# Zoosh - A Social Media React Application

**Aaron Moran**

**Conor Shortt**

**Thomas Kenny**

B.Sc.(Hons) in Software Development

May 5, 2021

**Final Year Project**

Advised by: Kevin O'Brien

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

# Contents

# About this project

**Abstract** This project is a Social Media platform where you can create communities, post up discussions and follow your friends. Our goal was to create a platform that performs well under stress testing and has very fast loading speeds while handling data through requests.

**Authors** Aaron Moran, Conor Shortt, Thomas Kenny.

# Chapter 1

# Introduction

The final year project requires you to show initiative, consistency and the ability to showcase what you have learned over the course of four years. Three members consist of Aaron Moran, Conor Shortt and Thomas Kenny. We are like minded students that share an interest for learning new skills and developing our ability as computer programmers. We have different skills that when combined complement each other from creativity to problem solving.

With most companies looking for experience with Agile development we thought this would be the perfect opportunity to gain experience using this methodology in a team environment. We broke the project up into several phases and collaborated with each other on getting features implemented. This would also give us structure and balance when developing our application. Agile development and Scrum methodologies go hand in hand, and so we decided that to increase productivity and maintain consistency, we would implement mock Scrum meetings each week, with short daily meetings in between. An advantage that we had over other teams was that we currently live together, so this allowed for much better integration as a team, as we could perform Scrum meetings in person. We will also gain valuable experience carrying out testing as when we are working within industry we will have to carry out extensive testing on important features we may be working on.

With our interest in social media we that a social media website was a project that we felt we all could contribute too. A social media website requires multiple features, the in depth use of a database and an engaging User Interface. We were confident from the start with the team we have we could develop an application that could meet the standard of the websites out in the world today.

Before settling on a idea, we researched many different websites and applications. The four main sources of inspiration came from the following websites Reddit, Medium, Instagram, Medium and Twitter. All of these applications serve the purpose of joining a community and sharing knowledge on a wide variety of topics. The design and layout we have implemented has been inspired by the way these applications have been built, the simplistic design allows for an easy to use application with a very small learning curve. The goal we had from the very beginning was to create an application which allowed the user to perform tasks with the least amount of clicks and navigating.

- Medium - A platform to read articles.
- Reddit - Front page of the internet.
- Instagram - Photo sharing platform.
- Twitter - Micro blogging social network.

From the research we carried out we held a meeting and brainstormed what our social media website would consist of. A website that users could create groups/societies and discuss topics with each other. Having multiple users engage with our website was the target. For this to be achieved our User Interface needed to be engaging and captivate users. By making our application very user friendly this could result in a daily active user rather than just a once off visit to our website.

With the pandemic and the effect it has had on student life during the college year clubs and societies became neglected we found this to be an opportunity to try and bring these clubs and societies to a digital format. There are universities around Ireland which have their own website for joining clubs and societies but they require an application process which then might result in a club never becoming active, these university websites also do not implement features such as creating posts, comments and reactions. This way we developed a website that does not require an application phase, and anybody can create a club or society online for free and share information as they wish. This would give users a chance to meet new people and engage with societies while not being able to socialise within the college campus, especially for new first years entering into third level institutions.

## 1.1 Github URL

https://github.com/applied-project-2020 GitHub was the obvious version control of choice. The development of the project was organized by the creation of a GitHub organization, of which we were all added as members. This allowed for the creation of multiple repositories to separate different final year project components: Software (Web Application, Mobile Application), and Dissertation.

- **Web Application**: This repository was the main repository we worked on for the duration of semester one. With over three hundred commits, it's fair to say that we extensively used GitHub to manage our collaborative development, while simultaneously refining our skills with Version Control software, which will definitely aid us in future team based developer roles. This repository containing our MERN stack application named ***zoosh*** was broken into two main folders: *Client* and *Back-End*. The *Client* folder holds the front end React application, which is served to the user through their browser, and is the application that they interact with. The *Back-End* folder holds the web server, that is run with Node.js which is an asynchronous, event driven Javascript runtime, used to build scalable network applications that can handle many concurrent connections. The server is used to make requests to the database, which is cloud-hosted at cloud.mongodb.com. The *Back-End* folder also holds all of the necessary database models, routes, etc that are used for the storing of data.

