CS3099 Final Submission

Nebula

[redacted], [redacted], [redacted], [redacted]

Group A12 Juniour Honours Project Final Report Submission



University of
St Andrews

Computer Science
University Of St Andrews
Scotland
April 2021

1 Abstract

The task was to create a platform for distributed online communities which is federated and for the target of a university environment. We have achieved this through the form of a website and have added our own theme, that being around sci-fi. Additionally we were tasked to follow Agile processes and Scrum to work as a team and develop our project. We also was part of a supergroup, to which we would work with other groups to ensure federation was achieved. This report introduces our initial aims and objectives, what has been achieved and how we worked together as a team (including how we followed Agile processes and Scrum) to successfully create our project.

A live version of our site can be found hosted at https://nebula0.herokuapp.com/

2 Declaration

We declare that the material submitted for assessment is our own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 11582 words long, including project specification and plan and discluding the Appendices.

In submitting this project report to the University of St Andrews, we give permission for it to be made available for use in accordance with the regulations of the University Library. We also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis.

We retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1 Abstract **1**

2 Declaration **1**

3 Int **3**

4 Pr **4**

4.1 . . . 4
. . . 4
. . . 4
. . . 5
. . . 6
. . . 7

4.2 . . . 8

4.3 . . . 11
. . . 11

4.4 . . . 12

4.5 . . . 12

5 Ag **16**

6 Ev **17**

6.1 . . . 17

6.2 might be considered missing - . . . 19
. . . 19
. . . 20

6.3 . . . 21

6.4 . . . 21

6.5 . . . 22
. . . 22
. . . 22
d Comparison with our System . . . 23

7 Co **25**

7.1 . . . 25

7.2 . . . 25

7.3 . . . 26

7.4 . . . 27

7.5 . . . 27

8 Bi **28**

9 Ap **29**

9.1 . . . 29
9.1.1 Meeting held on 22/10/2020 . . . 29
9.1.2 Meeting held on 06/11/2020 . . . 30
9.1.3 Meeting held on 06/01/2021 . . . 31
9.1.4 Meeting held on 24/01/2021 . . . 32
9.1.5 Meeting held on 08/02/2021 . . . 33
9.1.6 Meeting held on 20/02/2021 . . . 34
9.1.7 Meeting held on 21/03/2021 . . . 35

9.2 Weekly Progress Reports . . . 36
9.2.1 Semester 1 . . . 36
9.2.2 Semester 2 . . . 37

9.3 Additional Figures . . . 39

3 Introduction - [redacted]

As a global pandemic has swept the globe and university life becoming more insular, the motivation behind the Junior Honours project brief for this academic year was to create a platform (via any medium, such as a mobile application or desktop software) targeted at university students, which allowed for the facilitation of distributed online communities. The specifics of how our system would operate, what it would be able to facilitate, as well as its structure and general theme, would be down to us as a team. The details of what was expected as well as a specification was provided but to a purposely loose degree, this is to allow us to be creative, create a system which we would need to build from the foundations upwards, and made it completely down to us on what we wish to build, in relation to creating a platform for distributed online communities which was federated and targeted to a university environment.

To date, as a team, we believe that we have been successful in creating a solution that successfully achieved all of the above. Considering our initial user stories and what we considered to be minimum requirements, desirable features, and major extensions at the beginning of this academic year, we have come a long way by meeting all our minimum requirements and implementing a number of additional features, of high quality. Furthermore, a further requirement of our system was to design, develop and implement a supergroup protocol in collaboration with all other groups in our supergroup in order to allow for federation between servers. This would let users local to one group's server interact and share content with users on any other servers that the local server federates with.

The combination of the initial (largely loose specification), with our requirements we created as a team and the ever-evolving supergroup protocol over the academic year, gave an outline plan of what our system would look like. The general structure would be communities, where users would join or create communities about topics of interest (such as movies or sports) or a common entity (such as an academic family or enrolled onto a module) and various forms of media would be shared among members of such groups. Building on this structure, as a team we have designed, and implemented features, keeping all present constraints in mind. Over the academic year we have also had to make a number of decisions, these include; design decisions, technologies used and how we decided to work as a team (in relation to Agile methodologies) just to name a few, and these have also been outlined in the report (more specifically in the Project details section).

Further to designing and implementing the system, we have also implemented a testing regime for our system, such that for each feature we create (as well as for when evaluating the system as a whole), we can mechanically evaluate it, find bugs and see how our system responds to normal use cases, certain edge cases and put under general stress. For most features on the backend, we have created several tests (as well as checks as part of the continuous integration (CI) runner, before any branch is merged into the main branch). Furthermore, we set up a number of user tests (conducted by a human, which tests for normal, edge cases) as well as getting individuals external to the team, to critically evaluate it, find bugs and give us an unbiased opinion and verdict on parts to our system.

In terms of evaluating our system, we are genuinely proud of what we have produced and feel like we made great progress on the system during the academic year and team members have learned a lot from the process. However, we also realise no project is ever "fully" completed and can no longer be extended to or improved, and we have critically evaluated our current system against what is publicly available as well as considering what we would like to add to our system given more time. This is outlined in the evaluation and conclusion sections of this report. We also realise as a team, in hindsight and seeing how things have worked out over the academic year, we may wish to have done certain things differently, and again we have critically evaluated this below.

4 Project Details - [redacted]

4.1 Frontend: Design Decisions - [redacted]

4.1.1 Main Goals - [redacted]

When designing and implementing any UI the user experience should always be at the forefront of the mind. Is this feature easy to access? Will implementing this feature enhance user experience, or will it hinder? Are features intuitive and natural to the user? These are all important questions to think about during the design and implementation of a successful UI. In our case, early on we decided as a group to place a focus on creating a fast and dynamic, yet unique and user focused frontend. Without careful design and implementation often one of these comes with the sacrifice of the other. Our goal was to avoid this.

We can separate out the frontend design decisions into two main sections:

- Server Communication and the "bakcend of the fronted"
  - Code running within the web browser which retrieves data and generates content for the user.
- UI / UX
  - The appearance of the site. Decisions taking into account details such as accessibility, "eye-catchiness", usability, ergonomics and more.

Another goal of ours was to provide transparency to the user. In our view this meant focusing on elegant error handling (providing response information to the user) and a fail safe UI. This meant focusing a lot of our efforts on ironing out bugs and integrating safeguarded requests to prevent the user experience from being interrupted (to an extent) due to possible communication issues with the backend.

4.1.2 UI/UX - [redacted]

As would be expected, we wanted to prioritise 'function' over 'form' with the design of our interface. That said, we realised that there were still many opportunities for us to construct our front end in such a way that it would not only be accessible and easy to use, but also aesthetically interesting and pleasing. Normally, the Average Time on Page for web users tends to be around one minute (CITE klipfolio); however, on our site users could potentially spend much longer than that if they are reading posts, or chatting with others. This meant that we had to make sure that our interface would not be visually exhausting to look at for extended periods of time. Popular social media sites tend to have a more understated, almost minimalist aesthetic. There could be multiple reasons for this - clearly opting for simple designs would likely maximise ease of use, but it is also worth noting that a simple website design allows for user-created content (that is, posts, comments, profile pages and such) to become the focal points. With our design, we decided that this was a good trend to follow, but we attempted to strike a balance between following this trend and emphasising the light-hearted science fiction theme of our product (which, for instance, can be seen in our use of names such as signals and clusters).

The following three images (1 2 3) show the progression of our platform's logo from an initial idea to the final design. We believe that the calligraphy-inspired font in the third image, along with the shooting star markings embody our website's theme.

We decided on our science fiction theme in a group meeting, and proceeded to brainstorm ideas for our platform's name - we initially thought of names inspired by the various space missions attempted in recent decades, but we eventually settled on the name 'Nebula'. A nebula is defined as 'a giant cloud of gas or dust in space' (cite nasa). However, more than that, we believed that the word 'nebula' tends to remind most people of the visually spectacular images that have been taken of them - for example, the image of the Orion Nebula taken by the Hubble Space Telescope in 2006. In other words, we believed that the word 'nebula' was more often than not associated with words such as 'large', 'spectacular', or perhaps even 'inspiring'. This led us to the conclusion that the name 'Nebula' would work very well with the goals and theme of our platform.

Since accessibility and ease of use were important points for us throughout the design of our front end, we took measures including (but not limited to) ensuring that the text on our website was reasonably large and



Figure 1: Logo Version 1



Figure 2: Logo Version 2



Figure 3: Logo Version 3

coloured such that it did not blend with the backgrounds, and using icons to convey interface information to users (doing this makes the entire front end appear less 'heavy', visually).

4.1.3 Internal API - [redacted]

At the end of the day, the frontend's role was to provide a visual interface to the server, retrieving and sending data. In the case of our project, this data was in the form of posts, communities, users and other related information, displayed in various ways. The server is a REST API answering HTTP requests. As discussed in detail in the backend section our backend is separated into two API's; internal and external. This gave us more flexibility on the information we chose to send to the frontend, meaning we could extend the federation specification to include information to make our site more dynamic and unique.

Communication to the server on the frontend is done through using 'axios' (more specifically 'VueAxios')

a "Promised based HTTP client for the browser and NodeJS"(1). This allows us to integrate asynchronous HTTP requests to the backend to dynamically load and generate content for the user. Axios is feature rich (allowing modification and addition of headers and sending of different content types) and provides an easy to use API, making response and error handling smooth and information heavy - aiding in that goal of transparency to the user.

With scalability in mind, an initial goal of ours was to limit the number of HTTP requests sent to the backend, with the goal of transferring much of the computational work to the user. This goal was maintained to an extent, however we quickly realised that this tradeoff would more than likely compromise the user experience. Our choice to use the Actix framework on the backend meant that our server was extremely durable to high traffic, and therefore HTTP requests from the frontend are extremely cheap. Due to these factors our frontend compromises limiting HTTP requests where it is feasible and computationally inexpensive to process the data at the client side, but to heavily place our focus on user experience and utilise the speed of the backend where possible.

There is a great example of this in the retrieval of post comments. Our thought process when solving this problem demonstrates our aims when creating the internal API, and similar problems can be found across the entire site. Since we are using a separate protocol to the federation, it was our decision how we wanted to send out comment information in posts. These were our main options:

- Send a post with recursively embedded comments - each comment containing all information (title, author, body etc.)
- Send a post with only top level comment IDs

Since comments are treated as a post (we decided to maintain this in the internal API). The structure of children is recursive and would match whatever we picked. Our end solution was a hybrid of these two methods. We decided that the first implementation would be too bulky; much of the information sent would be initially redundant (due to homepage posts not displaying comments by default) thus resulting in unnecessary overhead on the backend and wasted memory on the frontend. The second solution solved these problems but introduced a new problem; for each comment we would need to send an HTTP request for each of the children, and so on for each of their children etc. With scalability in mind the overhead of retrieving thousands of comments and each of their descendants seemed excessive. Additionally, we knew that pressing the "see comments" button would display all comments. Therefore our hybrid solution comes in - when retrieving a comment, the first solution is implemented, meaning all comments (and their full information) are nested. Hence, our internal API follows this protocol;

- When requesting a post with a null parent, the children field will contain an array of the IDs of the children
- When requesting a post with a non-null parent, the children field will contain an array of the child post objects and their descendents recursively nested.

As mentioned a similar thought process can be followed when analysing other internal endpoints and requests sent by the frontend.

4.1.4 Efficiency - [redacted]

Efficiency can often be a compromise on the frontend when creating a rich, dynamic site. That said, measures can be taken to limit computationally expensive operations. The first concern for efficiency I have outlined above; HTTP requests. Computationally, these are extremely cheap to construct and send. However the user experience can be affected if a great deal of time is spent waiting for the response of multiple requests. As aforementioned we can negate this by somewhat limiting the number of HTTP requests, but as described we want to focus on user experience and some requests are inevitable. Therefore the asynchronicity of Javascript and Axios comes to the rescue. This allows us to render information as it comes in - the alternative being having to wait for all the requests to finish before displaying content. This way the user always has some information on the screen even whilst waiting for larger responses to be received and rendered. Take for example the homepage. The list of posts is the largest section of the page and also the largest data request possibly with thousands of post objects being received and rendered. Whilst this loads other information

is already readily available; the profile and community section on the right, the App bar at the top, and the messages section. The asynchronicity of the pages limits "down time" on the site, enhancing the user experience.

