Final Project Documentation

GET1033: Exploring Computational Media Literacy

Project Name: Collab-Whiteboard

# Introduction

The purpose of my final project is an online collaborative whiteboard which anyone can draw and edit. I decided on this as my final project as I wanted to learn how to create a fully functional website since we learnt a bit about HTML and CSS in one of our tutorials. I also wanted to create something interactive for the user(s), so that the user(s) can interact with my project! For this project, it is created with:

* Node.js
  + Server framework
* Express.js
  + Web Application framework
* Express-ejs-layouts
  + Allow reusing of HTML code on the web application
* Socket.io.js
  + Allows for communication between server and clients
* Jscolor.js
  + A miniature colour picker interface
* Bootstrap
  + A toolkit for creating webpages which includes HTML, CSS and JavaScript
* GitHub
  + A version control software that also helps deploy apps on Heroku
* Heroku
  + A cloud based web deployment service provider

# Features

As an online collaborative whiteboard, it includes a few features such as:

* An interactive whiteboard, which the user can draw on
  + Users can freely sketch their thoughts and whatever comes to their mind!
* Multiple users can draw contribute to the whiteboard
  + The whiteboard is collaborative, users can share their thoughts on the same canvas.
* A colour picker, which allows the changing of the brush colour
  + Thanks to Jscolour.js, users can change the brush colour to whatever they fancy.
* Brush size picker, increase or decrease width of the brush
  + Max brush size: 9!
* Various touch devices/browsers are supported
  + Tested with Safari and Chrome browsers!
* Resizing of browser window
  + The canvas will stay at the same aspect ratio! (16:9)
* Download your sketches
  + After drawing together with your friends, you can download the image! (Unsupported on Safari browsers)

# Timeline

The project timeline was from week 8 to week 13 of this semester. In each of the following weeks, I accomplished the following tasks.

* Week 8
  1. Set up node.js
  2. Set up a running server with Express.js and Socket.io.js to allow for receiving and broadcasting of messages between client and server.
* Week 9
  1. Prototyping of the canvas whiteboard, using a code example of a whiteboard from Socket.io.js package. The code example allowed us to send and receive messages to and from the server, simulating a collaborative whiteboard. This whiteboard had 5 basic colours to choose from.
  2. Creating the look and theme of the website with Bootstrap and CSS.
  3. Making the canvas whiteboard functional, allowing for drawing of lines and erasing of those lines as well.
* Week 10
  1. Created buttons to increase and decrease brush size.
  2. Created a prototype resize method to automatically scale the canvas to the user’s browser/window dimensions.
  3. Creating GitHub and Heroku accounts to deploy the prototype onto the internet.
  4. Testing of the prototype with Chrome browser.
* Week 11
  1. Overhaul of resize method.
  2. Created an about page with the resources I have used.
  3. Implementing touch functions to allow mobile users to interact with the website as well.
  4. Testing of the website with mobile browsers (Chrome and Safari)
* Week 12
  1. Bug fixes with the resize method.
  2. Implementing the download button to allow users to download their sketches.
  3. Implementing the jscolor.js colour picker which allows users to select their desired colour instead of the basic 5 colours.
* Week 13
  1. Cleaning up of code and improving the look of the website.
  2. Updated the about page.
  3. Documentation of the processes used during the project.

# Code Documentation

For this section, I will cover the files that I mainly edit and use in my web application, which includes HTML, CSS and JavaScript.