- **Documentation**: This repository is used to store the contents of our dissertation. Managing the development of our dissertation was something we discussed for a lengthy period of time. As Overleaf only allows up two collaborators to work on the same LaTex document simultaneously for free, but passed that it requires a monthly subscription to their premium service. Instead of this, we decided to manage the documentation with GitHub, and as such, we created the *Documentation* repository. This is also the ideal way to track the amount that we are each contributing to the documentation.

- **Mobile Application**: This repository is used to hold the mobile application version of our ***zoosh*** app. Although it hasn't been kept as up-to-date as our web application, it was still an interesting application to develop. The mobile version was created using React Native, which isn't that different structurally from ReactJS, only requiring a couple

of tweaks to our code. Developing the mobile application was purely for concept and researching the potential of porting the website to mobile, the database can be easily shared between the two as MongoDB allows you to sync the database from MongoDB Atlas to MongoDB Realm. This allows for a fast deployment of a mobile application as the only features that would need to be worked on would be the user interface design and some standard mobile application libraries such as notifications and touch gestures.

# Chapter 2

# Context

The purpose of our project is to allow people to communicate through communities sharing ideas and knowledge whether it be news or comical. The objectives which we want to achieve from developing this project is to create and application where users can follow, react and comment and join communities as they wish without any restrictions.

All of the development processes will be discussed throughout the document as follows:

- Introduction - This will run down through the team members and the reason as to why we chose this project.
- Context. (Current)
- Methodology - This chapter will discuss the steps we took to develop the project and the testing which we carried out along the way.
- Technology Review - This is where we run down through the development stack that we chose and why we chose it.
- System Design - This chapter discusses the design decisions that we chose and the implementation of all our components.
- System Evaluation - Within this chapter we discuss the issues we faced and how we carried out experiments to overcome and solve the bugs.
- Conclusion - This is where we discuss the final stages of development and what we learned while working as a team on a project.

# Chapter 3

# Methodology

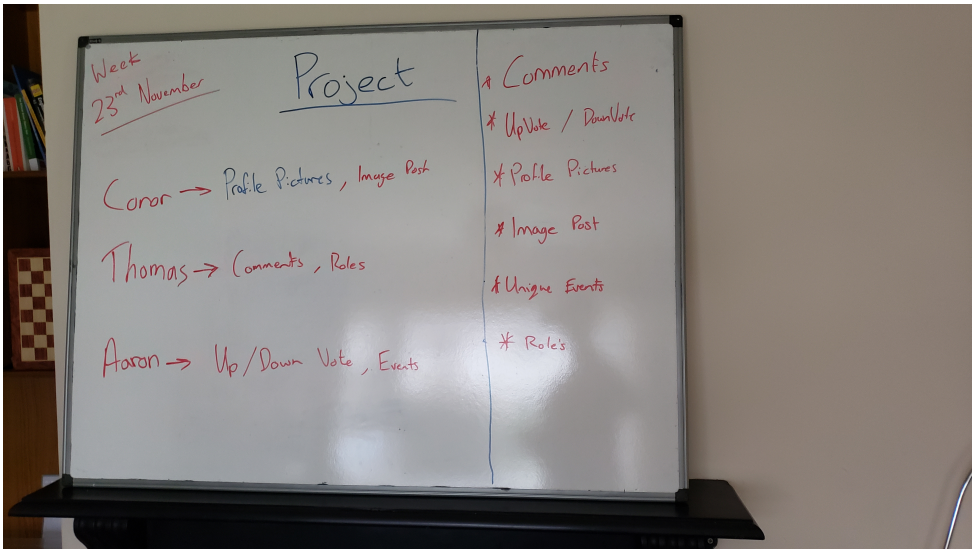
- For the development of this application, we decided the best approach to take was to use Agile. Agile is a methodology focused around iterative development. Using this method requires a team to work collaboratively and consistently. Before we started any development it was important to have a clear plan of what the features we wanted to incorporate and develop. Living in the same house proved useful in these times as we were able to have in-person meetings. Our first meeting we drew up a plan of how we were going to develop this application. Our first aim of was to agree on a development stack. After research into numerous development stacks such as the MEAN stack and MERN stack we decided on the MERN stack. We carried out extensive research into the factors that are common to all great websites, such as:

- **Functional** - Probably the most important aspect of every website. It must work quickly and as expected. Page loading times must be kept as low as possible, especially when working with large amounts of data - this was a big issue that we encountered when dealing with images.