The next concern for efficiency comes from rendering large amounts of dynamic information. We utilise VueJs's List Rendering (2) very frequently in our site to render dynamically sized lists of content. With scalability in mind, attempting to render thousands of posts on one page will cause obvious performance issues. As is often the case here the simplest solution is the best. We simply limit the number of posts rendered at a time (and the same for comments). If the user wishes to load more we provide that option. The benefits of this are obvious, and by ensuring via various metrics that the posts they see are the most relevant to them, we limit the number of posts they see on a given page to a low, easily renderable (and hence fast) number.

Luckily for us VueJS already has lots of built in measures for optimizing rendering. A concern when rendering a component with many child components is that a change of state in said component will result in a cascading re-render of all the child components. This is often not at all necessary. As described on the Vue website page "Comparison with other frameworks" (3), reducing the unnecessary renders in child components in other frameworks such as React requires careful consideration and utilisation of methods such as "shouldComponentUpdate". In VueJS "a component's dependencies are automatically tracked during re-render". An example of this coming into play in our system is in our post objects. These could contain thousands of child comments as nested components (although as described above we have limited this, but for example's sake), therefore a change of state of the post which has no bearing on the comment child components, such as the post title, would force a re-render of all the comments in React if it was not implemented carefully. In VueJS however this dependency is tracked, and therefore since the affected attribute does not concern the comments or any other child components, the child components are not re-rendered/affected.

VueJS also tries to optimise re-rendering on state changes within Vue List rendering. To avoid re-rendering the entire list on change to any one component we need a method of tracking each unique item in the list. Hence the use of a "key" when defining a list render. As described on their List rendering page (2) VueJS "patches" each item in place in the array, rather than changing the order (particularly useful for us as we do post sorting). This in place editing of list items is much more efficient because each item can be directly indexed via their index in the list (determined by the object key). Therefore editing the order of the list becomes an O(1) operation rather than O(n) should we have to shuffle items through the array. This operation may be more expensive if we are affecting the entire list, for example reversing the order, however since the majority of our operations on the array are affecting only a limited number of objects at a time this method is much more efficient. In order to have each element "reflect what should be rendered at each index" we provide the aforementioned key to allow vue to track each list item's identity. Additionally, certain array methods (such as push and pop) are integrated and supported such that the list render is mutated only as necessary, rather than affecting the entire list. All of the above aids in the site's performance and efficiency.

4.1.5 Expandability - [redacted]

Expandability is less of a concern for the frontend as it is for the backend but nonetheless something we need to consider. Where on the backend the problem becomes the increase in load and frequency of requests, on the frontend scalability leads to an increase in received information and larger response sizes / data received. These concerns can also be slightly addressed on the backend; being more careful / clever about what data we select to send, rather than leaving all the processing to the frontend. On the frontend these concerns can be addressed through ensuring efficiency, by the means described in the section above.

In terms of expanding the site itself and creating room for additional features in the future, this was mainly allowed for through our development process which is described in more detail in the "Development" section later on. The key idea was to maintain independence and develop the site compartmentally. In VueJS, this meant ensuring that each feature was compartmentalized into at least one, and preferably more, separate components. By doing this if we were to add a new feature we could simply add a new component (or perhaps edit a current one). This meant that the addition of new features could be done independently of other features, keeping the site operational and functional.

4.2 Frontend: Technologies - [redacted]

When deciding on technologies there were a few key factors we considered when choosing a framework. The first decision to make was the choice between client side and server side rendering. Server side rendering offloads the work of generating the DOM content of the web page to a server side application. Client side intialises with a set HTML document, then sends subsequent requests to the server to retrieve information which is then used to populate the site.

How do these differ in user experience? We need to look at the interaction of a webpage by the user in terms of different interactions. In our site here are some examples of types of interaction.

- Initial Page Load (homepage)
- Subsequent Page loads (e.g. profile page)
- Refresh of displayed information (Re-getting posts)
- Sending of information (Creating a post)

On a static page, where information provided by the server is stable and unlikely to change, server side rendering seems to be the preferred solution. This paper by *Mathias Beke* (4) measures an initial average load time of a web application to be about 50% lower for a server side rendered application against a client side application (see figures 4 and 5, figures from (4)).

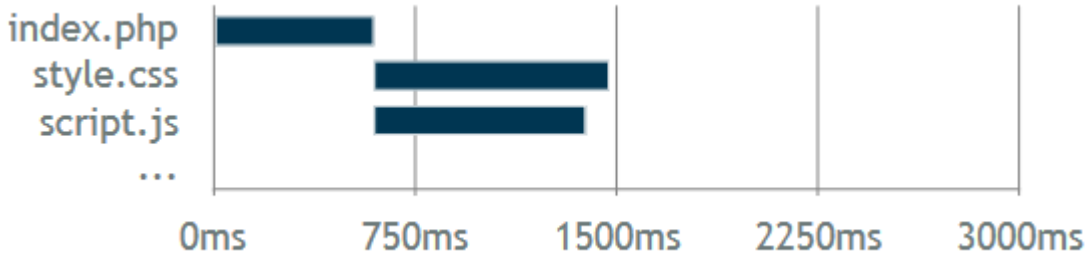


Figure 4: Network waterfall of server-side rendering for initial page load.

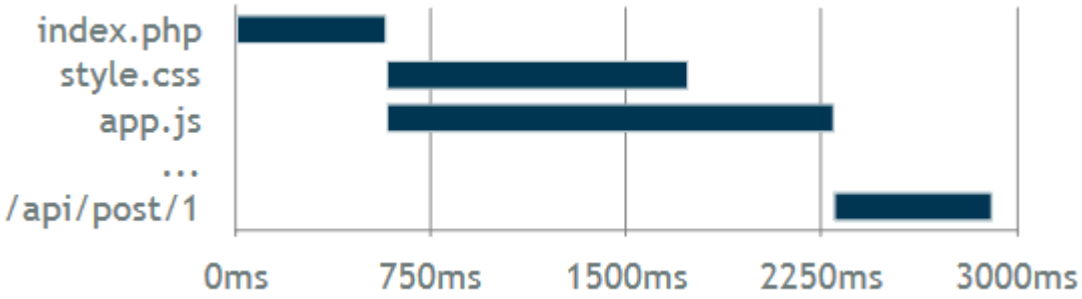


Figure 5: Network waterfall of client-side rendering for initial page load.

On client side rendering that initial load is delayed by the load of a much larger initial DOM and accompanying client side code which will be used to generate the future content of the site. The lack of dynamic local content on the server side generated client application means comparatively there is little code to load and run - all this work is done on the server. This is great, however it comes with some drawbacks. Since this site is now locally static it means the site must be compartmentalised into separate

HTML "pages". For example on our site if a user was to navigate to the profile page the server would have to generate a whole new HTML document to send back to the user (containing the relative information about said user). However in our implementation (client side) this HTML is already loaded client side, so only an HTTP request is required to load the new information - resulting in much smaller load on the server and faster client side updating. As well as this, this re-render of the entire page's HTML is required whenever any information is updated. For example on our site if we are to make a new post and post it to the server, in order to keep everything up to date locally and display that new post, the entire DOM of that page must be re-rendered server side and sent back to the user. Whereas client side we can simply update the DOM with that new post.

With this we can begin to see how with a site as dynamic as a social media site, client side rendering overall poses a much more efficient (and easier to develop) solution.

With this said, we now had to choose a framework for frontend development. We very briefly considered vanilla JS, and although it would be fast we knew it would be much harder to develop in than if we were to use a dedicated framework. For us the choice immediately came to between VueJS and React. One member of our group did have experience with React, however since either way the majority of the group would be learning a new technology this had little bearing on the final decision.

Both React and VueJS utilise a virtual DOM - essentially a copy of the actual DOM which is analysed to see which components need updating - much faster than trying to do so on the regular DOM. For standard operations, this table (5) gives fascinating insight into some of the benchmarks for different frameworks respective speeds. We can narrow down the results to compare VueJS and ReactJS (See Figure 6).

Operation	VueJS time (ms)	ReactJS time (ms)
Create rows	182 ± 8	181 ± 7
Replace rows	159 ± 3	157 ± 2
Partial Update	156 ± 10	82 ± 3
Select row	11 ± 2	10 ± 2
Swap Row	20 ± 3	107 ± 2
Remove row	54 ± 2	50 ± 1
Create many rows	1603 ± 35	1935 ± 33
Append rows to large table	343 ± 6	269 ± 7
Clear Rows	192 ± 6	175 ± 4

Figure 6: Table Comparing basic list operation computation times on the VueJS framework and the ReactJS framework. All these operations are done on tables with 1000+ rows - details can be seen at source (5)

Looking at these metrics are particularly useful as these "row operations" are very common place in our application. We can think of the list of comments / posts being our "table" in this respect, so any list rendering in our application will relate to the benchmarks shown above.

We can see that for the majority of operations the benchmarks are incredibly similar. The two most noticeably different rows are the "Swap row" and "Partial Update" operations. We have actually explained the difference in speed for the "Swap row" operation above in the "efficiency" section where Vue utilises in place operations to "patch" rows to increase the operations efficiency. This results in a decrease in operations time of about 80% (with 1k items in the list), particularly useful for us as we are often doing list alterations when we sort posts, or insert a new post. The partial update operation is described as the "Time to update the text of every 10th row ... for a table with 10k rows". This is about half as fast in Vue as in React, and although I could not think of a use case for this, it may have some effect on our application.

Both frameworks "encapsulate" separate components very well - which was very important for the way we wanted to develop our application. As well as making debugging and addition of new features much easier. This encapsulation also made it very easy to reuse components, particularly important on a social media site where there will be lots of repeating objects such as posts or comments. Both frameworks did this very well.

A metric very worth considering especially in a project such as this is the learning curve of the framework. In this project, with multiple modules and other commitments happening in parallel, the chosen framework had to have a small learning curve and be easy to pick up. Even more important in our group since we were a smaller group of only 4 rather than 5 people, and 3 out of our 4 members had close to zero experience with frontend development. Unfortunately "learning curve" is not really a quantifiable metric, however there were a few ways we could analyse this:

- Documentation
  - Both the quality and depth of it.
- Community Support

Both React and Vue are extremely strong in these regards, both having very well written documentation and huge community backing (Vue JS backed more by the community and React backed more by Facebook, its parent company). This paper (6) looks at some of the community statistics as well as providing information about the "Github stars" (figure 7) each respective framework has. This indirectly correlates with the community use of said framework and hence will correlate with the amount of community support and advice available online - making them easier to learn.

