lines in the browser.

With the lines in the browser I wanted to start sending them to the server and then back to all clients. I had never sent data over a socket so I started socket and React tutorial. This led me to creating a second page with 3 buttons on it. Red Blue and Change background. This showed me how to emit signals to the server and receive them on both the client and server.

Now that I had free hand drawing and a live updating app I had to integrate them somehow. In order to put the live signals into the drawing app I had to become deeply familiar with the event handlers which we had done in class before. Ultimately I wanted to send the last line when someone let go of the mouse.

Since the data structure was complicated, on my first pass I sent the entire structure on mouse up (all the lines every time, not just the last one) and when the server got any data structure I sent it to all clients. I also had to write code on the client, if the server sent a drawing replace my drawing with that drawing. There were plenty of errors when I first did this and I used the browser console to see logs of why the drawing being sent back wasn’t being drawn.

It turns out the Socket library was changing the data structure of the drawing. The way we generated the SVGs from the lines variable expected it to be a list of list of maps. When we sent it to and from the server it came back as an array of arrays of objects. I had to write some client side code to convert this (which was a very difficult one liner) and it worked eventually.

This was cool and the first time we saw drawings live update but it was totally inefficient and had some bugs. For example, if two people drew at the exact same time, one line would be overridden. So my next challenge was to only send the last line. To the server and have the server only send us lines that were drawn and it is on the client to add them to the drawing.

This required me to become even more familiar with the drawing data structure and how socket.io would translate those lines. Eventually though you could draw a line, it would push only that line to the server and the server would push only that line to all clients.

Now I wanted to add the URL piece to my project. This meant sending new information to the server. I added the path as an argument and wanted the server to send that to only clients connected at that path. It turns out I am not familiar enough with how socket libraries work and it was much easier to send the line to all clients with the path it came from, and have the clients ignore lines with different paths.

After, I wanted to save the drawings for each path so I created a database. I wanted the database to be more complex but ran into some difficulties so to get things moving I created a single table that saved the entire drawing as JSON. This was supposed to be temporary but snuck itself into the final project and would the next thing I change.

I also had to do a little work to get the clients to show any existing drawing for that path if someone left something there. I created a full refresh signal that gets called on connection and sets the current drawing to the full drawing. This is very similar to the original prototype I had.

**What did I achieve:**

Live updating white board that is accessible on the internet. Users can see the line drawn as soon as the other user finishes drawing that line.

* Free hand drawing – User’s can free hand draw with only one color
* Live updating – Drawing’s update only when a line is finished
* All URLs valid - Completely Achieved
* Drawings remain FOREVER unless cleared by the clear button – Completely Achieved
* Drawings remain FOREVER even if server goes down – Stored in a MySQL database but the schema is not complex or extensible
* Must be fun! – Definitely fun.
* Available on the internet – Deploy to azure, Completely Achieved

**What didn’t I achieve:**

When I set out for this project I grand idea of what I wanted the project to be. I found that I spent way more time focusing on the technical issues than adding new features. I wanted there to be colors and erasers but it turned I spent most of my time on the networking. When I first started the database I had an interesting idea for a schema but I ran into some problems and had to cut corners. This makes the database not extensible at all.

* Colors – Can’t be just css change. Each line needs a new variable that is the color, which would a data structure change. This would require some refactoring that I couldn’t do in time.
* Erasers – An eraser can just be the color background, so this sits on top of that. This would constantly cause the image to get larger and larger. Removing points and splitting lines would also be difficult.
* DB Schema – Time Constraints, The original schema was as simple but I was never able to effectively store the lines

**What Did I Learn:**

This project taught me a lot about not only coding but designing and planning as well. I’ve done very minimal JavaScript before approaching this project but after this project I feel well versed and comfortable using JavaScript. I do want to work on front ends when I get a full time job. I realized I like seeing my changes as soon as I make them in instead of needing to save, compile, restart the server. I used JavaScript for the client and server, I found working the client more gratifying.

Learning how to use all these different JavaScript libraries was super helpful and my coding confidence has shot up. I learned how servers and clients talk to each other. Also I learned how important an extensible database is if I want to make future changes. Going into this project I felt lost because I’ve never really developed anything this size or had any of these functionalities. Reading documentation was the best way I learned how to use these libraries and frameworks, but I also watched some videos. Hearing someone explain something in their own words can simplify a lot of things.

Over the course of the project, I learned networking with socket.io, modern front end development with react, deploying a server with azure, and not quitting when I’m struggling.

**What Can I Improve:**

Some of the things I can improve on with the project are:

* Database – right now it is a single table which just includes the lines and my program makes those lines black. I want to change the database design where it can also hold the color for each line and gives the user choices, as well as store the contents the line across tables.
* Customizable title – I feel like it would add to the white bored and maybe even rules for future people who haven’t used that white board.
* Colors – I want the user to be able to mess around with different colors and see what art they can make.
* Eraser – should be able to erase certain parts of the white board. This was a lot more complex than I thought because if I split a line that must be translated into two lines in the database.
* Login – have a login to remember what white boards you have changed and maybe access private white boards
* Private boards – maybe for companies or classes, could be used for interviews or tutoring online.

Some things I can improve about my process:

* When I knew I wanted store drawings in the database. Deciding on a data layout much earlier in the project would’ve saved me some server work as well as define a better schema.
* I spent a lot time learning about what a “Stack” is before picking and truly settling on the technologies. Now that I am more familiar with actually implementing a stack and not just theorizing about it, I know more of what to look for when choosing technologies.
* Time management was something I struggled with a little. Sometimes I spent far too long focusing on something that didn’t move the project forward (such as a small bug).
* I’m still trying to become more comfortable with how callback functions work in JavaScript. The database library was asynchronous and this is a little difficult to wrap my mind around. Becoming more familiar with that style will give me a leg up in JavaScript.