- **Easy to Use** - User Experience (UX) plays a key role in helping visitors use, understand, and continue to use an application. If an application is difficult to use/understand, you can guarantee that a user won't stay longer than a few seconds. We made sure that the design we had created was user friendly and carried out blocks of black box testing to ensure that there was no confusion when using the website. Any feedback that we were given on positioning of components and the fluidity of the features we tweaked so that the design would relate to a lot of people and required no learning curve as navigating around the website was familiar.

- **Robustness** - The application should work when used on different devices, the page should automatically resize and adjust when used on different sized screens.

  With our choice of stack agreed upon, the next step was dividing the roles in the group. Conor and Thomas have an interest in back-end development and Aaron was keen on developing the Front-end. The first meeting was beneficial as we had a concrete foundation to build upon.
  We decided the best approach to development was to work on two - three features a week. We set up a whiteboard with a list of features to be developed, every week we would divide the features between the three of us. An example of our work process is shown below.




Each week we would have a meeting detailing the work done and evaluate the features developed. This was an important process as we got to discover what we could improve and what works well. If something needed improving we would put it back on the list for future meetings. If a feature was functional and proved to be efficient we would cross it off the board.
These meetings were also valuable for tracking bugs throughout the development process. If a team member encountered a bug or an issue we would write the it on the board and discuss it at the next meeting. Bug tracking was a process that requires good communication and teamwork as for an issue to be overcome we had to share solutions and

potential bug fixes.
Throughout the development of this project, we have used many collaboration tools including GitHub, Slack and Discord. While using these tools we were able to communicate new ideas and solutions to problems we may have been facing. GitHub helped us to track our progress and any mistakes that were made we could easily rollback commits. As we are all working on this project remotely, VOIP was incredibly important as we could communicate instantly, raising any issues we may be having and coming to brainstorming a solution.

## 3.1 Testing

Testing was a vital part of the development process. Using Agile required us to test after each feature was incorporated. We used many testing methods during the process of this application the main testing methods used were unit testing, smoke testing ,regression testing and stress testing.

- **Unit Testing** is type of software testing where individual components are tested during the development phase.
  We decided as a collective that it would be best if these tests were carried out at the end of the week. This was decided as features would be implemented at the end of week. If a component failed one of these tests we would work together to get the feature up and running. Carrying out unit tests provided our application with more efficient components and also saved us time.

- **Smoke Testing** is a type of software testing where tests are carried out to ensure the most important features work.
  We carried out smoke testing once a major feature was added. One of these tests carried out was to ensure our feed was working correctly. The feed was an important feature, this required us to test it quite frequently. As a result, the feed became more efficient and improved with every test.

- **Regression Testing** is used to ensure that no recent changes have altered previously working features.
  Regression testing was carried out by the entire group before we pushed to Github. If a member of the team was developing a feature, before this feature was to be implemented they would be required to carry

out a test to ensure this feature did not alter a preexisting component. Regression testing resulted in our application being more efficient.

- **Stress Testing** is used to determine the stability of an application by testing beyond normal operational capacity.
  Stress testing was vital in the development of our application to ensure our application did not slow down when put under stress. This was best displayed when loading large amounts of images. Without stress testing and experiments our application would not be as efficient as it is today.

- **User Acceptance Testing** is carried out on users to test the software to validate that it is performing according to the required real-life scenarios.
  We carried out this testing by allowing our friends to demo the website. This test was crucial to understand if our website was up to standard of other social media platforms. From this testing phase we learned more about how our website behaves under control of end users. In doing so we were able to adjust and tweak minor details to ensure our website met a high standard.