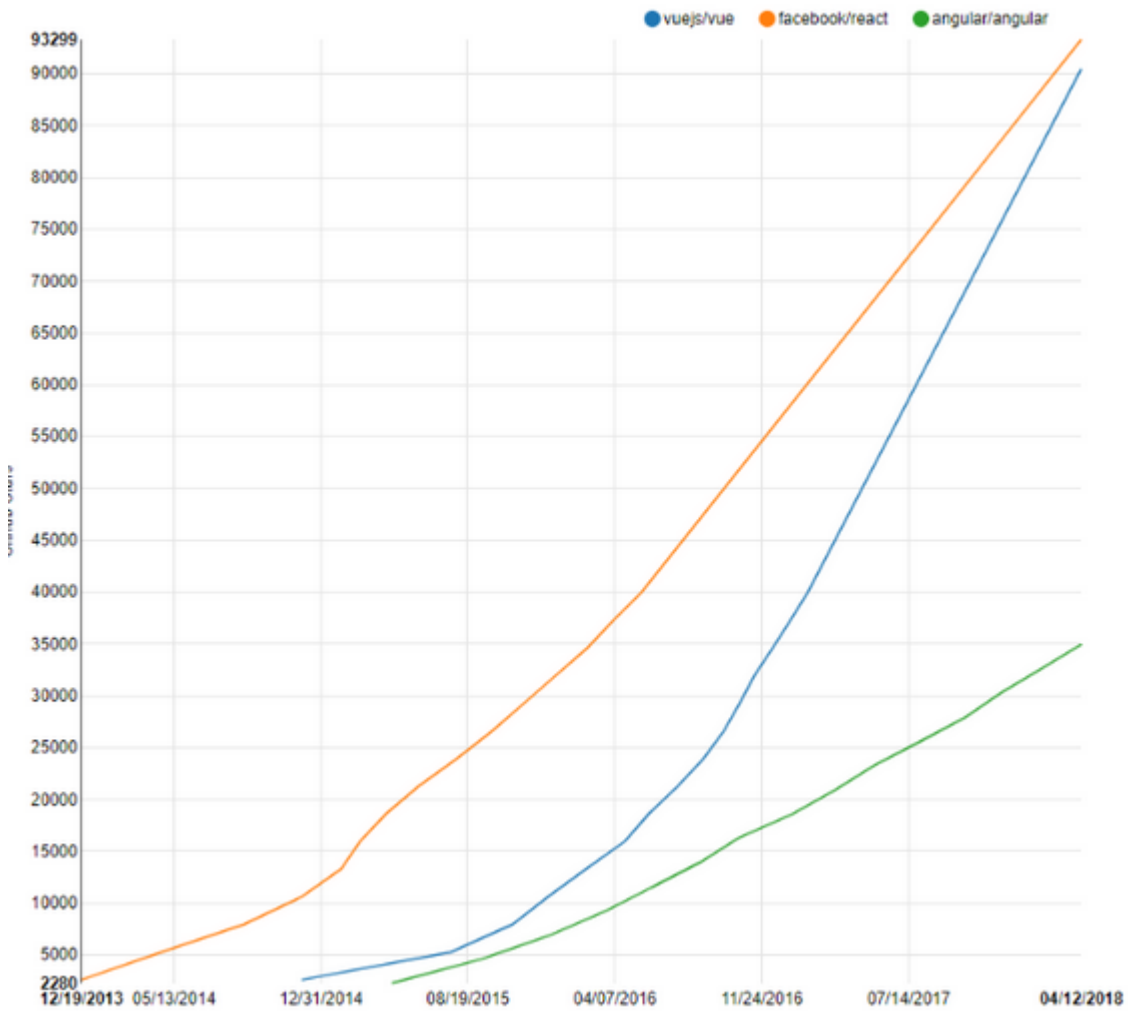


Figure 7: Github Stars over time

We can see how quickly support for VueJS has risen over time. Today (March 2021) the figures stand at 166,000 stars for React and 181,000 stars for VueJS. Meaning support for both frameworks within the community is incredibly high and hence it is easy to find support. The aforementioned paper also notes that there is an increase in the learning curve for ReactJS due to its required use of JSX - a separate HTML - JS syntax. This is optional for Vue, and hence is perhaps one less thing to learn. NPM downloads can also give us an insight into the use of these frameworks over time (figure 8).

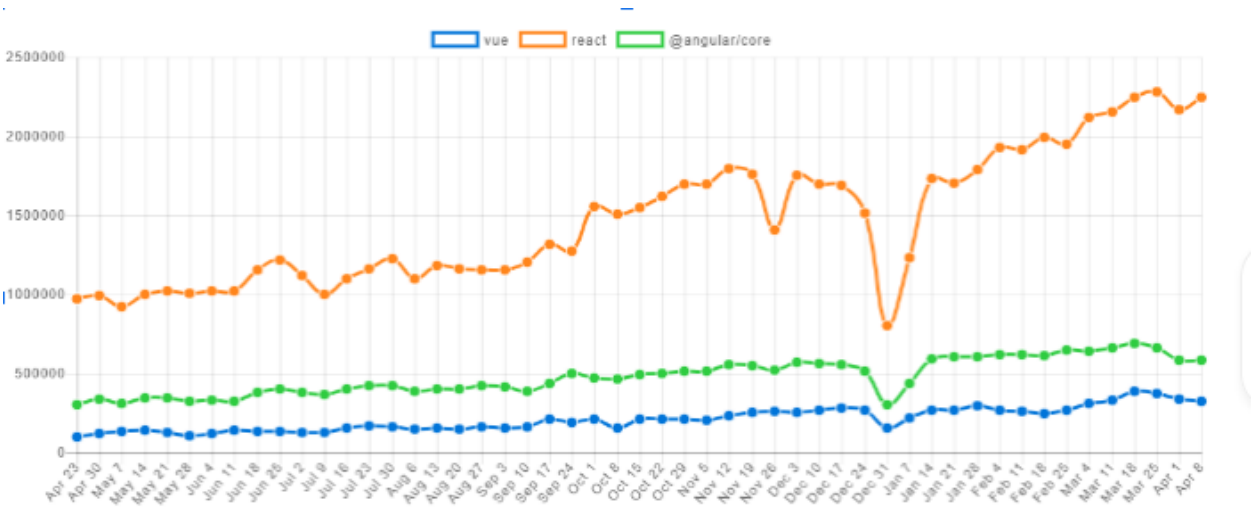


Figure 8: NPM downloads over time

(plus its cool seeing the dips on new years and christmas).

OverallI think although these metrics are interesting, for our purposes either framework would have been suitable for this application. Both are flexible and able to create reactive, interesting websites whilst also having strong community support and solid documentation. At the end it came down to the fact that several of us were interested in learning VueJS and this was the perfect opportunity to do so.

4.3 Frontend: Development - [redacted]

4.3.1 Frontend Development with Agile and Scrum - [redacted]

For the most part when developing a frontend and user interface using agile, practices do not have to change. The incremental approach of agile carries over nicely to frontend development where as aforementioned we can easily compartmentalise the frontend into components which can be independently developed without affecting the rest of the project.

In our scrum meetings user stories were put into the scrum backlog in the same way as would be expected, oftentimes without much adjustment needed from the user story to describe technicalities as designing a UI is a much more high level process.

In order to aid development we each developed different regions of the site, rather than having multiple users working over each other to develop one feature. For example [redacted] did much of the homepage and "post" component developments, whereas [redacted] did work on communities and community pages, and [redacted] worked on instant messaging and the components surrounding that (to list a limited few examples). These each are separate entities, often on entirely separate pages on the site meaning work is independent so the implementation of new features can be completed in parallel, speeding up the development of our website.

This paper (7) details nicely some of the key differences in the agile development process which arise when doing UI design. Where on regular Agile development such as during development of the backend of our site agile methods iterate on code with iterations "lasting weeks", UI comparatively is also an incremental

process rather than a continuous one, and so naturally is incremental in its approach (creating components, buttons, functionality etc.) and as such iterations may last just "hours or days". Additionally, both forms of development require testing, but where regular agile development can be tested through automatic means, it is very difficult to do this for UI, and so testing must be done through manual means or, preferably, through a staging process where users can experience the site.

The paper also mentions how UI design can be fully completed before development. A process known as "Big Design Up Front", but warns of the dangers of it; "the more of the design that is determined up front, the more difficult it is to change later on, when time and experience may make change desirable". In our case we had a mix of features we were certain we would implement (such as posts, communities, comments etc.), but also a number of features we didn't decide to implement until later into the project (ability to post pictures, instant messaging etc.). As such it was useful to do this preliminary design process, for which we used a tool called "Figma" 14 to draw up some preliminary site designs, but to use these as inspiration and a roadmap for what we wanted our site to look like rather than a concrete design. Therefore nothing was set in stone, this allowed us to be flexible with our website design and with the features of the site. This is of course extremely useful in the agile process, where user stories and features may change at any point.

As mentioned, testing was an entirely different process on the frontend compared to the backend. There are tools out there for testing user interfaces automatically, such as "Cypress.io" (8). Given more time this would certainly have been an interesting feature to look into. However unlike unit testing on the backend, frontend feature testing would have been an entirely new skill to build and learn. And, as a team, we decided that this was not something we wanted to do in the time constraints we had. Instead, we went for the regular approach of testing through using the site. Essentially all this involved was emulating a user going through the site and using each feature we implemented. This could of course be done incrementally; testing a feature in this way as it was implemented. Doing this made it very easy and quick to find bugs within our site. To enhance this process and remove some of the natural bias, we made sure to each test each other's features once they were implemented. This avoided the producer of that site feature testing their own feature, which may lead to subconscious bias as to the way they use it resulting in edge cases and locations which may cause bugs being accidentally ignored and not tested. Therefore the best approach was to notify the rest of the team when a feature was implemented and then to communicate feedback to that team member after testing said feature.

An even better improvement on this would be to use a staging environment and release the site to a controlled number of non-involved users who themselves could then provide feedback to the development team. This again alleviates the risk of that developmental bias coming into play. A staging environment / site was created, and given more time it would be interesting to release this site to a limited number of users to gain feedback.

4.4 Backend - Federation Protocol - [redacted]

The federation protocol has a security specification in order to provide authentication between servers, but does not define any form of encryption. This was done as the core specification is only a HTTP API and does not require any specific transport-layer protocol meaning the use of TLS is technically allowed (although there is an issue with respect to port numbers and the hostname textual representation). The security specification requires that requests to servers contain "Digest" and "Signature" headers, containing a SHA-512 hash of the request body and an RSA SHA-512 PKCS#1 signature of a predefined string containing several request parameters respectively. In our implementation this is done using middleware, allowing every single request towards a federation endpoint to be validated before it is received by a route handler. This prevents developers from accidentally not including validation on an endpoint and results in cleaner, organised code.

4.5 Backend - Infrastructure - [redacted]

There are several components to the infrastructure of this project. GitHub is used to host the source code, issues and boards are used to organise work between group members and sprints. This integration was found to be significantly more productive than the GitLab interface and Trello which required group members to manually determine which issues and pull requests correspond to which notes on the board.