## server.js

|  |
| --- |
| **1 //init and create server**  **2**  **3 //require express**  **4 var express = require('express');**  **5 //create the app with express**  **6 var app = express();**  **7**  **8 //create a server with the express server app creation**  **9 var server = require('http').createServer(app);**  **10**  **11 //wrap the socket io to intercept connections**  **12 var io = require('socket.io')(server);**  **13 var port = process.env.PORT || 8080;**  **14**  **15 //use ejs and express layout first**  **16 var expressLayouts = require('express-ejs-layouts');**  **17 app.set('view engine', 'ejs');**  **18 app.use(expressLayouts);**  **19**  **20 //then use router**  **21 var router = require('./app/routes');**  **22 app.use(router);**  **23**  **24 //set static files (css images and javascript etc) location**  **25 app.use(express.static(\_\_dirname + '/public'));**  **26**  **27 //whenever someone connects**  **28 io.on('connection', function(socket) {**  **29 console.log(io.engine.clientsCount + ' users');**  **30 socket.on('drawing', (data) => socket.broadcast.emit('drawing', data));**  **31**  **32 //Whenever someone disconnects this piece of code executed**  **33 socket.on('disconnect', function () {**  **34 console.log(io.engine.clientsCount + ' users');**  **35 });**  **36 });**  **37**  **38 //start server**  **39 server.listen(port, function(){**  **40 console.log('app started');**  **41 });** |

For the server.js, we are using node.js to run this JavaScript file, which will allow us to create a server. Express.js and Node.js are used to help us create a web application, at lines 4 to 6.

At line 11, Socket.io.js then allows us to wrap the server with a connection interceptor, which can allow us to read those connections and interact with them. The port number is selected, at line 13, to dictate on which port the app runs on.

EmbeddedJavaScript (EJS) and Express-EJS-Layouts are used to help us reuse some portions of our webpages across the website itself, at lines 16 to 18.

A router is used to help us navigate the pages within our website without using very long links, created at line 21. Line 25 allows us to set a location where we can put our files to be used, such as images and css files.

Line 27 to 36 will allow the server to handle connections and broadcast the actions emitted by the client, elaborated later. Lastly, line 39 onwards starts the server!

## router.js

|  |
| --- |
| **1 //require express, path**  **2 var express = require('express');**  **3 var path = require('path');**  **4**  **5 //create router object**  **6 var router = express.Router();**  **7**  **8 //export router**  **9 module.exports = router;**  **10**  **11 //route for homepage**  **12 router.get('/', function(req, res) {**  **13 res.render('pages/index');**  **14 });**  **15**  **16 //route for about page**  **17 router.get('/about', function(req, res) {**  **18 res.render('pages/about');**  **19 });** |

For router.js, it is just a simple file which we use to create the router object in server.js. This file contains all the routes that our webpage uses, including the page directory and the shortened form. In this web application, you can see that my webpage is quite small, only having 2 pages. The index page, which is the whiteboard itself, is shortened to “/”. The about page is shortened to “/about”. This allows us to reduce the link to a shorter version, reducing errors when we are writing the HTML in subsequent web pages!

## layout.ejs

|  |
| --- |
| **1 <!DOCTYPE html>**  **2 <html lang='en'>**  **3**  **4 <head>**  **5 <meta charset="UTF-8">**  **6**  **7 <!-- the bootstrap stuff here-->**  **8 <!-- Latest compiled and minified CSS -->**  **9 <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">**  **10**  **11 <!-- my own css -->**  **12 <link rel="stylesheet" type="text/css" href="/css/styles.css">**  **13**  **14 <!--Title of webpage-->**  **15 <title>Collaborative Whiteboard</title>**  **16 </head>**  **17** |

The layout.ejs is where we create a standardized layout for our website utilizing EJS and Express-EJS-Layouts. It is similar to a HTML file. This allows us to have a nice standard navigation bar at the top of our browser, where we can use to customize on each page.

At line 9, I use the bootstrap’s CSS to provide my website with a basic theme and framework so that I can focus on the building of the application logic.

At line 12, I then incorporate my own CSS to further customize the website to my liking!

|  |
| --- |
| **18 <body>**  **19 <header>**  **20 <nav class="navbar navbar-fixed-top navbar-inverse">**  **21 <div class="container-fluid">**  **22**  **23 <div class="navbar-header">**  **24 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">**  **25 <span class="icon-bar"></span>**  **26 <span class="icon-bar"></span>**  **27 <span class="icon-bar"></span>**  **28 </button>**  **29 <a class="navbar-brand" href="/about">Collaborative Whiteboard</a>**  **30 </div>**  **31**  **32 <%- defineContent('navBarColor') %>**  **33**  **34 </div>**  **35**  **36 </nav>**  **37 </header>**  **38 <%- defineContent('body') %>**  **39 </body>**  **40**  **41 <%- defineContent('script') %>**  **42 <!-- javascript for bootstrap and jquery for responsiveness-->**  **43 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>**  **44 <!-- Latest compiled and minified JavaScript -->**  **45 <script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>**  **46 </html>** |

At line 20, I create a navigation bar for my website, which uses the bootstrap navigation bar properties. Line 24 shows that I create a button to toggle the navigation bar, so for smaller screens, the navbar will be able to expand and collapse.