# Chapter 4

# Technology Review

The development stack that we chose for this project was the M.E.R.N stack. This development stack consists of MongoDB, Express.js, React,js and Node.js. We chose this stack as it was best way for us to spread the work load throughout the group as it touches all areas of website development such as front-end, back-end and databases. The M.E.R.N stack is a highly used development stack within industry and across many well respected applications we use today. React was the best choice JavaScript library for us to use as it is highly documented online and is backed by the world leading social media company Facebook. We were able to use many resources throughout the development cycle such as ReactJS Documentation and StackOverflow.

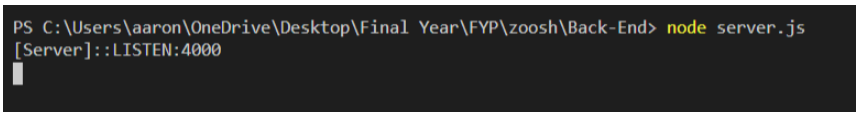
## 4.1 ReactJS

React has become a leading member of the JavaScript libraries which are extremely popular today. The main uses of React are to allow developers to created a large scale application in a fast, scalable and simple way. React is considered one of the easiest JavaScript libraries to learn as to start developing you only need basic knowledge of HTML and CSS, there are no limits when it comes to React as they are constantly updating their library and exploring new techniques for developers to utilise. React uses a virtual DOM, this is similar to the browser DOM but it is stored in memory. When the render method gets called changes are made to the virtual DOM first. A diff() algorithm is used to compare the changes and updates. If there are changes they are only shown in the browser DOM, this makes Reacts performance second to none [1]. Comparing this to AngularJS which uses a regular DOM the performance of React is greater than Angular because of

the virtual DOM [2]

## 4.2 NodeJS

NodeJS has been extremely important in the functionality of our website as it handles all of our server environment, without this our website would not function as no data would be loaded in and users would not be able to access the website. Node handles all of our Create, Open, Read, Delete and Update functionality. Node handles all of our form data and passes it to the correct collections within our MongoDB database.




## 4.3 MongoDB

MongoDB is an extremely simple out of the box database which is also highly used today. The ability to simply store data within collections inside of a MongoDB database and to receive the data within your application allows developers to move quickly. MongoDB has a user friendly interactive database where you can easily examine the data which is being passed and stored inside of your collections. After reviewing a paper found on the International Journal of Engineering Research and Technology MongoDB suited the needs of our application [3]. With our website being data heavy scalability was an important feature we looked for in a database and MongoDB provides that. The more users that used our website we could handle as the database can be scaled easily. MongoDB also provides us with dynamic queries. This was vital in the development phase as we could cherry pick data that we needed at a time. This also increased performance of the website. A social website that is data heavy needs to be able to back up data in case of a crash or malfunction. MongoDB provides us with the ability to easily replicate data in case such incident may occur with its high availability and replication. So if a users data was lost or corrupted we would have a backup of that data and simply revert the users data. With our website holding numerous object arrays MongoDB's json format was perfect for exploring our database. MongoDB's document oriented storage helped save time during the developing process[4]. Having a flexible schema provided us with the opportunity

to swap and change names and values in a short space of time if necessary. Throughout our development process we added, removed and changed collections in our database, using a NoSQL database made this efficient to do.

## 4.4 Express

Express is a back-end web application framework for Nodejs. Instead of writing a full web server by hand on Node, developers use Express to greatly simplify the creation of a web server. There's no need to continuously repeat the same code, as you would with the Node.js HTTP module.

## 4.5 Styling and Bootstrapping

We have followed all of the community and industry standards in regards to styling and bootstrapping our application. We have looked in detail to the correct colorways and decided to use 'Blue' as our main color scheme as psychologically it is considered as the worlds favourite color. Colors are an extremely important factor to a websites design especially if it is a social media platform where you want your users to be engaged with the material that they follow and read [5]. If the color scheme of the website does not match the website, the conversion rate will begin to decrease as people do not feel as engaged as they would on other platforms that understand the psychology of the user and what the user wants.

## 4.6 Simplicity

From the very beginning of the development process we have always kept simplicity in mind. This way we limited the website to a specific amount of features and functionalities and this also stream lines with the Front-End design of the application. Our website implements each feature without overloading the user with information.