GitHub Actions are used for Continuous Integration and Continuous Deployment; unit tests are automatically run for each commit on a branch, branches may only be merged into ‘main‘ if all tests pass and every commit on ‘main‘ is deployed automatically (see figure 9 for testing example). Heroku was used for hosting, and allows for easy deployment from Actions using the ‘dpl‘ Ruby tool. Since the backend serves the static frontend content and we wanted to enable the compile-time checks offered by the ‘SQLx‘ library, deployment is slightly more complex than the default configuration. The frontend is built by Actions first then copied into the backend directory in the CD pipeline which is then sent to Heroku where it is built. Compiling the backend on Heroku means the database URL is available as an environment variable and so can be used by ‘SQLx‘.

```
~/D/n/backend (federation_improvements|●1) $ cargo test
   Compiling backend v0.1.0 (/Users/ferdiamckeogh/Downloads/nebula/backend)
    Finished test [unoptimized + debuginfo] target(s) in 19.87s
     Running target/debug/deps/backend-6c1979e5b81d66b2

running 55 tests
test internal::admins::test::get_admin_fail ... ok
test internal::admins::test::remove_admin_nonexistant_fail ... ok
test internal::admins::test::get_admin_success ... ok
test internal::admins::test::get_nonadmin_success ... ok
test internal::admins::test::add_self_admin_fail ... ok
test internal::admins::test::remove_admin_success ... ok
test internal::admins::test::remove_admin_unauthorized_fail ... ok
test internal::communities::test::add_nonexistant_moderator_fail ... ok
test internal::admins::test::add_admin_success ... ok
test internal::communities::test::get_community_fail ... ok
test internal::communities::test::delete_community_success ... ok
test internal::communities::test::add_moderator_notauthorized_fail ... ok
test internal::admins::test::remove_self_admin_success ... ok
test internal::admins::test::remove_admin_notadmin_success ... ok
test internal::communities::test::remove_self_moderator_success ... ok
test internal::communities::test::add_already_moderator_success ... ok
test internal::communities::test::create_community_success ... ok
test internal::images::test::add_image_invalid_fail ... ok
test internal::communities::test::subscribe_success ... ok
test internal::communities::test::remove_nonmoderator_fail ... ok
test internal::images::test::add_image_too_big_fail ... ok
test internal::communities::test::remove_nonexistant_user_fail ... ok
test internal::communities::test::remove_moderator_success ... ok
test internal::images::test::add_image_success ... ok
test internal::images::test::remove_image_not_found_fail ... ok
test internal::messages::test::send_message_receiver_notexist_fail ... ok
test internal::images::test::remove_image_success ... ok
test internal::communities::test::add_moderator_success ... ok
test internal::posts::test::comment_success ... ok
test internal::posts::test::delete_post_success ... ok
test internal::messages::test::send_message_success ... ok
test internal::messages::test::get_chat_success ... ok
test internal::messages::test::unread_mark_success ... ok
test internal::posts::test::create_post_success ... ok
test internal::test::index_success ... ok
test internal::test::invalid_path_index_success ... ok
test internal::remotes::test::add_remote_notadmin_fail ... ok
test internal::images::test::remove_image_not_authorized_fail ... ok
test internal::users::test::create_user ... ok
test internal::remotes::test::add_remote_bad_format_fail ... ok
test internal::users::test::delete_user_fail ... ok
test internal::posts::test::edit_post_success ... ok
test internal::remotes::test::add_remote_bad_host_fail ... ok
test internal::users::test::logout_fail ... ok
test internal::users::test::delete_user_success ... ok
test internal::users::test::login_fail ... ok
test internal::users::test::logout_success ... ok
test internal::posts::test::nested_comment_success ... ok
test internal::users::test::login_success ... ok
test internal::users::test::change_password_success ... ok
test internal::remotes::test::add_remove_remote_success ... ok
test models::internal::test::userid_username_parse ... ok
test internal::users::test::forgot_password_success ... ok
test models::internal::test::userid_hostname_parse ... ok
test models::internal::test::userid_hostname_with_port_parse ... ok

test result: ok. 55 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 12.80s

     Running target/debug/deps/backend-4474c1882f83ebb2

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

   Doc-tests backend

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Figure 9: Backend Unit Test Runs

We use Sentry (figure 10) for error tracking as it provides a convenient interface to examine a request that failed; giving the backtrace if a panic occurred or the log messages leading up to the error. It can even provide a ‘cURL‘ command to replicate the request being viewed. Prometheus and Grafana were deployed on a lab machine using ‘podman-compose‘ and used for collecting and presenting various metrics such as memory utilisation and average request latencies (figure 11).

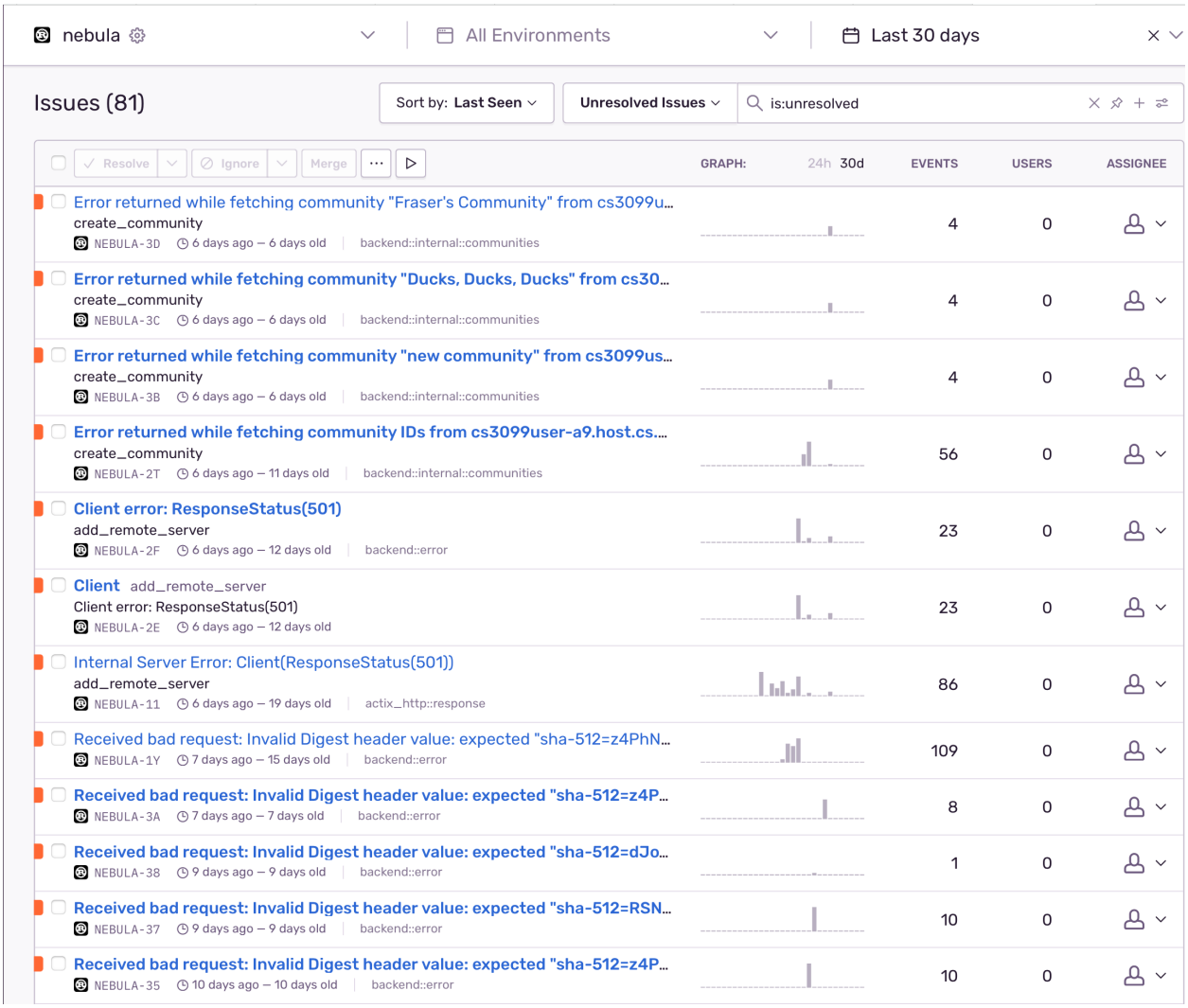


Figure 10: Screenshot showing Sentry usage



Figure 11: Screenshot showing Metrics Page

5 Agile and Scrum - [redacted]

Our scrum and agile methodology for development has not changed much throughout our development of the site. Each sprint starts with a sprint meeting. This is often a combined stage with the previous sprint's retrospective. In which we evaluate the previous sprint, looking for places in which we can improve and utilising the last sprint's feedback to decide on tasks we want to accomplish in the next sprint and to overall improve the productivity of the next sprint. The sprint retrospectives as mentioned are used to go over the failures and successes of the last sprint. In these we discuss a few things, including but not limited to

- Functionality we've implemented
- Problems we have faced
- The sprint duration - evaluating if the duration needs adjusting
- Future functionality/feature Improvements

These sprint retrospectives are crucial to improving our site and making us more efficient as a team. This is the main thing in the Agile process that has changed throughout this project; our ability to utilise the agile development method to its fullest, making us better at being more efficient and concentrated on the tasks we are given, as well as meaning we are more realistic in what we hope to accomplish in each sprint.

During the sprint we have regular (normally every 1 or 2 days) "stand-up meetings". These are more colloquial and are used to gauge the progress of each member of the group's tasks. Normally in these meetings we discuss the following:

- Our progress, here we can ensure that we are on track to clear the sprint backlog (or identify any changes which need to be made)
- Problems faced. This a great opportunity to communicate problems so that we can effectively deal with them

This keeps every member of our group up to date with what the other members of the group are up to, as well as being an opportunity for members of the group to discuss ideas for implementing features, or to discuss problems they may be having. Other group members can then acknowledge these problems, and, if wanted can suggest feedback or ways to solve them.

We have used the term "keeping our finger on the pulse". Overall this is the goal of all of our meetings. Ensuring that each team member is in the loop therefore avoiding the possibility of repeating work whilst making sure that every member is utilising their time on the project as best as possible.

To supplement the agile development process we used tools such as teh "Kanban board" on github 13. By transforming a sprint backlog into github issues, the kanban board would immediately show them in a Trello-like format, allowing us to easily track which features have been implemented, are in progress, or need to be implemented.

Using branches on github was also crucial to our success in this project. This allowed us to easily work incrementally, without disturbing any existing functionality we may have. By ensuring each pull request into the main branch was peer reviewed by at least one other member of the team we were able to maintain a high standard of code, as well as a consistently functional deployed site.

Overall I believe as a team we were able to utilise agile well to incrementally develop the project and manage our expectations of what we could achieve in a given timeline. The result of this was a project we are all extremely happy with, which we managed to achieve despite other commitments and being a smaller group.

6 Evaluation and Critical Appraisal - [redacted]

6.1 Comparison with the initial Specification - [redacted]

Below, we have made use of the initial requirements we set out in the first submission and indicate whether

Requirement	Status	Comments
Our platform must be accessible with any modern, mainstream web browser	Complete	Need to have javascript enabled (vuejs). Tested on Chrome, Firefox and Opera.
Users must be able to create accounts (with their university e-mail addresses) on the server network	Changed	We decided to remove this constraint and not require users to provide an email address, authentication is achieved using a username and password combination
Registered users must be able to create groups/societies on our server	Complete	
Registered users must be able to join groups/societies hosted on any server on our supergroup's network	Complete	Known as "subscribing" to a community.
Groups must have 'group pages' which can be accessed and viewed by users	Complete	
Group members must be able to write (text-based) posts on their groups' pages	Complete	
Registered users must be able to send direct messages to other users on the network	Complete	Uses HTTP requests, in process of upgrading to utilise web sockets.
Registered users must be able to delete any posts or unsend any messages they have written	Complete (delete posts) Disallowed unsending messages	Unsending messages is not implemented in the federation protocol and cannot be done.
The account details of registered users must be securely stored and safe from potential attackers	Complete	
Groups should have administrator functionality - that is, some members of a group may be its administrators, and will be able to perform operations that normal group members should be unable to	Half Complete	Backend endpoints all setup, just need a frontend interface for them.
Certain groups should be made exclusive by their administrators, so that only users who receive invitations (sent by said administrators) may join.	Not Completed	

Requirement	Status	Comments
Posts should be able to contain more than just text - for example, it should be possible for users to add images to posts.	Completed	Images can be uploaded to the server, and accessed directly or through embedding in markdown.
Registered users should be able to report content which they consider offensive or inappropriate, leading to it being removed.	Not Completed	
There should be an account recovery feature, enabling a registered user to either change or be reminded of their password if they ever forget it.	Completed	Uses uniquely generated recovery phrase to change password
The direct messaging feature could also facilitate voice or video calls between users.	Not Completed	
Subscribing to receive live notifications for new posts, comments and replies.	Not Completed	
There could be a separate bot API, enabling users to create bot accounts. These accounts could then be used for tasks such as automated messaging or page administration.	Half Completed	While a bot could easily create a regular user account and interface using the HTTP API, no specific functionality has been added to make bots easier to implement (such as API keys instead of JWT authentication).

6.2 Features implemented and how they can be improved as well as what might be considered missing - [redacted]

(Features implemented have been taken from the second submission, with features being implemented after the second submission added below, with further comments also added)

6.2.1 Backend - 170005973

Functionality Implemented	Details of Functionality Implemented
Users can create an account and login and logout as they need to	JSON Web Tokens (HS256 algorithm) are stored in the database so backend application is now stateless and therefore improves scalability
Users can change their password as well as reset their password if they've forgotten it (using the recovery key provided to them when they register an account)	Recovery key is generated during account creation and consists of 6 BIP39 words Password reset endpoint requires either that the current user is logged in or the recovery key is supplied
Users can delete their account	
Users can send and receive messages from other users	Including users on remote servers
Users can subscribe and unsubscribe from communities	
Users can create new communities	
Federation	All endpoints from latest specification have been implemented Remote servers can fetch communities, user information and posts, search for communities, users and posts and create new posts Federation tested successfully with group A1's backend
Federation Security Protocol	Middleware to verify signatures of all incoming requests implemented and tested Federation client library adding signatures to outgoing requests implemented and tested

6.2.2 Frontend - [redacted]

Functionality Implemented	Details of Functionality Implemented
Login page	Users redirected to this page if the page they are requesting requires authentication, such as homepage (this may be subject to change), community pages. Fully functioning.
Register page	Includes a password strength bar to communicate to the person registering the strength of their password. Once a user is registered, the system takes them to a page to display them their recovery key before (after 10 seconds) taking them back to the homepage. Fully functioning.
Forgotten password page	Form (available when a user is not logged in), which allows the user to change their password if they have forgotten it through entering their username and recovery key.
Change password page	Form (available when a user is logged in), which allows the user to change their password, if they wish to do so, through simply entering their new password and confirming it (a password strength bar is also provided here).