At line 29, the navbar brand corresponds to the clickable brand that is on the navbar.

Line 32, 38 and 41 shows a “defineContent” method, which will pull in the corresponding HTML code on the page that is using this layout.ejs.

At the end, we add in the jQuery and bootstrap JavaScript files to allow for our web application to have responsiveness for some elements, such as the collapsing of the navigation bar.

## index.ejs

|  |
| --- |
| **1 <%- contentFor('body') %>**  **2 <container>**  **3 <canvas class="whiteboard" width="1024" height="576"background="white"></canvas>**  **4 </container>**  **5** |

For the index.ejs file, we have our main whiteboard. At line 1, there is the “contentFor” method, which corresponds to the “defineContent” method in layout.js with the parameter “body”. This will create a canvas object which the user will draw on.

|  |
| --- |
| **6 <%- contentFor('navBarColor') %>**  **7**  **8 <div class="collapse navbar-collapse" id="myNavbar">**  **9 <form class="form-inline">**  **10**  **11 <button**  **12 class="btn navbar-btn jscolor {valueElement:null,value:'0000FF',zIndex:9999}" name="colourpicker">**  **13 Choose a Colour!**  **14 </button>**  **15**  **16 <button type="button" class="btn navbar-btn btn-default" name="eraser">Eraser!</button>**  **17**  **18 <div class="btn-group" role="group">**  **19 <button type="button" class="btn btn-default navbar-btn -" aria-label="Up" name="brushDown">**  **20 <span class="glyphicon glyphicon-triangle-bottom"></span>**  **21 </button>**  **22 <button type="button" class="btn btn-default navbar-btn disabled" name="brushSize">**  **23 Brush Size: 2**  **24 </button>**  **25 <button type="button" class="btn btn-default navbar-btn +" aria-label="Down" name="brushUp">**  **26 <span class="glyphicon glyphicon-triangle-top"></span>**  **27 </button>**  **28 </div>**  **29**  **30 <div class="btn-group navbar-right" role="group">**  **31 <a type="button" class="btn navbar-btn btn-success" name="download">**  **32 Download JPEG!**  **33 </a>**  **34 </div>**  **35**  **36 </form>**  **37 </div>**  **38**  **39**  **40**  **41 <%- contentFor('script') %>**  **42 <script src = "/socket.io/socket.io.js"></script>**  **43 <script>**  **44 //creates a socket and allows interaction with server**  **45 var socket = io();**  **46 //run canvas.js**  **47 </script>**  **48**  **49 <!-- my canvas javascript-->**  **50 <script src="/javascript/canvas.js"></script>**  **51 <!-- colourpicker -->**  **52 <script src="/javascript/jscolor.js"></script>** |

Continued from last page, Lines 6 to 37 is the codes which create the navigation bar elements, such as the colour picker, the brush size selector, the eraser button and the download image button. These elements are given unique names in their “name” attribute which is used in the canvas.js file!

Lines 41 to 47 adds the socket.io.js file, which allows us to create a socket object that we will use to send and receive instructions to the server.

Line 50 allows me to import my script to this webpage, utilizing its methods to help us interact with the whiteboard!

Line 52 is a jscolor.js file which I imported to this project to allow users to pick their colour from a colour picker that allows for many colours!

