Personal Portfolio

N9683755 – John Board

As a developer on this project, I spent most my time writing code. My contributions can be broken down into roughly four categories:

* Server Administration & DevOps
* Databasing
* Backend Development
* Front End Design & Development

I drew from past experience in databasing & UI/UX subjects, as well as knowledge from past work experience in server administration and development operations.

I contributed a huge amount of code to this project, which would be impractical to go through each file in depth. I suggest for a full look at what I did, look at the git commits. I also helped other developers (Ash Clarke) with some of their code, but the work was primarily theirs.

The following eight artefacts from Sprints 1 & 2 should demonstrate the above four categories of contribution.

**A LIVE DEMO OF THIS APP WILL BE MADE AVAILABLE UNTIL THE 20th OF JUNE, AND IS ACCESABLE AT:**

<http://eante.com.au:3030/>

# Sprint 1

## Artefact 1

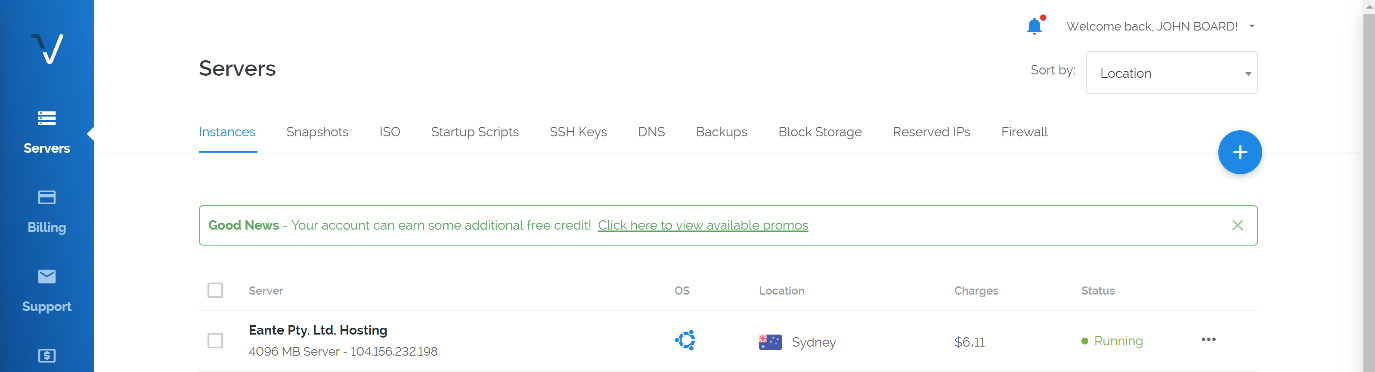
*Type: Server Administration & Development Operators*

First of all, I set up a git repository – <https://github.com/IFB299G15/MediaVault>. I maintained the “curator” of this repository during the project.

Secondly, I set up a web server on which we could develop and test on. This process involved:

1. Creating a new user on my VPS server
2. Installing Meteor (our development framework)
3. Creating a DNS entry

I used my company’s development server. It runs Vultr

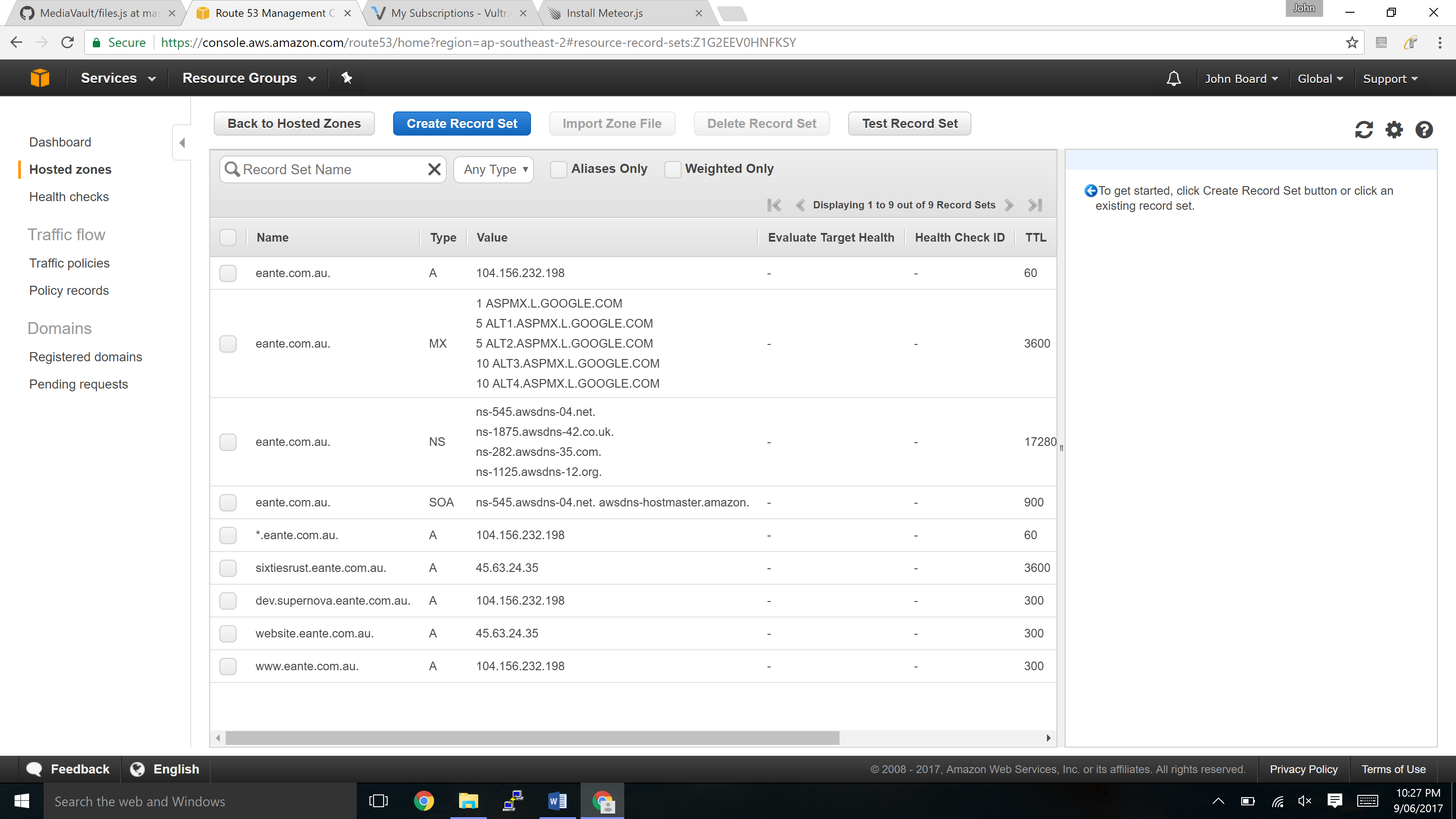


On the server I created a new user using the command sudo useradd, and then set the password using sudo passwd.