Navbar	Implemented site-wide navbar, which has links to settings, prole, search bar (fully functional) and homepage.
Homepage	Displays all the users to which the user has unread messages from. A link to the chat dialogue page. A button to create a community (which gives a layover to allow the user to enter information about the new community they wish to create). A button to open the chat dialogue page. Space on the homepage mainly taken up from posts, but instead of posts being universal, they shows posts from specific communities.
Admin page	Only accessible via the navigation bar, for users who are admins. On the page, all users who are admins are displayed.
Profile page	Allows user to change their password (giving them a link to the change password page), when they visit their own profile page. Allows the user to also delete their account, if they decide to visit their own profile page. Shows all the relevant information about the user that is signed in (for what we have implemented thus far). Allows users to change their profile picture when on their own profile.
Chat dialogue page	Allows for users to read and send messages to specific users, as well as searching for users to send messages to and read messages from them (with a sidebar supporting this).
404 Page	A page to which a user is redirected if they are trying to access a page that doesn't exist.
Search Result page	A search page which returns all the communities and posts which match the search term, accessible when searching something through the search bar. (Once all posts and communities have been returned, further functionality is available such as viewing a community page or commenting on a post).

Requirement Comments	Status
Community Page	A page, exclusively to display information about a community and allow the user to interact with that community (such as subscribing to it, creating a post, seeing information about the community in the about and it's moderators).
Posts	"A glimpse" of the full details of the post. Also provides a thumbnail showcasing any media within the post. Provides full functionality to interact with the post such as comment, delete, edit and like/dislike.
Full post page	Users can view posts and request to see more information. Renders the images and markdown in full. (maintaining post structure)

6.3 How above features can be further improved - [redacted]

- An improvement on the changing password functionality, Instead of the user having to enter a recovery key to reset or change their password, one way to improve on this is to have the recovery key sent to their email address or phone number (if recorded at registration stage) and then verifying themselves by entering the correct recovery key.
- Following on from the previous point, another feature we could add is multi-level factor authentication, when the user logs on or decides to change their password. (This feature is discussed further later in the evaluation, when discussing user security).
- One way the direct messaging could be improved on is to allow for media such as photos, audio and videos to be sent between users. Another follow on feature could be allowing for the facilitation of voice or video calls.
- The way subscribing to communities could be improved is to allow for users to subscribe to a set of communities (e.g: all communities to do with a particular sport or all communities to do with music).
- When a user registers their account, one improvement that can be made is by add an account confirmation feature where a user enters their email address and has to confirm their account once created (by accessing a link that is sent to them by email or to their phone number), this would avoid a person's email address being used on the site without their knowledge.
- When a user wishes to change their password, the way this process could be improved is instead of the user having to enter a recovery key to reset or change their password, we could have the recovery key sent to their email address or phone number (if recorded at registration stage) and then verifying themselves by entering the correct recovery key.
- When a user deletes their account, it would be a good addition if they got confirmation (via email or a certificate) that their account has been deleted.

6.4 Summative Statement - [redacted]

On the topic of considering what we have achieved and the scope for further improvement, we would believe that not meeting significant extensions for this project (given the timeframe we had and current working situation we found ourselves in) is actually not a bad a thing, as we made the decision to ensure that our system had the fundamental features and was built to a high quality, rather than taking risks to implement ambitious extensions and neglecting the basic functionality of our system, this decision came informally mainly due to the size of the team (being smaller than the average size) and what we deemed to be achievable at given points throughout the year, given the timeframe we had.

6.5 User Interface, User Experience and User Security - [redacted]

6.5.1 User Security - [redacted]

One aspect of user security we decided on early on is in relation to restrictions on user passwords to the site. On this point, we decided to only set a minimum length for the password that a user can use (that being 8 characters) and the user can use any password they like that meets the requirements. Furthermore, when a user enters their password when registering their account, they are provided an indication of how strong their password is, through a password strength meter. We agreed and believed as a team, that it's the onus of the user to set strong passwords and setting requirements for passwords was actually counterproductive. When reading the guidance on password administration (for system owners) (9), the guidance states "This guidance advocates a greater reliance on technical defences and organisational processes, with passwords forming just one part of your wider access control and identity management approach" and further adding "the NCSC do not recommend the use of complexity requirements when implementing user generated passwords". Further outlines "a greater reliance on technical defences and organisational processes", which mean following this guidance, that would be what we would greatly focus on if we were to build further on this system. One such idea to achieve this, and this is something that is become widely adopted with many larger sites, especially when suspicious activity seems to be taking place (for example, a person logging in with an IP address that indicates the person is in a country the account owner has never logged in from), is the use of multi-factor authentication. This brings added security in the sense that the user not only needs to know what their password is, but they must authenticate via two vector routes (and in total multiple ways), whether that be using something biological (such as a fingerprint, which would seem too extreme for the type of system were producing) or something they have (such as their phone with their corresponding phone number, which would fit more with our system), so to further extend our system we could add two-factor authentication (when signing in with a new IP address, that's different to one the user has sign in with before), where the user needs to be able to state their password and authenticate themselves through a text or phone-call to their mobile phone.

6.5.2 User Interface and User Experience Evaluation - [redacted]

So in terms of our user interface and what the user sees and interacts with, our system by an abstract description (and aided with the technologies we have decided to use), has different sections to which a user can do different things, this includes and is not limited: register an account, creating a post and viewing posts on a communities to name a few. We have taken this component structure and departmentalised our system in such a way to allow for the benefits outlined below. However, we realise that no user experience is perfect, with even the best system always lacking in optimal user experience and so we also explore points to our system's user interface which could be improved on.

As a team, we believe one good aspect to our user interface is that we have compartmentalised parts of our system efficiently, since our system includes a dedicated component (and corresponding sub-components) relating to direct messaging, a framework to allow for information and posts about communities to be displayed as well as a homepage which displays all relevant posts from different communities. We have also treated the homepage as a sort of "spider-web center" such that everything from the homepage is accessible with the smallest possible amount of key-strokes and button clicks, hence following HCI (Human-Computer Interaction) guidelines for user efficiency and control. This design principle (ensuring content is accessible with the smallest possible input from the user) is one we've planned from when we started the project and have maintained to ensure the simplicity of our system to enhance the user experience, as well as ensuring our system can be further expanded and worked on.

A further aspect of the user interface and user experience, we believe would make our system even better is if the site was a more "smoother" interaction and loading of content took place in a more gradual way. The reason this is mentioned is because loading of content on our system can be quite "clunky" with content almost suddenly appearing on the site when being loaded, rather than gradually appearing and therefore being less impactful to the user's eye.

6.5.3 Advantages and Disadvantages of Publicly Available Systems and Comparison with our System - [redacted]

The current market on social networking sites is varied and allows for various forms of multimedia to be shared amongst people and in almost instant time (especially when compared to what was possible 10 years ago), as well as more sophisticated solutions being present with lower and lower performance requirements to access such services. We not only took inspiration from such sites (such as Twitter, Facebook, Reddit) to motivate what we would be creating, we also explored the benefits and drawbacks of such available systems, we have compared it to our own system, in what we have managed to produce. Below are some features and traits that are present on specific and various publicly available systems, with them evaluated against ours.

One particular quirk of certain systems like Twitter, Snapchat or YouTube, is that users are given a unique experience, since what can be facilitated is limited which gives a unique experience to users and encourages a certain type of dialogue. For example, on Twitter, content is encouraged to be short as short as possible, where posts are limited to 280 characters, photos can be uploaded as well as videos, where videos with shorter length are encouraged and favoured (10). Also considering Snapchat and what content is called, on Twitter posts are called "tweets" and on Snapchat posts are called "snaps", this gives a quirky element to such platforms and is much more fashionable than calling such content "posts". In relation to our system, we haven't imposed any sort of restriction on the content that a user can post (e.g: maximum length of a post or number of posts a person can make in a day) as especially it this system was to go live, we would want to encourage as much content be be uploaded as possible to in effect "kick-start" the energy on the site. However, that is definitely something we may wish to consider if we were to further develop our system (for instance if video hosting was supported by our system, we may impose a limit as to how long videos can be if they are to be uploaded to the site, if they are non-academic related videos and the opposite for academic content, such as videos of lectures). We have also implemented our own naming scheme (outlined in the design section above), such to give our system its own identity and a theme that individuals associated with our system, in the same way people associated certain themes with other social media sites.

Following on from the previous point, publicly accessible systems, such as Facebook and YouTube have come an incredibly long way, compared to ten years ago, in relation to how quickly content can be shared to a diverse and large audience in a matter of minutes and seconds. This benefits movies users quick access to media and information but also allows individuals, organisations and institutions to share media much more effectively. This has also been helped due to the increases of availability as well as speed of internet and broadband. Some sites have even facilitated the ability for users to "share posts" (called re-tweets on Twitter), where users may see a post they think is worth sharing to other people they have a mutual connection point with (such as users who are "friends" with them or part of are members of the same "group" on Facebook , or users who "follow" them on Twitter, etc.), which leads to incredibly quick sharing of content, with particular content going "viral".

However, following on from the two previous points, a more serious point stems from them, for almost the same reason. The issue with short, easily digestible content, shared over a platform which allows for the sharing of content almost instantly (and to a wide scale audience) is that misleading information (dubbed as "Fake news") can very easily be spread, leading to ill-informed behaviour as well as potentially leading to dangerous consequences. A point clearly shown during the pandemic, with news and factually incorrect information about both COVID-19 vaccines and vaccines in circulation, and has lead to costly campaigns by health organisations and governments as well as general panic and confusion. Sites such as Facebook and Twitter have been continually condemned for their inability to control such content, and consistently in response have put measures in place to stop such content getting out of hand, such as disclaimer messages before users seem suspicious and the ability for users to report content. We have ideas on this point, which are further outlined in the future work section, in the conclusion chapter of the report.