## about.ejs

|  |
| --- |
| **1 <%- contentFor('body') %>**  **2**  **3 <section class="body">**  **4 <!--do some stuff for about.html here-->**  **5 <div class="container">**  **6 <div class="jumbotron">**  **7 <h1 class="display-3">Collaborative Whiteboard!</h1>**  **8 <p class="lead">Made by Lim Shan Jing, for GET1033 final project component.</p>**  **9 <a href="/" class="btn btn-success btn-xlarge">Click me to go back to the whiteboard!</a>**  **10**  **11 <hr class="my-4">**  **12 <p>Credits to:**  **13**  **14 <dl>**  **15 <dt>Building a node.js website</dt>**  **16 <dd> - https://scotch.io/courses/build-a-nodejs-website/course-introduction</dd>**  **17 <dt>Socket.io (handles connection from client to server)</dt>**  **18 <dd> - https://github.com/socketio/socket.io</dd>**  **19 <dt>Using Socket.io together with express</dt>**  **20 <dd> - http://www.programwitherik.com/socket-io-tutorial-with-node-js-and-express/</dd>**  **21 <dt>Socket.io collaborative whiteboard example</dt>**  **22 <dd> - https://github.com/socketio/socket.io/tree/master/examples/whiteboard</dd>**  **23 <dt>Creating a web based drawing application</dt>**  **24 <dd> - https://code.tutsplus.com/tutorials/how-to-create-a-web-based-drawing-application-using-canvas--net-14288</dd>**  **25 <dt>Bootstrap UI</dt>**  **26 <dd> - http://getbootstrap.com/docs/4.0/getting-started/introduction/</dd>**  **27 <dd> - http://blog.getbootstrap.com/2016/07/25/bootstrap-3-3-7-released/</dd>**  **28 <dt>Download tag for html</dt>**  **29 <dd> - https://www.w3schools.com/tags/att\_a\_download.asp</dd>**  **30 <dt>Touch event handling</dt>**  **31 <dd> - http://bencentra.com/code/2014/12/05/html5-canvas-touch-events.html</dd>**  **32 <dt>Colour picker API</dt>**  **33 <dd> - http://jscolor.com/</dd>**  **34 <dt>Github</dt>**  **35 <dd> - https://github.com/</dd>**  **36 <dt>Heroku</dt>**  **37 <dd> - https://www.heroku.com/</dd>**  **38 </dl>**  **39**  **40 <hr class="my-4">**  **41 <a href="https://github.com/shanjing012/shanjing012.github.io" class="btn btn-success btn-lg">Github</a>**  **42 <h7>Click here for the Github repository! (Documentation here!)</h7>**  **43**  **44 <hr class="my-4">**  **45 <h7> <a rel="license" href="http://creativecommons.org/licenses/by/4.0/"> <img alt="Creative Commons License" style="border-width:0" src="https://i.creativecommons.org/l/by/4.0/88x31.png" /></a> This work is licensed under a <a rel="license" href="http://creativecommons.org/licenses/by/4.0/">Creative Commons Attribution 4.0 International License</a> </h7>**  **46 </p>**  **47 </div>**  **48 </div>**  **49**  **50 </section>** |

On about.ejs, I add all the resources I used in the creation of this web application! I add a button which allows the user to go back to the whiteboard. I also include a link to the GitHub repository, as well as added a creative commons license!

## styles.css

|  |
| --- |
| **1 head {**  **2 position: fixed;**  **3 }**  **4**  **5 body {**  **6 background-color: gray;**  **7 }**  **8**  **9 .btn-xlarge {**  **10 padding: 20px 28px;**  **11 font-size: 24px; //change this to your desired size**  **12 line-height: normal;**  **13 -webkit-border-radius: 8px;**  **14 -moz-border-radius: 8px;**  **15 border-radius: 8px;**  **16 }**  **17**  **18 .whiteboard {**  **19 margin-top: 50px;**  **20 margin-left: auto;**  **21 margin-right: auto;**  **22**  **23 display: block;**  **24 position: absolute;**  **25 top: 0;**  **26 bottom: 0;**  **27 left: 0;**  **28 right: 0;**  **29 background-color: white;**  **30 }**  **31**  **32 .jumbotron {**  **33 margin-top: 75px;**  **34 }**  **35**  **36 .navbar-right {**  **37 padding-right: 10px;**  **38 }**  **39**  **40 html {**  **41 position: relative;**  **42 min-height: 100%;**  **43 max-height: 100%;**  **44 }** |

On styles.css, I customize the elements which I have used in the web application further. I also added a btn-xlarge style for the button on the about.ejs page!