I installed Meteor by following the guide on their website

<https://www.meteor.com/install>

I’ve been having issues with my current domain host, so I transferred my domain from GoDaddy to Amazon Route53, and added the eante.com.au entries (which we used in the final product).



For this early stage of the project, I just git pulled down what we had in the repo and ran that. It ran at [eante.com.au:3030](http://eante.com.au:3030)

## Artefact 2

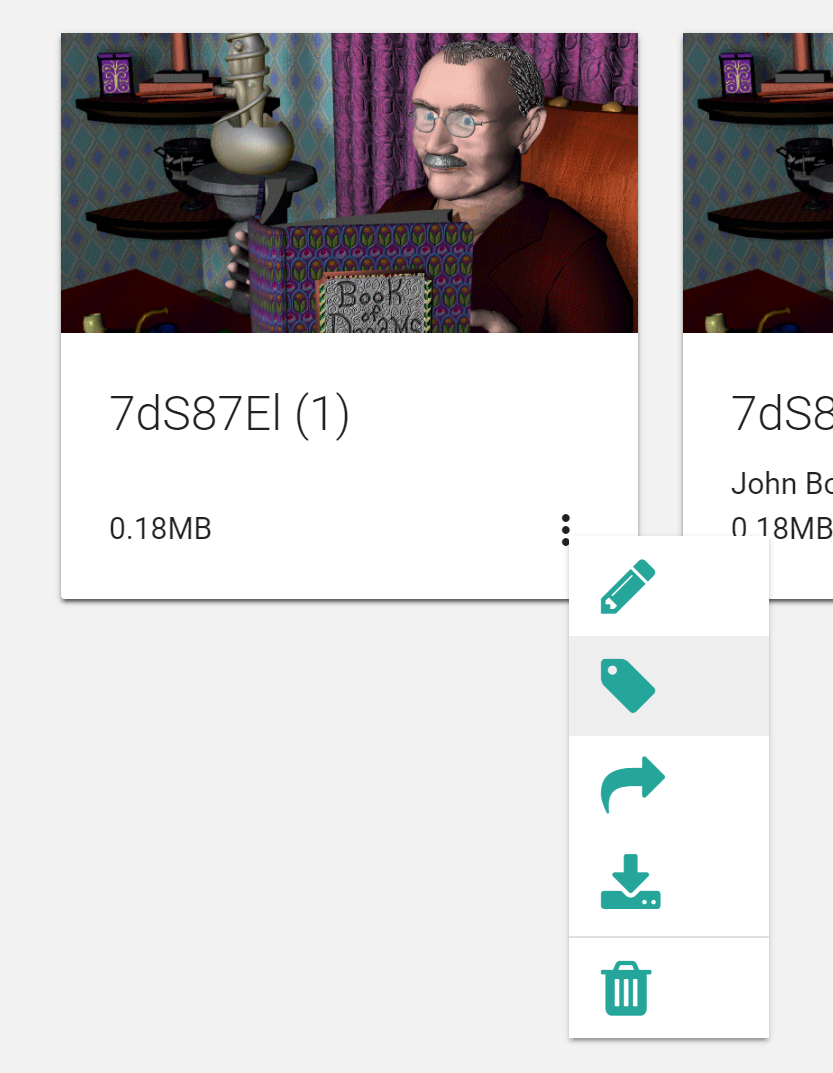
*Type: Databasing (Tags)*

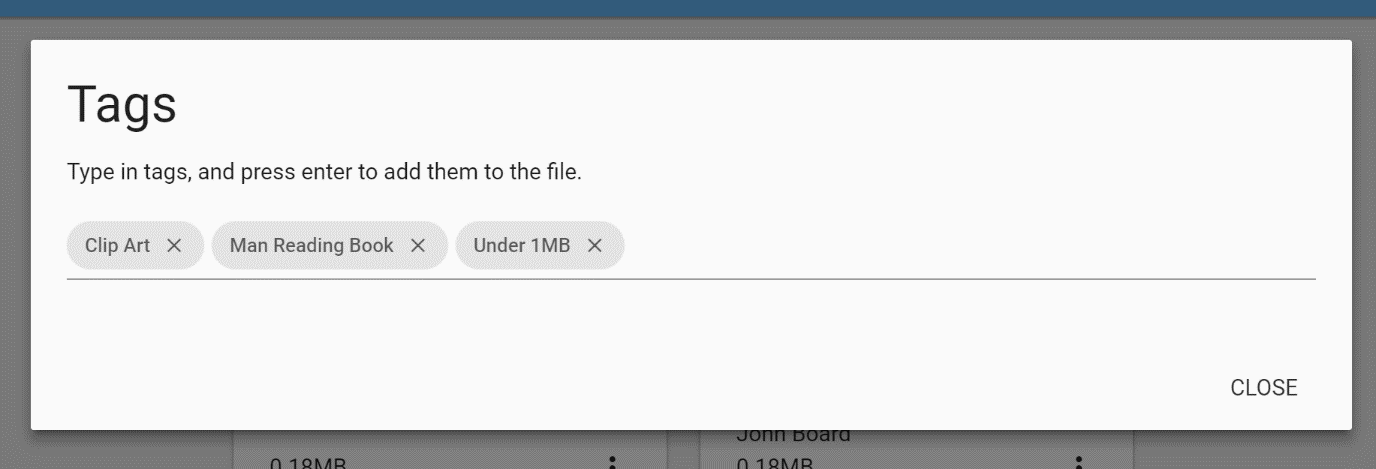
Meteor uses MongoDB for all it’s databasing. MongoDB is naturally quite flexible in how you store data, but as I’ve learnt from previous subjects, it’s really important to structure your data. As such I wrote the database code for tagging files to include a schema.