A distinct advantage of publicly available systems,especially on sites like Twitter, Reddit and YouTube, they facilitate allow users to interact with other users of similar interest. The direction to this comes in many forms, for instance, sites like Twitter and YouTube give recommendations, (using sophisticated algorithms) to pages (or videos) which deemed on previous behaviour, is likely to be of interest to you. On many sites as well, search functions allow for users to search by keywords on particular interest or trending events. Twitter for instance has a "trending" section, which shows you the most words that have been tweeted the most in the relatively recent time period, with normally events with a mass following (such as sporting events

or TV shows) being talked about the most. In comparison, our system does allow for the facilitation of communities and for users to subscribe to communities that are of interest to them, however unlike most established sites, we have no algorithms on the backend which allow for recommendation of communities or posts which they may like to see, this could be something we wish to implement in the future (to ensure we give users the best experience possible and allow them to see content which they deem to be interesting and what they want to see). At the moment, the only data that is collected from the user is their username and email address (and chosen password for the site) and so we may wish to ask for more data at the registration stage to gage their interests and recommend content which is likely to spark their interest.We however, must keep in mind user privacy (especially as this is a point may platforms have been criticised on) and ensure that their use of data is ethical and legal. A way we could even distinguish ourselves from other platforms is ensure that how the data of users is used is clear, concise and easy to understand which is in contrast to many platforms available at the moment.

One however important disadvantage that comes from publicly available systems is cyberbullying. As again, a consequence to short and quick communication amongst users of these systems, users can quickly become victims of cyberbullying as unwanted content is made visible to them and the only way to filter such content, is to either report the user posting the content (which on some sites can take a relatively long time to resolve) or stop making use of the system, which means the individual misses out of the benefits of the system, for incidents they haven't incited. Our system doesn't have anything in place to stop any sort of cyberbullying taking place or identifying its root cause and preventing it from happening, (We have put the emphasis on the moderators of communities and the site to take responsibility for this). However to further extend our system, we could add keywords checker (with some form of language processing) to all posts to ensure that any identified hateful posts are reported instantly to administrators such that they can take action quicker (as without this it may take some time before administrators are aware of situations starting and by then the situation could be out of control). Furthermore, thinking about this system as a system for university students, their safety and welfare is incredibly important and focusing this as a university platform, we could think about how administrators can implement particular university's policies (such as, in the case of Saint Andrews University, Student harassment and bullying policy (11)) and have certain incidents reported straight to the relevant person/organisation (such as the Head of School for a particular School or Student services respectively) rather than the administrator having to deal with each individual incident. This would streamline the whole process and effectively allow the system to be self-managing as we would have been implementing a policy specifically designed to deal with such incidents of bullying and harassment.

A further disadvantage following on from the previous point, is the ability, on most systems, for users to be able to use such systems with a lack of identity. This directly links with the previous disadvantage, simply due to the facts these users know they can use the system without having to state who they are and through other techniques, to stay anonymous online, such as virtual private networks (VPNs), they can even evade enforcement, like the police. Comparing this point with our system, we actually don't even require an email address for users to sign up, so there's no real safeguards in stopping people from having multiple accounts and/or being anonymous on our site, which does pose risks, such as the issue above around cyberbullying. One way round this, if we were to further develop our system, would be to only allow users to sign up to the system if they have a valid university email address, likely utilising LDAP. We could even extend our system to not have communities but add a layer of abstraction above that being universities, so users join a university community and within that university community are communities for particular interests (and they can only sign up to a particular university, if their email address belongs to the domain name of the university). However since it is unlikely the university would give us access to the school LDAP system it would require us hosting our own solution just for the proof-of-concept and so was decided against due to the complexity and time it would take to set up and maintain.

One last disadvantage to consider is the fact that publicly available systems can quickly become a prime target to hacker and malicious attacks, the earlier to gain access to information that would otherwise be hidden and the later to cause damage and inconvenience to users. Given the sort of data that users entrust with systems like Facebook and YouTube, such as personally identifiable data as well as payment information, users are of increased risk to fraud as well as personal risk if location information is available. This risk is heightened with the more information a system holds on their users. For our system, this would require periodically ensuring our system is secure, nullifying any potential threats and keeping up to date with

the latest techniques that are used to gain unauthorised access to system, and conducting the appropriate penetration tests.

7 Conclusions - [redacted]

7.1 Key Achievements and Significant Drawbacks - [redacted]

One key achievement of our team as a whole, is what we've actually been able to achieve given the fact we are a team of four. In informal comparisons with other groups, we would consider ourselves one of the leading groups in terms of what we've been able to produce, even though we are a person down to the average sized group. Not only does having one less person mean we lose out on work-hours towards the project, it means we miss out on the skills that we could have made use of and learnt as a team from. However, the way we got round this was again going the extra mile and learning new technologies in our own time and making up for that lost time. We also felt the combined efforts of team members has paid dividends over the academic year (especially during paired-programming sessions) and has really accelerated the progress we have made. Furthermore, we also turned this handicap into an advantage, by ensuring that we took advantage of the easier communication that comes with only having to communicate with a small number of team members.

Another key achievement of our team is being able to produce a high quality system, which does the fundamental aspects really well, exceeding the minimum requirements and allowing us to implement several features of high quality (rather than implementing more features but having a flimsy and fragile system). We decided throughout the academic year to focus on ensuring what we made was of high quality and only move onto the next feature once a preceding feature was implemented to a high quality, this tied in with the backend language we decided to use and it being type safe.

The largest drawback of our system currently is the compatability issues it faces:

- Use of a modern framework means poor backwards compatability on older web browsers.
- Poor compatability on mobile and smaller devices.

The first of which is not a large issue, but through our use of VueJS as the main framework many older browsers do not support this, and so the site is inaccessible through these sites. The latter of these two is simply due to not having time to adapt the site to fit a mobile sized browser. It would mainly be a case of CSS and styling which would need adjusting, and this is something we would plan to do as a future extension.

Going forward, our system in its current form, has plenty of scope for improvements, since we have ensured that our fundamental requirements and functionalities are of high quality, giving ourselves or other people the ability to focus on further features and extensions on the systems, without having to worry about the fundamental features and re-paying technical debt.

7.2 Future Work: (what we would do given more time) - [redacted]

Here are a number of features, (if given more time), we would have like to have implemented (takes have been taken from the evaluation section, when considering features that have been implemented):

- An improvement on the changing password functionality, Instead of the user having to enter a recovery key to reset or change their password, one way to improve on this is to have the recovery key sent to their email address or phone number (if recorded at registration stage) and then verifying themselves by entering the correct recovery key.
- Following on from the previous point, another feature we could add is multi-level factor authentication, when the user logs on or decides to change their password. (This feature is discussed further later in the evaluation, when discussing user security).
- One way the direct messaging could be improved on is to allow for media such as photos, audio and videos to be sent between users. Another follow on feature could be allowing for the facilitation of voice or video calls.

- The way subscribing to communities could be improved is to allow for users to subscribe to a set of communities (e.g: all communities to do with a particular sport or all communities to do with music).
- When a user registers their account, one improvement that can be made is by add an account confirmation feature where a user enters their email address and has to confirm their account once created (by accessing a link that is sent to them by email or to their phone number), this would avoid a person's email address being used on the site without their knowledge.
- When a user wishes to change their password, the way this process could be improved is instead of the user having to enter a recovery key to reset or change their password, we could have the recovery key sent to their email address or phone number (if recorded at registration stage) and then verifying themselves by entering the correct recovery key.
- When a user deletes their account, it would be a good addition if they got confirmation (via email or a certificate) that their account has been deleted.
- One final idea is increasing compatibility our system on various types of devices (such as mobile phones and tablets) as well as implementing backwards compatibility on older web browsers.

7.3 What We Would Have Done Differently - [redacted]

On reflection, there are a number of things we would do differently, given the benefit of hindsight. One of which being more quick to create an identity of a system and from that identity create our system and be better informed in making decisions which would be irreversible. In terms of deciding on an "identity" for our system and what sort of experience the user would get in turn, we believe we were late in making that decision, coming to a naming convention of our system (such as names for the system as a whole, posts, communities etc.) around week 6 or week 7. This wasn't a massive blockage in development, however, we think it would of been a good idea to have a finalise identity and semi-final idea of what the system would look like such that we could have encompassed and tied in that identity in the features we were implementing, more specifically on the frontend.

Another thing looking back could have done differently was our expectation on what we might achieve in a sprint and the corresponding sprint length. Looking back at our meeting minutes, we noticed that would frequently be over-ambitious with what we plan to achieve in a certain sprint, especially given deadlines that have appeared for other modules and team-member's other commitments. Whilst we thought it would be better to be over-ambitious and achieve more than if we were to be under-ambitious and achieve all our targets, probably in the interest of team-morale it might of been a better idea to increase our average sprint length or hold back on items on the upcoming sprint backlog. This however is a tricky area to judge, since what can be achieved in a given amount of time (especially whilst being full-time students) is tricky and is more of a skill that is honed in over time, rather than being able to instantly work that out. We could have also used techniques to judge how long it may take a particular feature to be implemented and allow that to inform us about our plan going forward and what an upcoming sprint length might be, such as Planning Poker (outlined the CS3099 lecture titled Scrum) (12), make use of a Software velocity burndown chart or even potentially talking to fourth year students about certain features and what they think is realistic. Planning Poker was a technique discussed in the Scrum lecture of CS3099 and its where for each backlog item each team member makes a private estimate and all estimates are revealed simultaneously and if all the same, we come to a consensus otherwise we discuss and repeat the process until we come to an agreement on the length of time it would take to implement a backlog item. This would have not only allowed for us to have come to a more accurate estimate of the length of time to implement a backlog item but it would have also allowed us to understand each other's estimates and share ideas on how a particular backlog item might be tackled. We also initially planned to make use of a software velocity burndown chart, however, during the middle of the first semester we thought the value it would add to the development effort, given the time it would take to create, update and maintain would not be as much as simply concentrating that same effort on working on the system instead, especially given we were a team of four and therefore "developer time" was more valuable. We did however make use of a Scrum board, which more helped us judge our current progress to our (less and less arbitrary as we got closer to the end) to what we wish to achieve as well as manage dependencies of certain features on our system as well as what is achievable and not achievable given