## canvas.js

In canvas.js is where the main functionalities of the web application are coded!

|  |
| --- |
| **1 'use strict';**  **2**  **3 (function() {**  **4**  **5 //CURRENT BRUSH OBJECT**  **6 var current = {**  **7 color: 'black',**  **8 brushS: 2,**  **9 eraser: false,**  **10 drawing: false**  **11 };**  **12**  **13 //Get elements from document.**  **14 var brushUp = document.getElementsByName('brushUp')[0];**  **15 var brushDown = document.getElementsByName('brushDown')[0];**  **16 var brushSize = document.getElementsByName('brushSize')[0];**  **17 var colorpicker = document.getElementsByName('colourpicker')[0];**  **18 var eraser = document.getElementsByName('eraser')[0];**  **19 var canvas = document.getElementsByClassName('whiteboard')[0];**  **20 var download = document.getElementsByName('download')[0];**  **21 var context = canvas.getContext('2d');**  **22** |

At the start of canvas.js, I create a current object, which stores the attributes of the current brush, in lines 6 to 11. I then get and assign the elements to variables, which are buttons from the document, which is the HTML page.

|  |
| --- |
| **23 //set background**  **24 context.fillStyle = 'white';**  **25 context.fillRect(0, 0, canvas.width, canvas.height);**  **26 context.fillStyle = current.color;**  **27**  **28**  **29 //EVENT LISTENER: Color picker**  **30 colorpicker.addEventListener('click', function() {**  **31 current.eraser = false;**  **32 }, false);**  **33**  **34 //EVENT LISTENER: Eraser**  **35 eraser.addEventListener('click', function() {**  **36 current.eraser = true;**  **37 }, false);**  **38** |

Next, the canvas background is set to white. Then, I create event listeners, which will wait until an event is fired upon the element. An example is on line 35, where the event listener waits for clicks on the eraser element. When the eraser element is clicked, the function will run, which sets current.eraser to true!

|  |
| --- |
| **39 //EVENT LISTENERS: Changing brush size**  **40 brushUp.addEventListener('click', function() {onBrushUpdate(1)}, false);**  **41 brushDown.addEventListener('click', function() {onBrushUpdate(-1)}, false);**  **42**  **43 //FUNCTION: updating brush size**  **44 function onBrushUpdate(e){**  **45 var newBrush = current.brushS;**  **46 current.brushS = current.brushS + e;**  **47 if((current.brushS > 9) || (current.brushS < 1))**  **48 {**  **49 current.brushS = newBrush;**  **50 }**  **51 brushSize.innerText = "Brush Size: " + current.brushS;**  **52 }**  **53** |

Same as above, creating event listeners which increase or decrease the brush size!

|  |
| --- |
| **54 //EVENT LISTENER: Download**  **55 download.addEventListener('click', onDownload, false);**  **56**  **57 //FUNCTION: Download image as jpeg**  **58 function onDownload(e){**  **59 var img = canvas.toDataURL('image/jpeg');**  **60 this.download = 'image';**  **61 this.href = img;**  **62 }**  **63** |

Here is the event listener for the download button, which allows the user to download the canvas as a jpeg file. Unfortunately, this feature is not supported on Safari browsers!