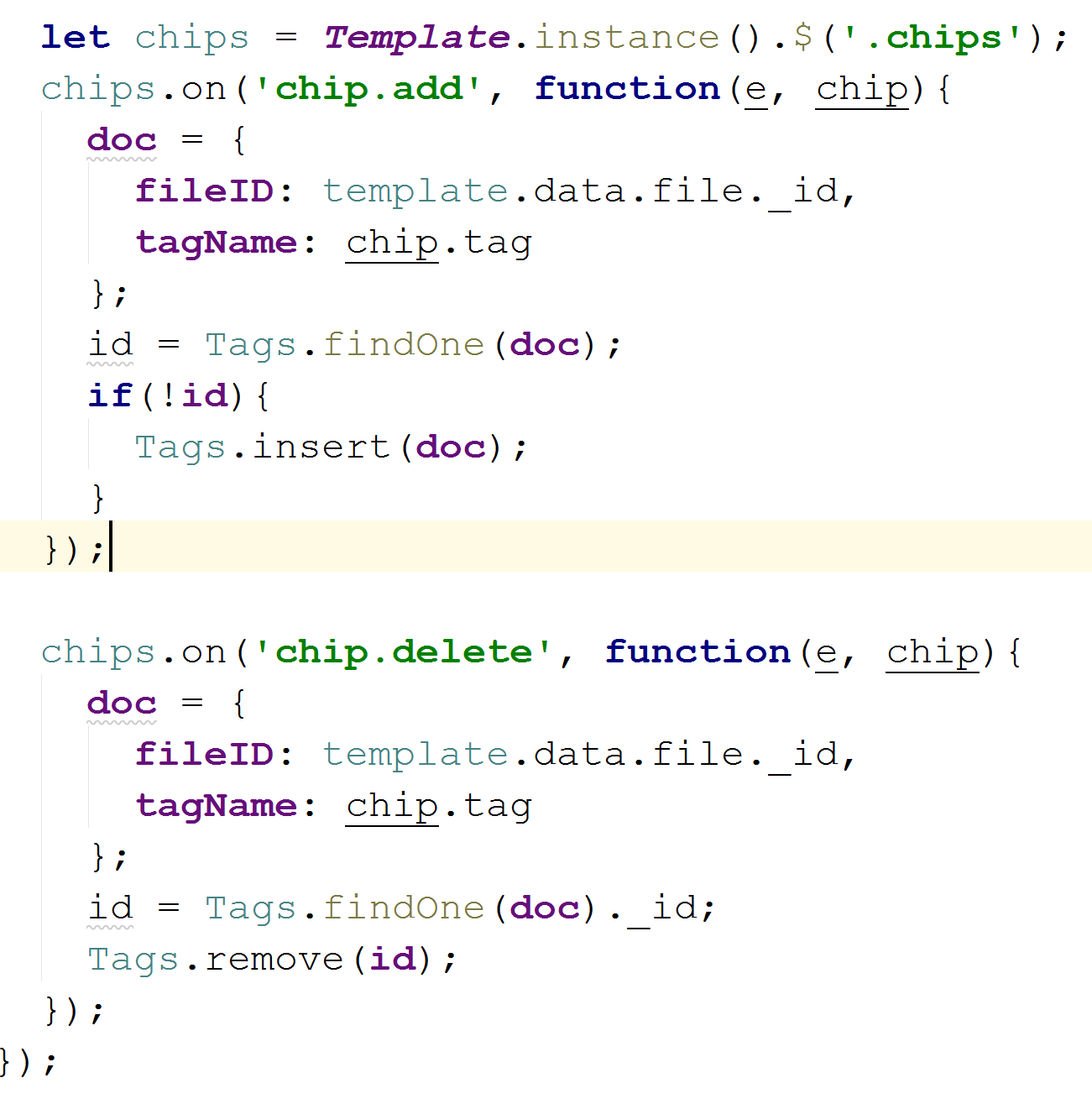
This code can be found under /imports/api/tags/tags.js

This resulted in the following UI:





Code for the UI managing tags can be found under /imports/client/ui/components/tagEditor/tagEditor.js in the repo:



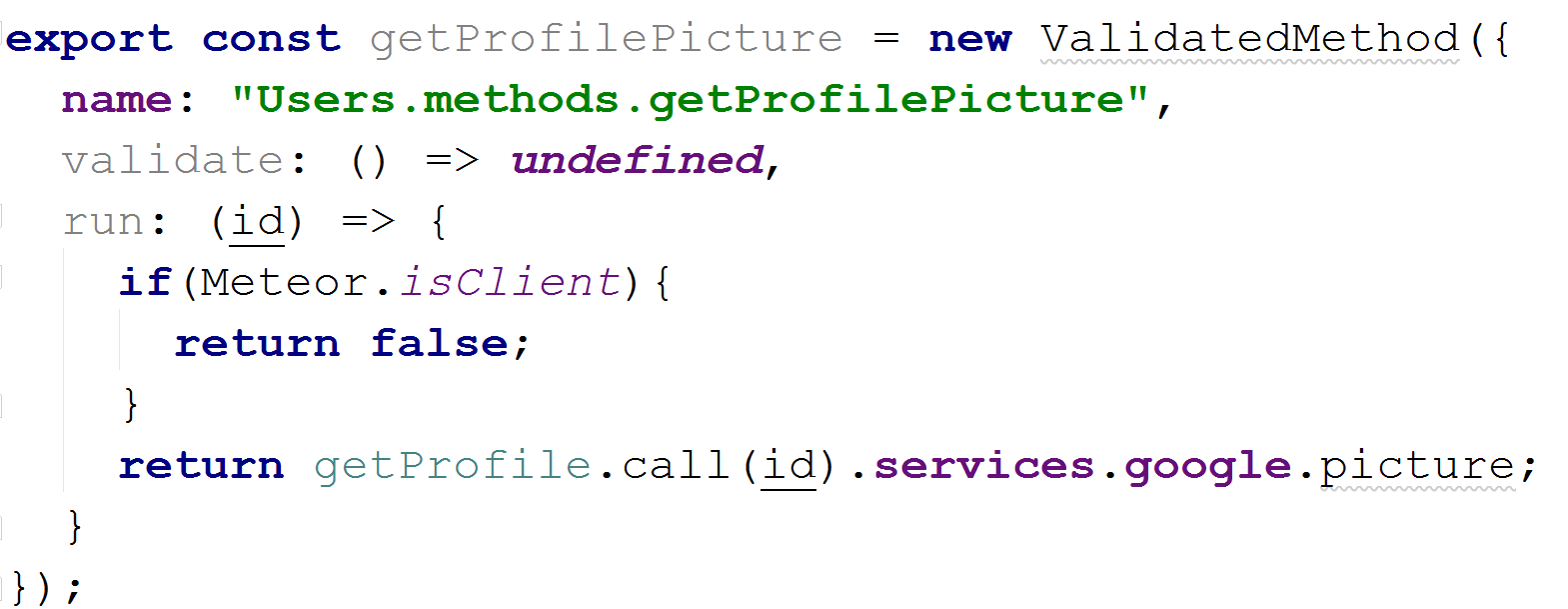
## Artefact 3

*Type: Backend Development*

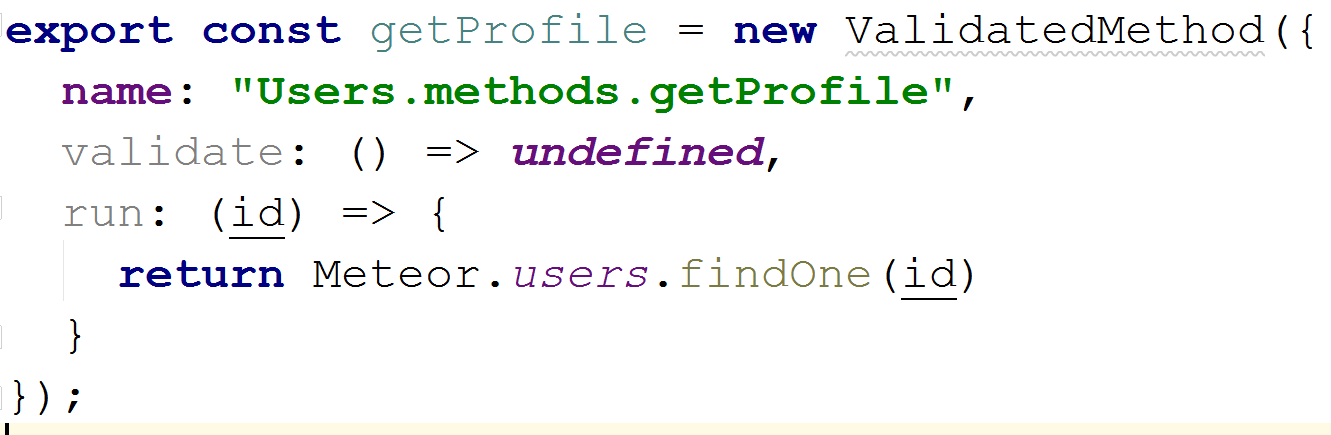
Meteor is a unique framework that tries to bridge the gap between backend, frontend, and database. To execute complicated and risky operations on the database, it’s best to perform them on the server in special functions called “Meteor Methods”. You can think of these methods as API endpoints for Meteor.

I wrote a number of such methods to help with the current user. These methods can be found in /imports/api/user/methods.js

Here’s an example of one such method, which gets the public url of the Google user’s profile picture from the user database:



You’ll notice this method actually calls another method, “getProfile”, which I also wrote:



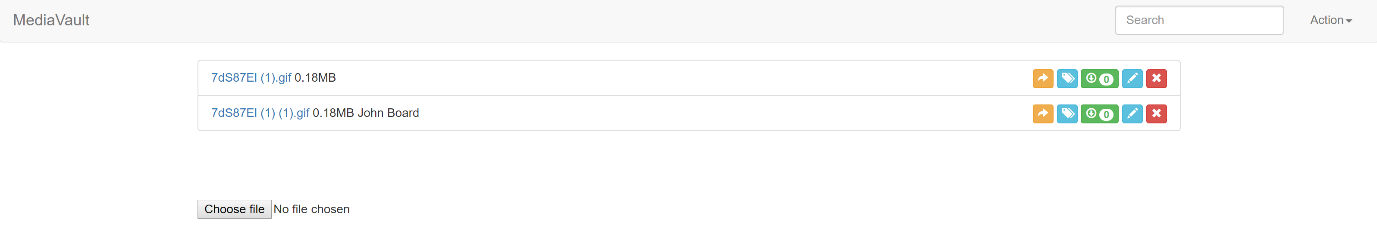
In this method, we check the user “collection” (table in SQL terms) for a document (row) with the \_id (primary key) of “id”. It then returns the document (row) / entry.

These methods are used in /imports/client/ui/pages/home/home.js/html for getting the profile picture which is displayed in the side bar.