the time that we had left (at any given moment, which again became easier to do as we got closer to the final deadline).

7.4 What We Have Learnt From The Process - [redacted]

One thing as a team, we have learnt from partaking in this module and has been a skill we've been honing in over this academic year, is being able to work successfully as a team, whilst all team members are remotely located. Given the pandemic and the government guidance on household contact, not one team member has actually seen another team member in the flesh. Which is a challenge in itself, especially given some of the practices we wanted to implement such as paired programming, use of a shared whiteboard to talk over ideas and just the impromptu bouncing of ideas that's only possible in person. However, we worked round this challenge by persevering and being creative with solutions to such a problem we found ourselves in, for example some practices we went with include: using Visual Studio LiveShare and organising a Team's call to conduct pair-programming, using the whiteboard feature in Microsoft Team to brainstorm ideas when we were to have meetings as well as organise impromptu calls to discuss a certain aspect of the system as well as impromptuly sharing ideas in the Team's group chat we had, to get feedback from other team members. These were just some of the ideas we had to ensure we could be as productive as possible, whilst all being remote. Furthermore, one could even see this as practice for a more and more common work-practice in the Software Engineering industry, that being individuals working from home and if team members are to have a career in Software Engineering they could even bring some ideas to their to-be teams.

Another element team-members have learnt from the project is learning new programming languages and frameworks. Before this project commendation, there we gaps in team members knowledge, for instance some members have never used the programming language Rust before and are only briefly aware of the framework Vue.js, however, from independent research to learning from each other, we were able to collectively improve our knowledge of the technologies used but further actually apply such skills to actually implement our system, which is a skill in itself. Looking back, it felt like semester one was a test-run from what we did and didn't know (and having to making use of team-members skills effectively such that we met the minimal viable product), we then took the the Christmas break as a opportunity as a team to fill in those gaps in our skillsets (for instance one member of the team ran through a Vue.js course on YouTube (13) and went through it over the break) and we went into semester 2 a lot more prepared and was able to make significant progress to what we have achieved to date.

An extension to the previous paragraph, another aspect of the project we have been able to to take learnings from is about Version Control and making use of it on a project of this scale. Team members have individually used version control for personal projects and documents, however most have never used version control on a team project, which brough different challenges (we decided to use Git for our version control). However, we set up a successful workflow and allowed team-members to take lead on this as well as teaching other team-members about the basics and how to utilise version control effectively. One key aspect of our workflow which was successful was when a pull request was made, we required that one other team-member approve the pull request, before it was merged into the main branch. Furthermore, when a major change to the project was made, team-members would indicate that to the rest of the team on our group chat on Teams and made recommendations to team-members to rebase their local repositories.

Lastly, a further element we as a team have taken learnings from is about agile software development and associated techniques but also actuallying using those principles for development. On this point, this was team-member's first major project they have been building, as part of a team where agile software development was encouraged. And whilst we may have learnt of it theoretically from lecture videos uploaded by Dr Edwin Brady and independent research, actually putting into practice was a different challenge. As a team, we have definitely risen to that challenge and have followed such principles and techniques throughout the year, these include and not excluded to: creating user stories, having a product and sprint backlog, making use of a sprint board, regular stand-up meetings and paired-programming.

7.5 Acknowledgements

We would finally like to take the opportunity to thank our project supervisor Dr Ishbel Duncan, for her guidance, support and consistent attention over the academic year. We would also like to thank Dr Edwin

Brady, for running the module and giving us the general framework and blank canvas to allow us to create what we have created.

8 Bibliography

References

[1] Axios
https://github.com/axios/axios

[2] Vue List Rendering, VueJS
https://vuejs.org/v2/guide/list.html

[3] Vue comparison to other frameworks, VueJS
https://vuejs.org/v2/guide/comparison.html

[4] Mathias Beke: On the Comparison of Software Quality Attributes for Client-side and Server-side Rendering
https://denbeke.be/thesis/versions/mathias-beke-final.pdf

[5] Stefan Krause: JS Frameworks Benchmark Table
https://stefankrause.net/js-frameworks-benchmark8/table.html

[6] Eric Wohlgethan: Supporting Web Development Decisions byComparing Three Major JavaScript Frameworks:Angular, React and Vue.js
https://reposit.haw-hamburg.de/bitstream/20.500.12738/8417/1/BA_Wohlgethan_2176410.pdf

[7] J. Ferreira, J. Noble and R. Biddle, "Agile Development Iterations and UI Design," Agile 2007 (AGILE 2007), Washington, DC, USA, 2007, pp. 50-58, doi: 10.1109/AGILE.2007.8.
https://ieeexplore.ieee.org/document/4293575

[8] Cypress.io: web testing framework.
https://www.cypress.io/

[9] NCSC: Guidance on Password Administration
https://www.ncsc.gov.uk/collection/passwords/updating-your-approach

[10] Twitter: 4 Best Practices for Short Video on Twitter
https://business.twitter.com/en/blog/creative-roundup-4-best-practices-for-short-video-on-twitter.html

[11] St Andrews: Student harassment and bullying policy.
https://www.st-andrews.ac.uk/policy/student-administration-non-academic-discipline/student-harassment-and-bullying-policy.pdf

[12] Brady, E 2020, Scrum: Sprint Cycles, lecture slides, CS3099 Software Engineering Team Project, Saint Andrews University, delivered online- pre-recorded.

[13] The Net Ninja: Vue JS 2 Tutorial Series
https://www.youtube.com/watch?v=5LYrN_cAJoAlist=PL4cUxeGkcC9gQcYgjhBoeQH7wiAyZNrYa

9 Appendices - [redacted]

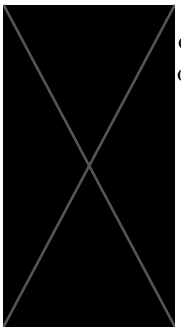
Meeting minutes- The earlier meeting minutes of our meetings, can also be found in previous reports, (ones whose date is before the submission date of the respective submission). The meeting minutes of the meetings post the second submission have been added below and have not featured before.

9.1 Scrum Meetings

9.1.1 Meeting held on 22/10/2020

Sprint Retrospective *(since we didn't have formal sprint previously, we just looked at what has gone well, what would be even better if and team-members would demonstrate what they have created thus far, the naming will remain the same to stay consistent)*

Sprint Review



ompleted the basics of the database, went through the schema and said there was still o do.

lked about the progress and current status of the supergroup protocol.

lked about what he has been learning, specially on how the frontend and backend worked

ent through the login page and register page (which has no endpoints).

owed us an alternative to Diesel, which was SQLx, the group seemed heavily in favour x.

What went well

- As a group we felt like the project is starting to come to life, especially with now the basics of the database and parts of the frontend starting to be complete.
- We felt that the regular meetings we had previously worked well in our favour and we wish to continue doing that- we hope that the stand-up meetings will facilitate this.

Even better if

- It would of been even better if the supergroup protocol was are a later stage than is currently.
  - To combat this, we talked about how we would like the supergroup protocol to look and hopefully we can use that to help us negotiate with the supergroup in ensuring that the protocol works for us in the best way it can.
- It would been even better if the department GitLab installation supported runners.
  - [redacted] is going to follow up with Stuart in ensuring that runners on the department's GitLab installation is set-up.

Decisions Made

- Decided as a team to move to SQLx from Diesel, as the tool to communicate with the database.
- We came to the decision that the sprint will be 2 weeks in length, which means we will have our next Sprint meeting on the 6th November.

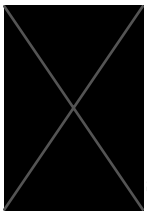
Sprint backlog for the next sprint

- Complete the login/register pages, ensuring that users can login and logout - [redacted]
- Complete database schema and finalise it - [redacted]
- Set up deployment using Heroku- [redacted]
- Complete Client Server API - [redacted]

9.1.2 Meeting held on 06/11/2020

Sprint Retrospective

Sprint Review


howed the finalised database schema to the team

howed the progress made on the Client Server API

alked about the current progress on the set up of deployment using Heroku

showed the team the login/register pages and  showed how it all works on the

What went well

- We felt that the daily stand-up meetings were working really well and as a team we saw much value in them and we decided that would definitely be part of our next sprint cycle.
- The team valued also the impromptu calls/messages team-members gave to each other and we came to a conclusion that we would attempt to have as many impromptu calls as possible throughout the next sprint cycle.

Even better if

- As a team we felt we could of made more progress in terms of the sprint backlog of the previous sprint, this was mainly due to coursework assignments getting in the way.
- Going forward we shall set informal deadlines, which individual team members will do their best to meet, such that we continue to make progress on the sprint backlog and in turn the product backlog.

Decisions Made

- Came to the decision there was no need for a follow-up sprint meeting as we decided that the sprint, working towards the minimum viable product (as well as as many extensions if time allows). We also came to the decision that the sprint backlog will be mainly focused on meeting the minimum viable product, however, that wouldn't stop us from implementing further features if time allowed.

Sprint backlog for the next sprint

- Complete Client Server API
- Set up retrieval of comments and posts (both for the frontend and backend)
- Create SQL commands to allow for the insertion, updating and deleting of comments and posts(i.e. the
- SQL commands to allow for certain functionalities to be complete)
- Complete Cookie authentication
- Complete Heroku set-up
- Connect end-points between the frontend and backend

9.1.3 Meeting held on 06/01/2021

Sprint Retrospective *(since we didn't have formal sprint previously, we just looked at what has gone well and even better if (from last semester), made any necessary decisions and created our sprint backlog for the next sprint)*

What went well

- Communication was key to our success last semester and we aim to continue that. We are also aiming to develop this further by improving our methods of tracking what work each member is working on.
- Scrum/Agile methodologies was working well for us and we hope to continue with those, as well as adopting practices like paired-programming and almost daily stand-up meetings.

Even better if

- We collectively thought that one issue we had last year was not always knowing what every team-member was working on, which lead to a occurrence where an effort was repeated, so as stated above we hope to introduce this new standard, which will allow us to work much more efficiently and ensure dependencies are respected.
- This involves further utilising our project board, and ensuring every individual feature is developed on separate branches to ensure atomicity and independency. As well as making it clear what features are currently being worked on.

Decisions Made

- Our next sprint will end on the 24th January (giving a sprint of 18 days), the reason for choosing that date was it was the day before the week of teaching began.

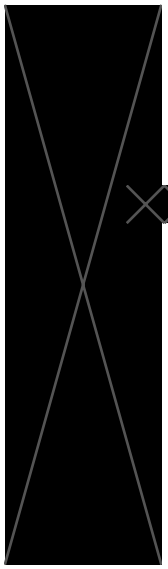
Sprint backlog for the next sprint

- Implement changing password and forgotten password function on our system
- Implement a password strength bar on the register form
- Implement nested comments
- Allow for deleting of posts
- Implement a search function
- Create a user's profile page
- Implement group pages

9.1.4 Meeting held on 24/01/2021

Sprint Retrospective

Sprint Review



s implemented account deletion and community creation on the backend and has also the search endpoints as well as implementing additional federation endpoints.

d [redacted] have been working on implementing the changing password and forgotten

[redacted] on the backend and [redacted] on the frontend.

plemented a password strength bar on the register page

s done up the main homepage as well as implementing nested comments and deleting orking with [redacted] to complete both of these on the backend as well as the frontend)

s also implemented a skeleton for search page and profile page

s gotten started on implementing the page for groups, as well as functionality to list all ps to join

s also implemented joining groups on the frontend, but needs the backend connection

What went well