|  |
| --- |
| **64 //EVENT LISTENER: Canvas**  **65 //when click**  **66 canvas.addEventListener('mousedown', onMouseDown, false);**  **67 //when not clicking**  **68 canvas.addEventListener('mouseup', onMouseUp, false);**  **69 //when mouse leaves the canvas**  **70 canvas.addEventListener('mouseout', onMouseUp, false);**  **71 //when mouse moves around**  **72 canvas.addEventListener('mousemove', throttle(onMouseMove, 5), false);**  **73**  **74 //EVENT LISTENER: Touch commands**  **75 //on touch start**  **76 canvas.addEventListener('touchstart', function(e) {**  **77 e.preventDefault();**  **78 //create a mouse event with the touch handler**  **79 var touch = e.touches[0];**  **80 var mouseEvent = new MouseEvent("mousedown", {**  **81 clientX: touch.clientX,**  **82 clientY: touch.clientY**  **83 });**  **84 onMouseDown(mouseEvent);**  **85 }, false);**  **86 //on touch move**  **87 canvas.addEventListener('touchmove', function(e) {**  **88 e.preventDefault();**  **89 //create a mouse event with the touch handler**  **90 var touch = e.touches[0];**  **91 var mouseEvent = new MouseEvent("mousemove", {**  **92 clientX: touch.clientX,**  **93 clientY: touch.clientY**  **94 });**  **95 onMouseMove(mouseEvent);**  **96 }, false);**  **97 //on touch end**  **98 canvas.addEventListener('touchend', function(e) {**  **99 e.preventDefault();**  **100 //create a mouse event with the touch handler**  **101 current.drawing = false;**  **102 }, false);**  **103**  **104** |

For this snippet of code, I am creating event listeners for the canvas itself. These include clicking and touch events for the canvas, which is how I was able to incorporate touch capabilities to the canvas. I did not have to create touch event listeners for the other elements because browsers will automatically translate touch events to click events, such as on mobile! But for the canvas itself, I had to consider that the user could move his finger on the canvas, which the click event listeners were unable to handle. Thus, through the event listener with “touchMove” parameter, I can track what the user’s finger position was! These event listeners will call the methods below!

|  |
| --- |
| **105 //FUNCTION: Drawing of line**  **106 function drawLine(x0, y0, x1, y1, current, emit){**  **107**  **108 context.beginPath();**  **109 if(current.eraser == true)**  **110 {**  **111**  **112 context.fillStyle = 'white';**  **113 context.arc(x1, y1, 30, 0, Math.PI\*2, true);**  **114 context.fill();**  **115 }**  **116 else**  **117 {**  **118 context.lineCap = "round";**  **119 context.strokeStyle = current.color;**  **120 context.moveTo(x0, y0);**  **121 context.lineTo(x1, y1);**  **122 context.lineWidth = parseInt(current.brushS);**  **123 context.stroke();**  **124 }**  **125**  **126 context.closePath();**  **127**  **128 if (!emit) { return; }**  **129 var w = canvas.width;**  **130 var h = canvas.height;**  **131**  **132 //emit to all other sockets**  **133 socket.emit('drawing', {**  **134 x0: x0 / w,**  **135 y0: y0 / h,**  **136 x1: x1 / w,**  **137 y1: y1 / h,**  **138 current: current**  **139 });**  **140**  **141 }**  **142** |

This is the drawLine method, which is called every time the canvas requires drawing. The x and y coordinates are passed in, together with the current object which represents the brush and its attributes. If current.eraser is true, meaning it is currently set to erase, it will draw a white circle centred on the coordinates passed in instead. Emit is a variable passed in to check whether it is a client or server message, if it is true, it means that the method is called by the client. Thus, we must send a message to the server, or “emit” the message, to draw the same line on the other clients, with the same attributes. This is how we allow all clients to collaborate on the same canvas!

|  |
| --- |
| **143 //FUNCTION: on click**  **144 function onMouseDown(e){**  **145**  **146 //change color when drawing.**  **147 current.color = colorpicker.jscolor.toHEXString();**  **148**  **149 current.drawing = true;**  **150 current.x = e.clientX - canvas.offsetLeft;**  **151 current.y = e.clientY - canvas.offsetTop;**  **152 }**  **153**  **154 //FUNCTION: on unclick**  **155 function onMouseUp(e){**  **156 if (!current.drawing) { return; }**  **157 current.drawing = false;**  **158 drawLine(current.x, current.y, e.clientX - canvas.offsetLeft, e.clientY - canvas.offsetTop, current, true);**  **159 }**  **160** |

These are methods that activate when the user clicks or lets go of the mouse button. On click, it will also set the current brush colour to the colour selected by the colour picker! On unclick, it will call the drawLine method.