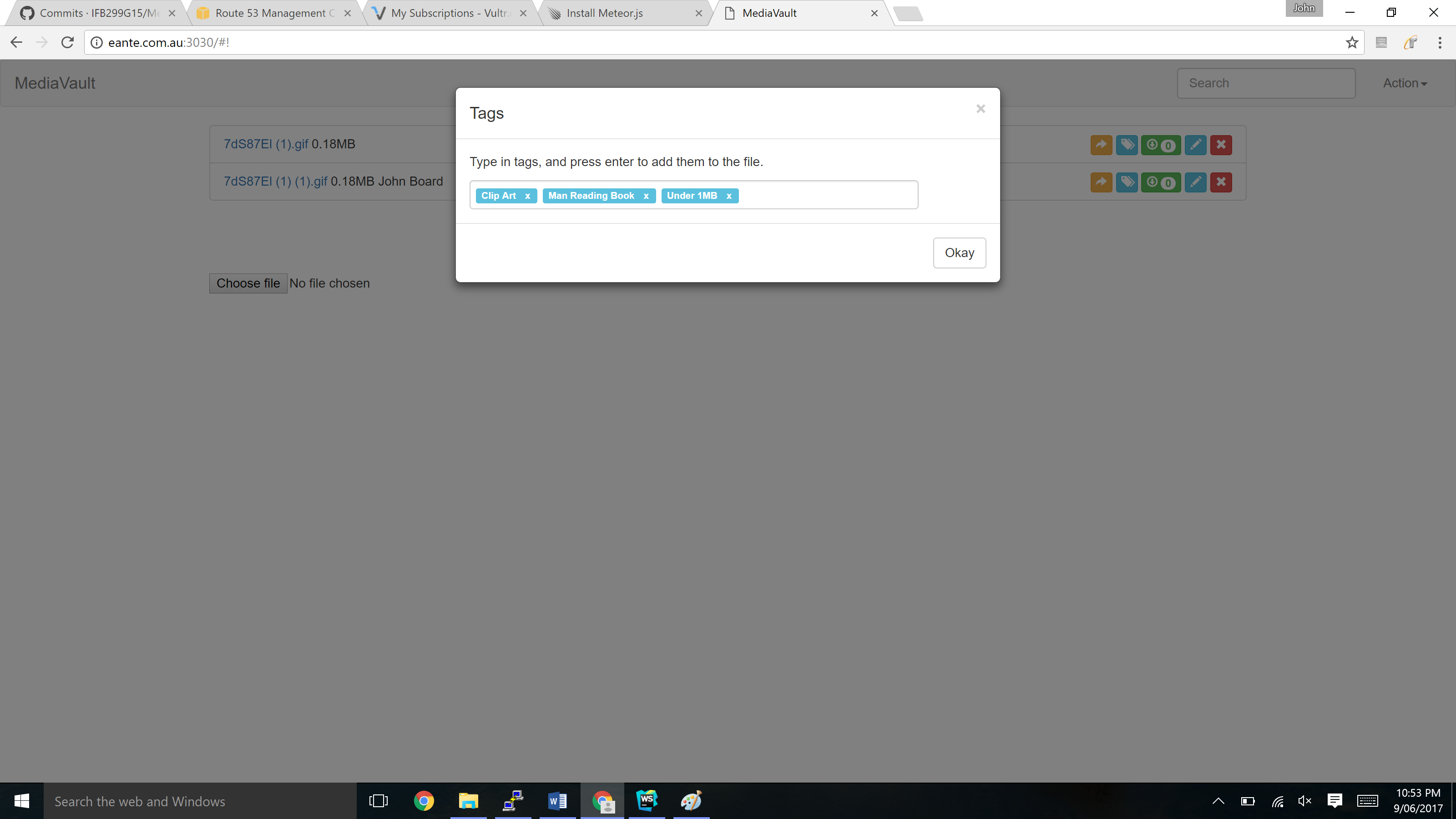
## Artefact 4

*Type: Frontend Design and Development*

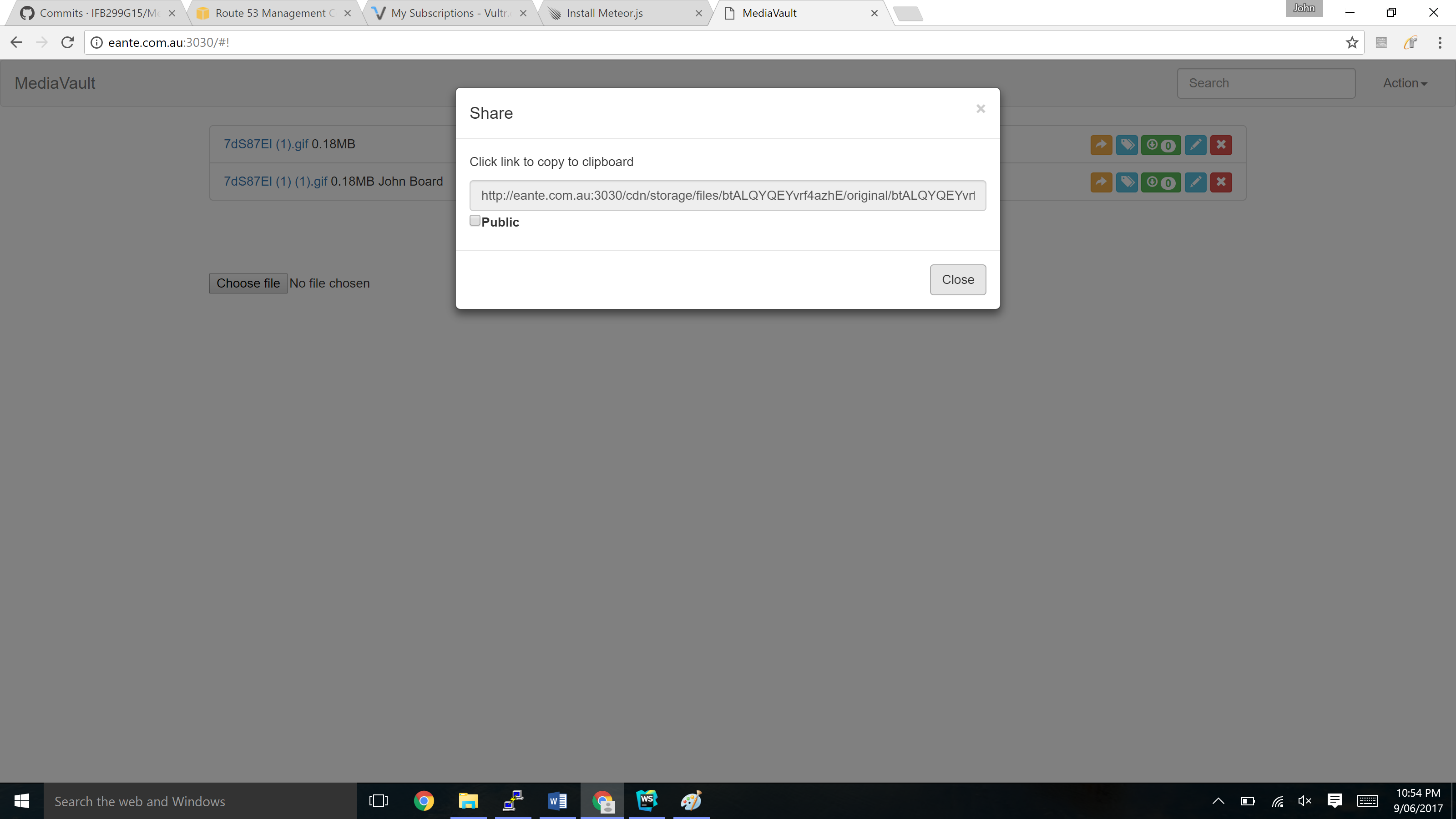
In sprint 1, the initial GUI framework was Twitter’s Bootstrap (which we changed in sprint 2). I designed a majority of what you see.



A list of files, with a number of action buttons. While this worked, it was a bit clunky.



This is a demo of the modals, for example the tags modal, and the share modal:



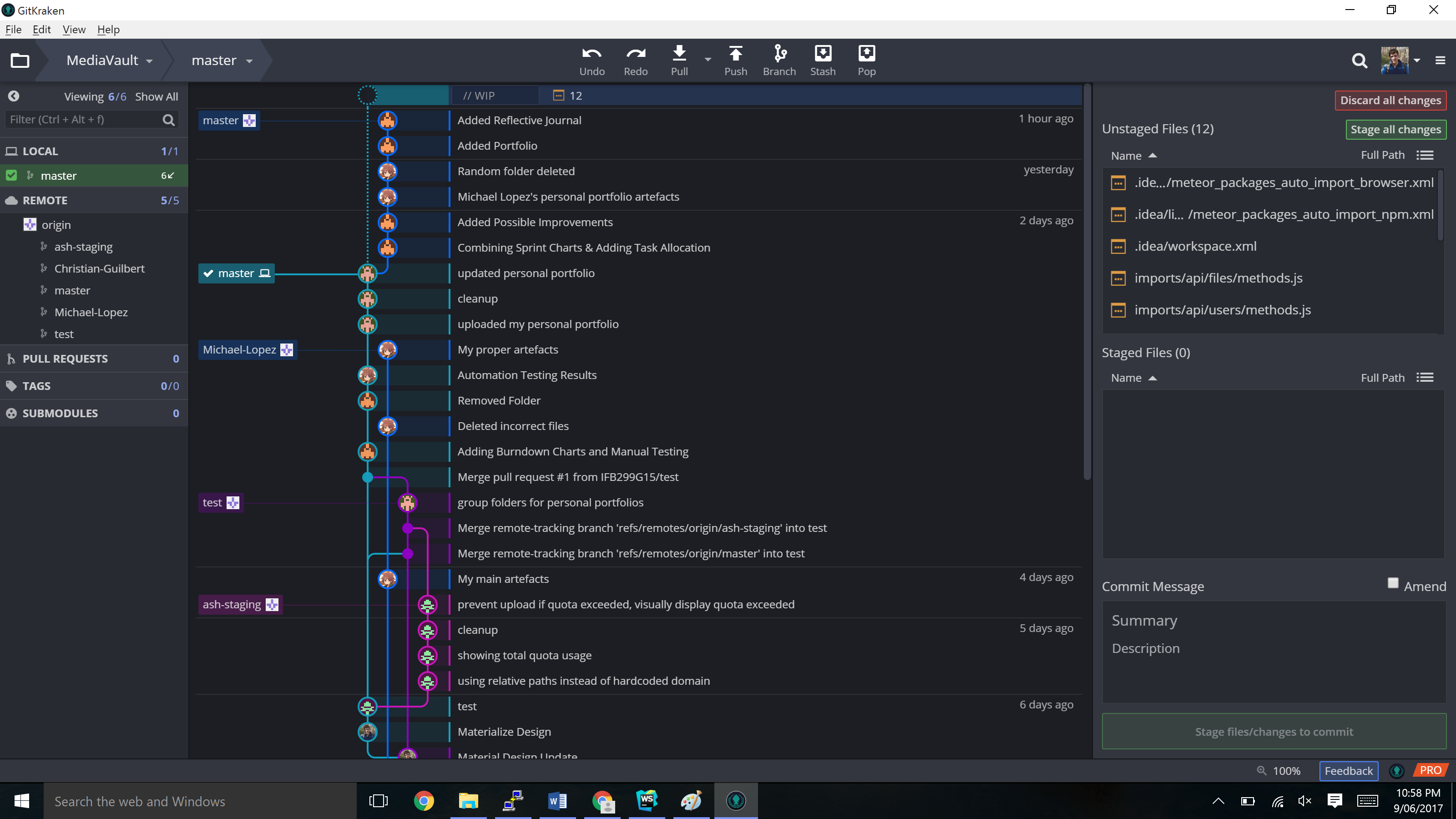
# Sprint 2

## Artefact 1

*Type: Server Administration and Development Operations*

While the server was helpful for running personally, other group members couldn’t effectively benefit from it. To combat this I changed the password and shared it with the rest of them.

While we were developing, we found a need to separate our “development”/”staging” code from our “production” code. I worked with Ashley to create a system whereby we would create a development/staging branch where each person did their work, and then we’d merge those changes back to the master. This was toward the end of our time, so we didn’t get a chance to fully implement the system. Here is an example of the branches at play:



I then integrated this with the server so that after making changes locally, members could upload their latest code by logging on to the server and pulling their changes from git. The server would automagically reload, and they could see their changes instantly on the internet. This is as close as we got to “continuous integration”.

## Artefact 2

*Type: Backend Development*

We felt that it was important that uploaded files have previews, that the user knows which file contains what. Similar to Google Drive.

For this, I found a node package called filepreview which can generate file previews of many different file formats by using linux command line tools such as ffmpeg, imagemagick, etc.

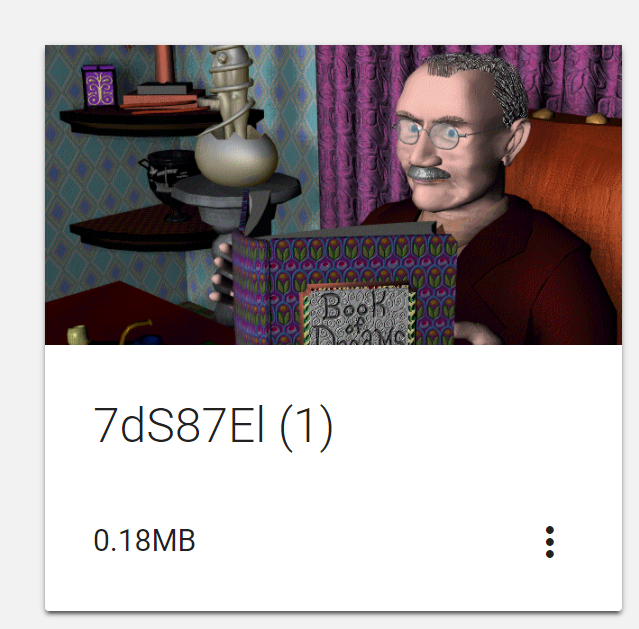
<https://www.npmjs.com/package/filepreview>



Although somewhat clunky in code, I wrote a backend method that would run every time a file was uploaded, and it’d generate a small png preview of the file uploaded, which would be then displayed in the browser.

This code worked quite well. It would generate excellent previews for image and movie files, although it had a bit of trouble with word documents and PDFS. With a bit of debugging, I’m sure I could fix this.

The file previews can be seen here:



## Artefact 3

*Type: Frontend*

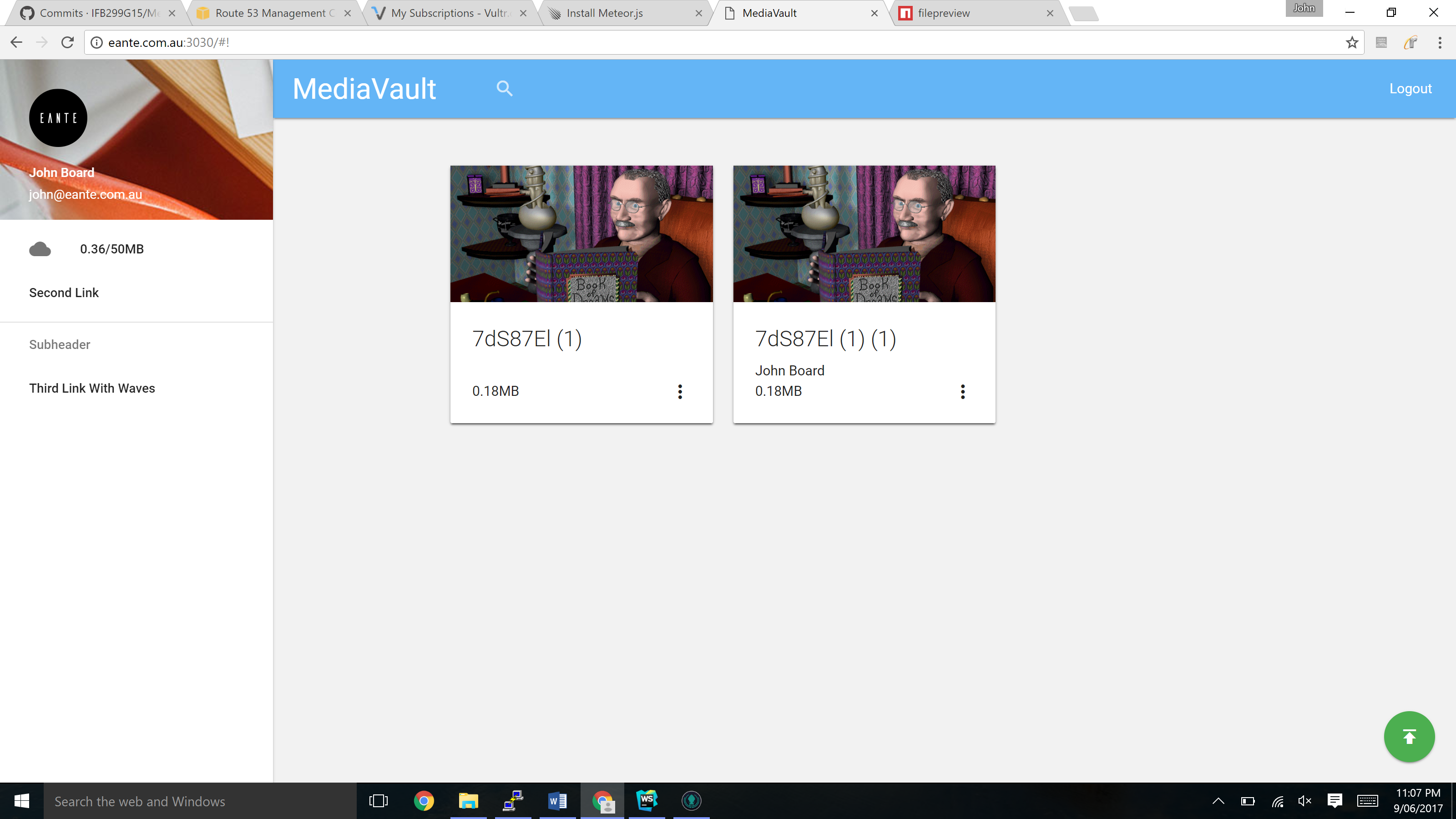
One of the biggest criticisms of the product with the first UI layout using bootstrap was that it felt all a bit clunky and compressed. With the help of the team, I decided to convert everything to Material design – Google’s Design standard.

<https://material.io/guidelines/>

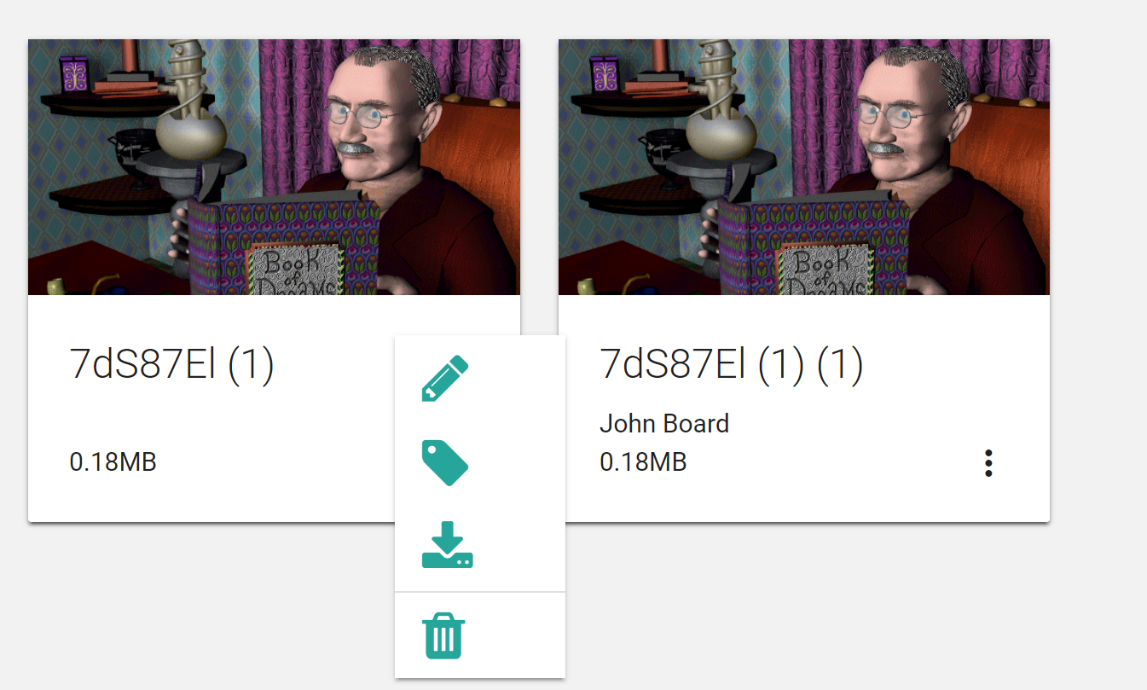
I used Materialize.

<http://materializecss.com/>

The end result looks like this:



I tried to take design ideas from google drive by representing files as “cards”. By interacting with each card by left clicking or right clicking, you could open up menus which allowed you to do further things to each file – instead of having a huge chunk of buttons all there already.



It took me a great deal of time to figure out how to get “right click custom menus” working. In the end the code looked like this:



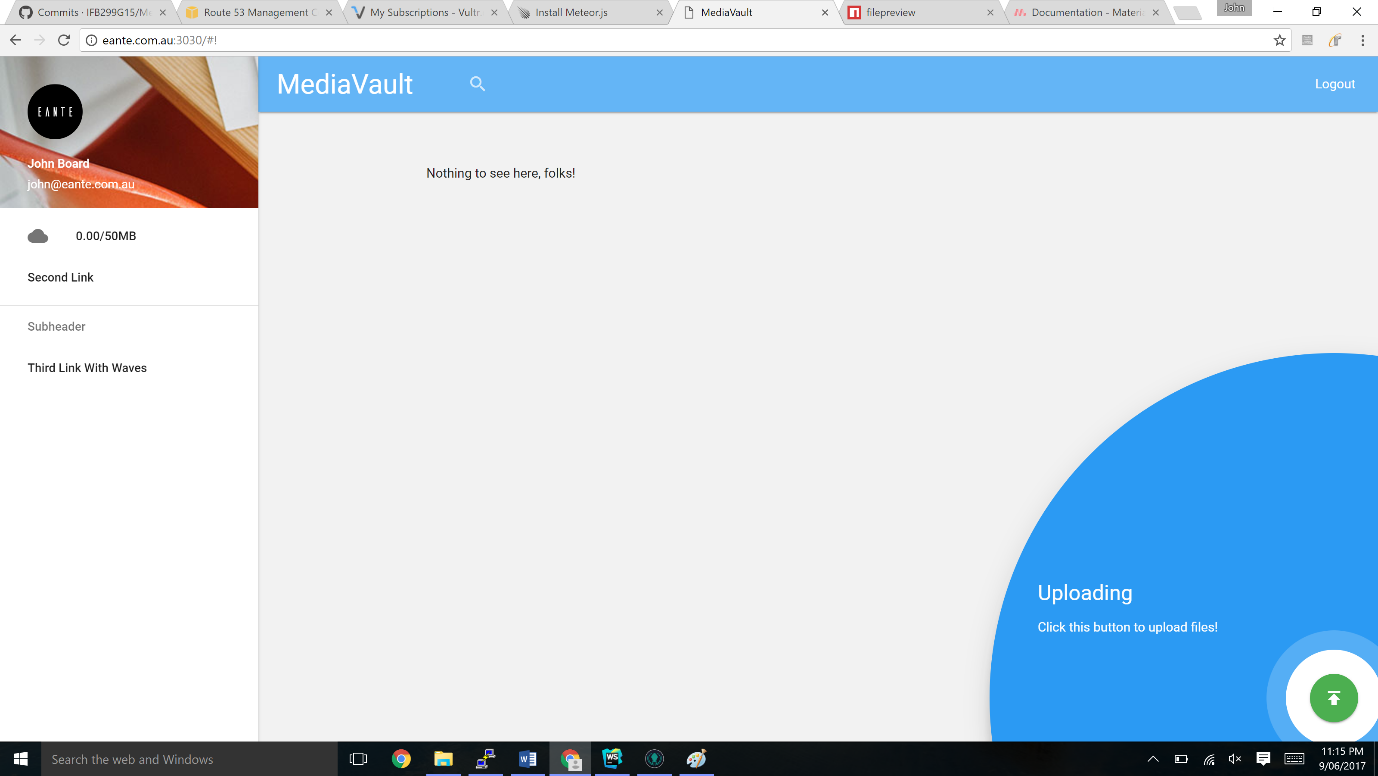
Upon right click, or left click on a specific button, it’d call the dropdown to “drop down”. It’d then modify the css to open at the mouse’s location.

This code can be found in /imports/client/ui/components/fileItem/fileItem.js

## Artefact 4

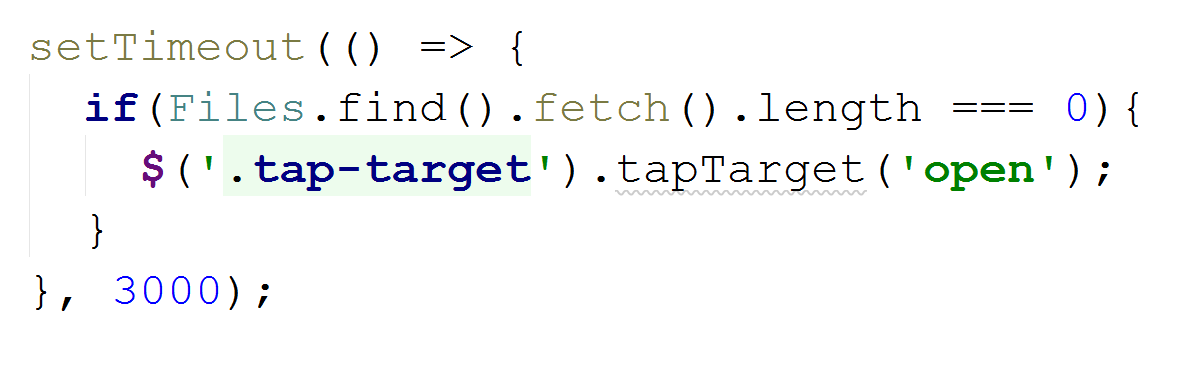
*Type: Frontend*

Again, as part of client feedback, I decided to make the interface more user friendly by adding in a bit of a “tutorial / helper”. When you open up the page, and you have no uploads in your folder, a helpful little animated circle appears, pulsating around the upload button, telling you to click that to upload files:

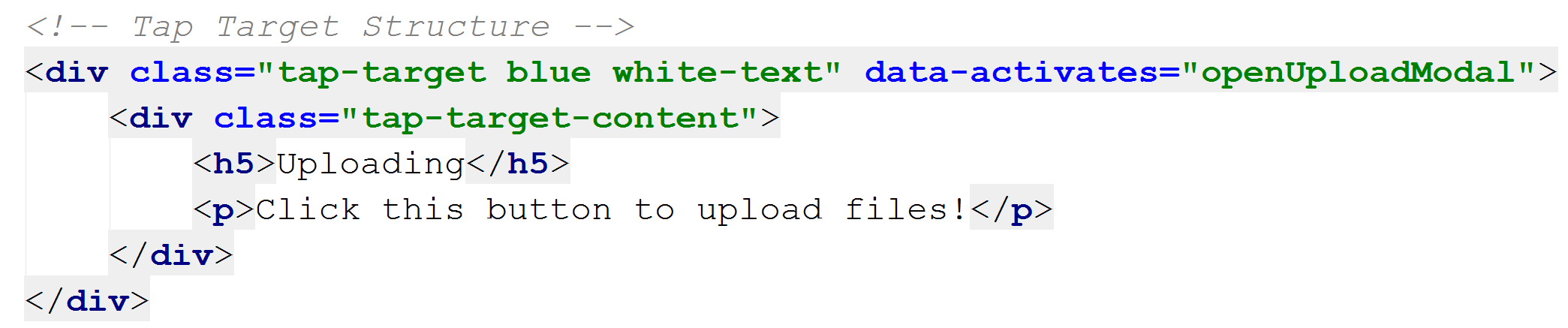


User Interface is not one of my areas of expertise, so it was great to have a bit of a play around with Materializes’s functionality to get a feel for what I could achieve, and how it’d impact the user’s experience.

The JS code to open it is in /imports/client/ui/pages/home/home.js



The HTML is in /imports/client/ui/pages/home/home.html



# Conclusion

This is by no means a comprehensive review of everything I’ve done on the project, but hopefully this portfolio of carefully selected contributions will give you a good overview of the different things I contributed to this project, drawing upon past experience from uni and work.