- We felt for this sprint the (bi-)daily stand-up were really useful as individual team members came across a number of problems, which were quickly rectified as other team-member knew about them as soon as was possible and normally another team-member would have an idea for a solution to that problem.

Even better if

- We felt that that the sprint length was too short, especially with the sort of features and functionalities we were attempting to implement for the sprint.

Decisions Made

- We felt we would still go ahead with a two week sprint length (rather than it being longer), since the semester was starting again and so future progress won't be as significant as the one we did over part of the Christmas break, however this is a point we may wish to consider approaching the next break, that being Spring vacation and we accepted if there was items of the sprint backlog that weren't completed, it wouldn't be an issue as we would rather be over ambitious in our plans than under ambitious.

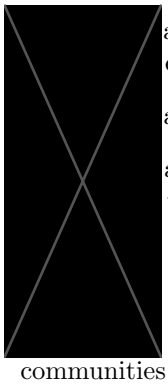
Sprint backlog for the next sprint

- Make comments refresh parent, rather than entire comment tree
- Finalise the search page and have a working first version
- Update the profile page to show all relevant information
- Implement latest federation specification
- Implement admin endpoints and moderator endpoints on the backend
- Have a Server setting page as well as endpoints on the backend for adding remote servers
- Implement the ability to create communities
- Start working on allowing for messages to be sent between users (chat function)

9.1.5 Meeting held on 08/02/2021

Sprint Retrospective

Sprint Review



as implemented community creation as well as account deletion on the frontend as well as ed on the chat function on the frontend

as also ensured that comments refresh parent, rather than the entire comment tree.

as changed the database to allow for foreign and local users all to be represented as well up to allow for chat functionality.

s changed the cookies such that they store JSON web tokens as well as implementing the the latest federation specification and implemented moderator and admin endpoints.

s started work on moderator page for moderating communities and has finished setup for communities page- connecting with with the rest of the frontend.

What went well

- Good amount of paired programming during the last sprint, which we think contributed to higher quality code and a learning opportunity for both the driver and navigator

Even better if

- We did feel the last sprint backlog was too ambitious, especially given the number of team members we have for our group in comparison to other groups. We feel this sort of issue of being over ambitious or under ambitious for the next sprint is not a exact science and is only a skill that is honed in over time rather than over an academic year. We are taking being over ambitious as a positive trait, since we would rather be over-ambitious and implement more features than be under-ambitious and not achieve as much as we could.

Decisions Made

- We decided the next sprint would be of length 12 days (rather than strictly 14 days) to give time to consider report writing.

Sprint backlog for the next sprint

- Mindmap ideas and experiment with potential themes for the system
- Create a first version of the community page
- Produce a first version for the chat functionality of our system
- Add more federation functionality
- Move the create post button to the individual community pages
- Change the URL format for profile pages, such that it would relate to the specific user (user/:userID)

9.1.6 Meeting held on 20/02/2021

Sprint Retrospective

Sprint Review

- [redacted] as implemented a first version of the chat functionality of the system of the frontend and nd running.
- [redacted] xed the database representation of users, implemented direct messaging on the backend, federation functionality, endpoint point for searching users and fixed the subscribe endpoint d a subscribe button).
- [redacted] ied up the file system in the code base, changed the url format for profile pages such that format user/:userID. A fixed a number of buttons and placed them in the right place on Also added some more error handling and a 404 page.
- [redacted] as produced a first version of the search page and has updated the profile page to show nt information.
- [redacted] orked with [redacted] to connect the community page with the rest of the website, and a so ma ng progress on the community moderator page, [redacted] has also started mind mapping potential themes for the system.

What went well

- Feels like things are coming together again for the final half of the second semester, since we feel we have brought everything together and the site is in a good place, most functionality required is there and we're in a very good position for the second half of the second semester

Even better if

- More federation functionality would like to be implemented, significant progress made in terms of federation endpoints so that remote servers can fetch our content, but remote server content is not very visible for our local users.

Decisions Made

- Decided on how the structure of the site should be laid out, including how the URL map works.
- Next sprint will have a length of 4 weeks (since we discussed about coursework that is required for other modules and thought 4 weeks would give us enough time to adequately work through the sprint backlog and focus on other modules) as well enough time just at the beginning of the spring break to plan our next steps for the final submission.
- We also decided this would be our last sprint as post the last sprint, we would turn our attention to the report, final submission and simply making minor tweaks and finishing touches to our system and having a sprint for that wouldn't make sense. So our next formal meeting would be our last one.

Sprint backlog for the next sprint

- Implement latest federation integrations
- Implement endpoint for liking a post
- Add like/dislikes/reacts to the frontend
- Update README.md file
- Get a first version complete for the community moderator page
- Implement post sorting on the frontend
- Changing setting page to an admin page
- Add edit button to a post to allow for editing of a post

9.1.7 Meeting held on 21/03/2021

Sprint Retrospective

Sprint Review

- [redacted] as changed the settings page to an admin page, which is only accessible by admins and ns to add and remove admins and well as remote servers (with [redacted] implementing end functionality such as retrieving all usernames who belong to admins), [redacted] has ented post sorting (on the homepage as well as within groups) on the frontend, which sorts tain attributes such as date posted or user who posted.
- [redacted] as implemented the ability for users to edit their posts on the frontend with [redacted] nternal endpoints for editing posts and implementing the functionality on the backend.
- [redacted] as also implemented site wide error handling as well as tidying up the file structure of the updating the community page.
- [redacted] as also implemented post image rendering and image uploading on the frontend.
- [redacted] as implemented the latest federation integration as well as implementing the image hosting en po n s with [redacted] implementing image hosting on the frontend.

What went well

- We're at the stage now where we can focus on putting the finishing touches on our system as well as make minor bug fixes when we discover them as well as turning our attention to the final report.
- We continue to work well as a team, communicating effectively, leading to seamless connecting up of the frontend and backend together.

Even better if

- We hoped to have made more progress on the sprint backlog, however given the several deadlines we had for other modules, on reflection, we made good progress.

Decisions Made

- We shall not have a formal sprint for the next couple of final weeks but rather just focus on a single core list of what is there left to do and do our best to work through that list as well as focus on final touches to the project and fixing any bugs that arise. We will also start to shift our focus on writing up the final report.

Whats Left to do

- Add likes/dislikes on the frontend
- Implement Post Pinning
- Frontend image handling
- Implement markdown rendering
- Implement Welcome/Login overlay on the homepage (removing the landing page)
- Graphing of statistics of system usage
- Adding web sockets
- Implement response and resource caching on the backend
- Improve integration of federated content

9.2 Weekly Progress Reports

9.2.1 Semester 1

Week 5 *Week beginning Monday 12 October 2020*

- Finalised the plan for the database schema
- Finalised choice of backend and frontend language
- The basic set-up was created for our project, in which we can start development form

Week 6 *Week beginning Monday 19 October 2020*

- Login/register page has been started (with no-endpoints), (not completed)
- Made a decision as a team to modify the initial plan which was to move over to SQLx from Diesel
- First draft of the database schema (i.e. the basis of the database), however not finalised

Week 7 *Week-beginning Monday 26 October 2020*

- For the last three weeks, we shall split into two groups; one group working on the frontend ([redacted] and 180005850) and one group working on the backend ([redacted] and [redacted])
- Login/register page has been complete, and system can login users in as well as log them out
- Database schema is complete, finalised and ready to use
- Heroku in the process of being set up
- Client Server API almost complete

Week 8 *Week-beginning Monday 2 November 2020*

- SQL commands created to allow for the insertion, updating and deleting of recording the table (i.e. the SQL commands to allow for certain functionalities to be complete)
- Set up a internal API on Swagger
- Client Server API complete
- Cookie authentication is complete
- Users can login and log out successfully (as well as register onto our site)
- Heroku set-up, (i.e. development, staging and production deployment environments created) and GitLab CI runners set-up, which allows for automated deployment to Heroku via GitLab CI

Week 9 *Week-beginning Monday 9 November 2020*

- Automated testing and integration tests (including units tests) have been created
- End-points between the frontend and backend created and match-up which allowed for:
- Implementation of the functionality which allows users to post comments and posts
- Implementation of the functionality which allows for the creation and deletion of posts, communities and users
- Also implemented the ability for a user to subscribe to communities
- Documentation added to the system.

9.2.2 Semester 2

Week 0 *Week-beginning Monday 15 January 2021*

- Password strength bar added to the register page
- Homepage revamped and updated
- Ability to delete comments
- Backend search endpoints implemented
- Additional federation endpoints were added
- Skeleton for group page created

Week 1 *Week-beginning Monday 22 January 2021*

- Change password and forgotten password function implemented (both frontend and backend)
- Joining a group implemented on the frontend
- Nested comments were implemented (including having recursive child posts)
- Skeleton implemented for search page and profile page
- Account deletion on the backend completed
- Community creation on the backend completed

Week 2 *Week-beginning Monday 1 February 2021*

- Account deletion on the frontend completed
- Community creation on the frontend completed (completing community creation)
- JSON web tokens now stored in cookies
- First version of the community page created
- First version of the profile page created

Week 3 *Week-beginning Monday 8 February 2021*

- Chat function on the backend implemented
- Database allows for both foreign and local users to be represented
- Endpoints implemented (including for latest federation specification, moderator and admin endpoints)
- Comments on post in a state for which we are happy with, refreshing parent, rather than entire comment tree

Week 4 *Week-beginning Monday 15 February 2021*

- First version of the chat element of the system is completed (both frontend and backend completed)
- Community pages implemented and connected onto the system
- 404 page created and now forms part of the system
- First version of the search result page created, (with a complete frontend and backend implementation)
- Profile page updated to show all relevant information

Week 5 *Week-beginning Monday 22 February 2021*

- Unsubscribing from community on the frontend completed, with endpoints for subscribing and unsubscribing being fixed
- Ability to edit posts, implemented on the frontend

Note: Little progress made during the weeks of 6 and 7 due to deadlines in other modules

Week 6 *Week-beginning Monday 1 March 2021*

- Sorting of posts have been implemented in both the homepage (where all posts are shown) and in the posts shown in individual community pages

Week 7 *Week-beginning Monday 8 March 2021*

- Nothing to report

Week 8 *Week-beginning Monday 15 March 2021*

- Site wide error handling has been implemented
- Image hosting endpoints have been implemented
- Setting page has been changed to a Admin page, which allows for the addition and removal of remote servers that the system is connected as well as users who are admins
- Switched to asynchronous clients for unit tests
- Implement metrics endpoint for scraping by Prometheus and displayed using Grafana, deployed to a lab machine
- Post image rendering and image uploading implemented on frontend as well

Spring Break Week 1 *Week-beginning Monday 22 March 2021*

- Users can now add a profile picture

Note: No much progress to report is notable from this point onwards, the reasons for this was that what was being achieved can be described with smaller scale bug fixes and adjustments, rather than any significant features being implemented and we also turned our focus to the report.

Spring Break Week 2 *Week-beginning Monday 29 March 2021*

- Nothing to report

Week 9 *Week-beginning Monday 5 April 2021*

- Nothing to report

9.3 Additional Figures

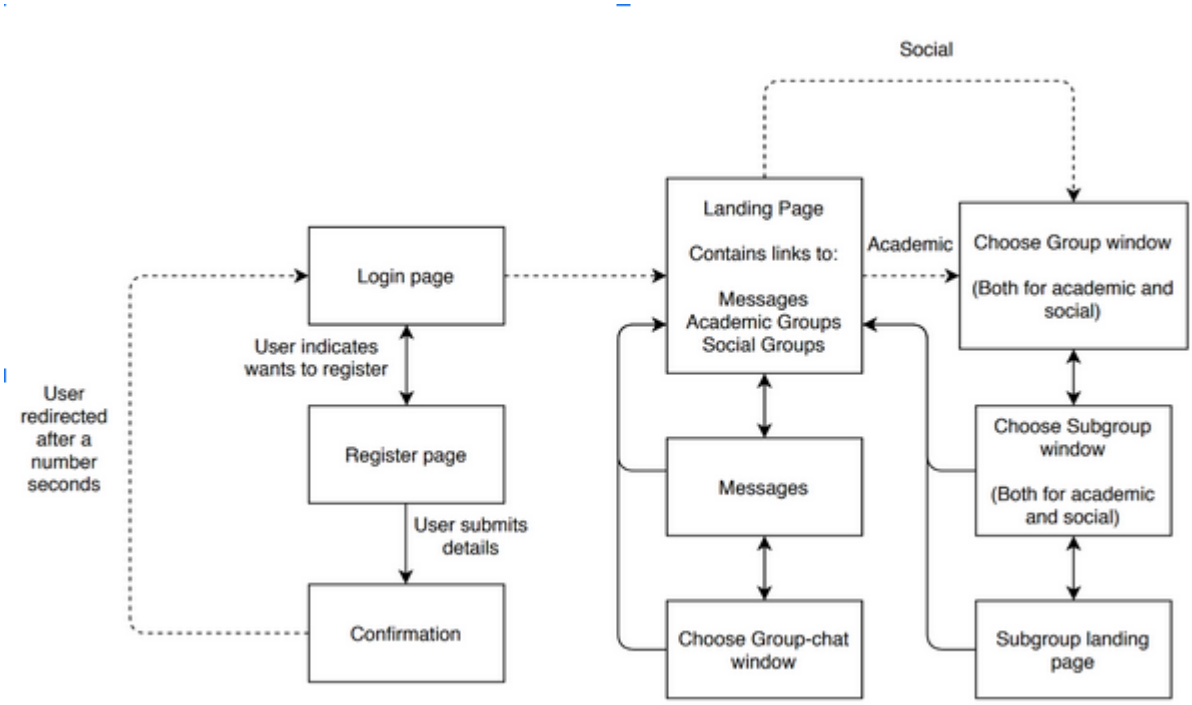


Figure 12: Sitemap Initially planned (can be found in first submission)

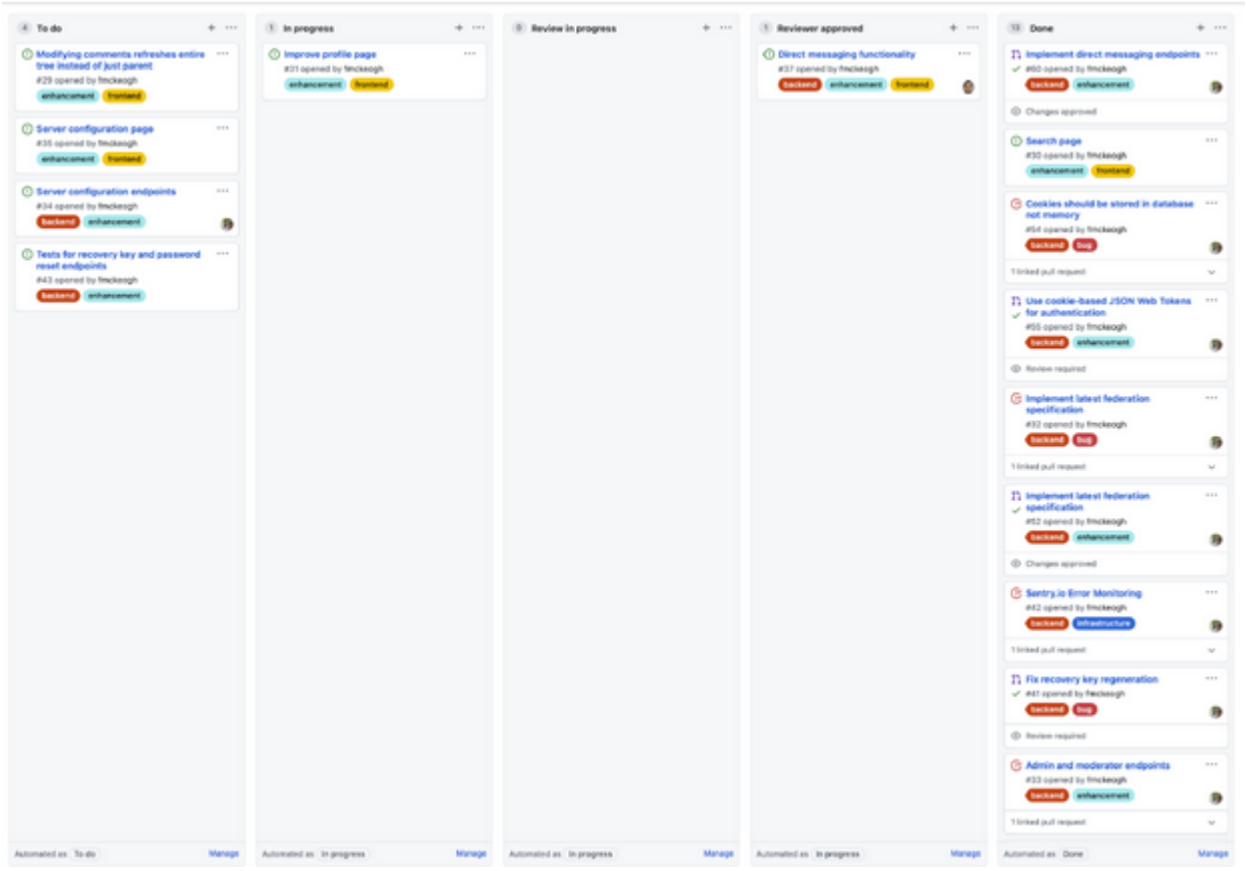


Figure 13: Example Sprintboard (can be found in third submission)

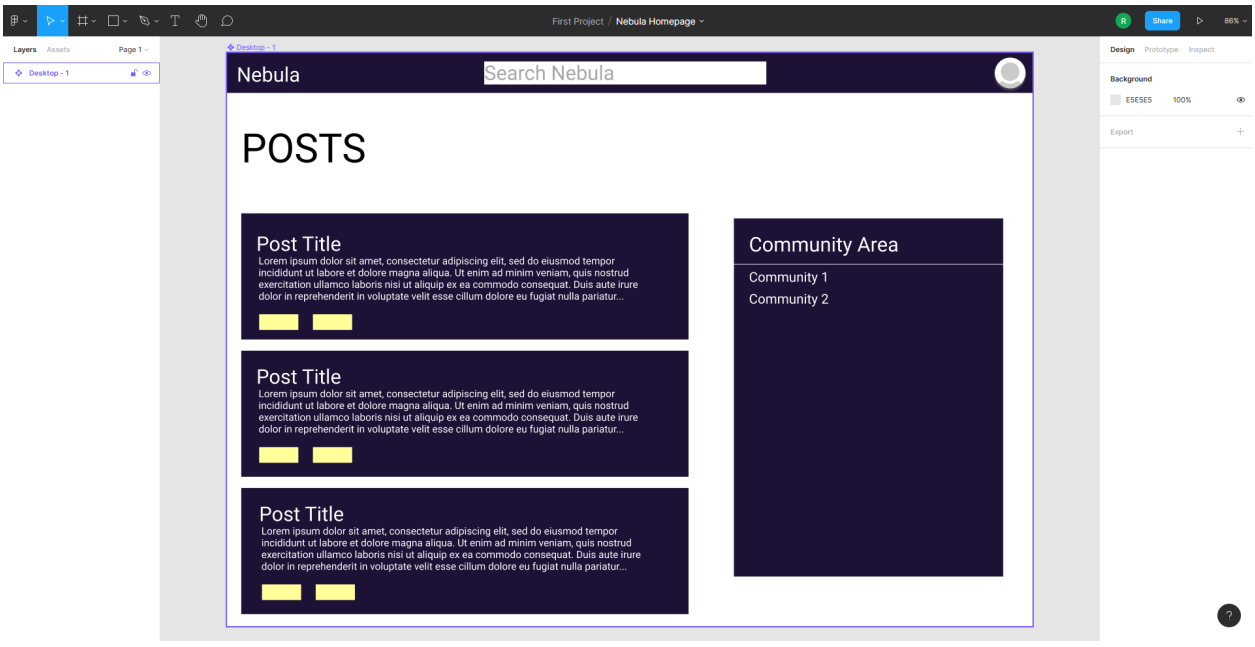


Figure 14: Figma for preliminary frontend design