|  |
| --- |
| **161 //FUNCTION: on movement of mouse method**  **162 function onMouseMove(e){**  **163 if (!current.drawing) { return; }**  **164**  **165 drawLine(current.x, current.y, e.clientX - canvas.offsetLeft, e.clientY - canvas.offsetTop, current, true);**  **166 current.x = e.clientX - canvas.offsetLeft;**  **167 current.y = e.clientY - canvas.offsetTop;**  **168 }**  **169**  **170 // limit the number of events per second**  **171 function throttle(callback, delay) {**  **172 var previousCall = new Date().getTime();**  **173 return function() {**  **174 var time = new Date().getTime();**  **175 if ((time - previousCall) >= delay) {**  **176 previousCall = time;**  **177 callback.apply(null, arguments);**  **178 }**  **179 };**  **180 }**  **181** |

On mouse movement is the same as the mouse unclick method, it will call drawLine method, with the coordinates of the client passed in! For the second method, throttle, this will limit the number of calls per second, preventing too many methods from being called, overloading the client/server.

|  |
| --- |
| **182 //SOCKET HANDLER: when others are drawing, do onDrawingEvent.**  **183 socket.on('drawing', onDrawingEvent);**  **184**  **185 //FUNCTION: when other people draw, this event is called**  **186 function onDrawingEvent(data){**  **187 var w = canvas.width;**  **188 var h = canvas.height;**  **189 drawLine(data.x0 \* w, data.y0 \* h, data.x1 \* w, data.y1 \* h, data.current);**  **190 }**  **191** |

For the socket, this function is called when the socket receives an ‘drawing’ event. In server.js, lines 27 to 36 shows the server broadcasting an event to all connected clients. This function will take in the broadcasted ‘drawing’ message and act on it, with the “data” object provided by the broadcast! Therefore, clients can send the brush strokes they have drawn to the server, who will broadcast the message to everyone!

|  |
| --- |
| **192 //resize window**  **193 window.addEventListener('resize', onResize, false);**  **194 onResize();**  **195** |

Here we add a resize event listener which will execute the onResize method when the browser/window is resized. We also run the onResize method once first to adjust the canvas to the browser/window size.

|  |
| --- |
| **196 function onResize() {**  **197 //save the image and post it again once it**  **198 //https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/putImageData**  **199 var imgData = context.getImageData(0, 0, canvas.width, canvas.height);**  **200 //need to create an image object to store the image then scale it toward the image itself.**  **201**  **202 if(window.innerHeight < 576 + 50 || window.innerWidth < 1024)**  **203 {**  **204 //scale down**  **205 //take into account the navbar**  **206 var scaleX = window.innerWidth / canvas.width;**  **207 var scaleY = (window.innerHeight - 50) / canvas.height;**  **208**  **209 var scale;**  **210 if(scaleX < scaleY)**  **211 scale = scaleX;**  **212 else**  **213 scale = scaleY;**  **214**  **215 canvas.height = canvas.height \* scale;**  **216 canvas.width = canvas.width \* scale;**  **217 //context.putImageData(imgData, 0, 0);**  **218 }**  **219 else**  **220 {**  **221 //scale back up to 1024 and 576**  **222 var scaleX = 1024 / canvas.width;**  **223 var scaleY = 576 / canvas.height;**  **224**  **225 context.scale(scaleX, scaleY);**  **226 canvas.height = canvas.height \* scaleY;**  **227 canvas.width = canvas.width \* scaleX;**  **228 //context.putImageData(imgData, 0, 0);**  **229 }**  **230 //draw the scaled image**  **231 //create white background first**  **232 context.fillStyle = 'white';**  **233 context.fillRect(0, 0, canvas.width, canvas.height);**  **234 context.fillStyle = current.color;**  **235 context.putImageData(imgData, 0, 0);**  **236 }**  **237**  **238 })();** |

Lastly, the resize method. How it works is that it will save the canvas as a imageData variable first. Then, it will check whether the new size of the window is able to accommodate the canvas at its max width and height (1024:576), accounting for the navigation bar. If it is unable, it will scale the canvas down by calculating the scale and transforming the canvas width and height. Else, the canvas will be scaled back to its max size. Lastly, it will fill the background with white again, then put the imageData onto the canvas.