## 4.7 Docker

Docker is a powerful tool that allows for many different advantages when deploying software. We utilized Docker later on in our development when we realized our GitHub instructions for installing and running the project were getting rather complicated. As the user had to navigate through multiple

directories, and run npm install in frontend and backend folders, along with running both the client and server on two command prompt instances. After research, we found that the containerization of the application was the best route to follow, as this creates:

- Consistency - A consistent, isolated environment. Everything remains the same regardless of where the app is deployed, resulting in massive productivity: less time debugging and more time launching new features and functionality for users.
- Mobility - The ability to be deployed anywhere, free of development environment limitations.
- Simplicity - The creation of two Dockerfiles for both frontend and backend allowed for a docker-compose.yml file that runs both Dockerfiles simultaneously, which installs all of the necessary dependencies for the user, and then subsequently runs the client and server all in the one command prompt.

It is of utmost importance that Docker is performant, as server virtualization is broadly used in IT enterprises. Virtualization allows for the running of different services of operating systems on the cloud. To be able to host microservice applications, which consists of different operations performed by smaller individual deployed services, a low-overhead virtualization technology is needed. Docker fills in this void, as it is lightweight and performant, not to mention open-source technology. A performance evaluation performed by Amit M Potdar et al, compared multiple facets of testing to evaluate the performance of Docker vs a Virtual Machine (VM). One such test was the *Maximum Prime Number Operation*, which determines the time taken to calculate the maximum prime number. The maximum prime number was taken to be 50'000. Using 4 thread operations, the VM calculated the prime number in about 37 seconds. Comparatively, the Docker machine completed the operation in just under 20 seconds. This was due to the hypervisor in the virtual machine. This wasn't the only area tested, however, as both machine were compared with *zip-compression tests*, *memory performance*, *disk i/o performance*, *load testing*, and more. Out of all of the tests, Docker performed significantly better than the virtual machine, with some tests Docker performing more than twice as fast, if not more than the VM[6].

## 4.8 Github

GitHub is an extremely useful tool when working in a collaborative environment. GitHub is a version control system, that automatically handles the merging of multiple local code bases into one cloud version of the code. GitHub shows you the exact differences and changes that were made by each person collaborating on the repository, which is what makes it so useful for tracking the input of each and every team member. Containing millions of repositories, GitHub has grown massively in popularity within the development community. This has also led to some issues, a software engineering conference held in India highlighted that, as open source contributions increase greatly, which are in the form of issues that report bugs or adding additional features, this leads to an increasing challenge for integrators, as the number of concurrent issues reported by developers as users greatly exceeds the amount that the number of core contributors can handle[7]. In our own final year project, we initially created a GitHub Organization, which allowed us to have multiple repositories, for each aspect of the final year project.

As GitHub is a breeding ground for open-source projects, there may need to be a Code of Conduct according to a research article in the ACM digital library, which states that "the rapid growth of open source software necessitates a deeper understanding of moderation and governance methods currently used within these projects."[8]. This study conducted a qualitative analysis of a random sample of GitHub issues, and found that codes of conduct are utilized both proactively and reactively to control community behaviour. Though, too much moderation can result in mass community backlash. The control must be balanced, by discouraging potentially offensive speech, and encouraging inclusive participation.

Although GitHub is a massively open-source website that contains millions of repositories that can facilitate code re-use, common misconceptions about the licensing of open-source software can cause problems. According to another research-article in the ACM digital library, where they compiled an extensive library of java projects from GitHub, out of 94 evaluated licenses in files and projects, they discovered that 29.6% of them may have been involved in code-borrowing, and 9.4% of them could potentially violate original licenses[9].

## 4.9 Heroku

## 4.10 Stack Comparison: MEAN vs MERN

MEAN vs MERN is a conversation that is becoming increasingly popular in the development community. With social media giant Facebook being the main developers of React, it is no doubt taking over as the main Javascript Library for building user interfaces or UI components. The only real difference of MEAN vs MERN is React vs Angular. So, in a comparitive analysis [10] of MEAN stack vs MERN stack, they compared React and Angular under the following headings: *Performance*, *Architecture*, *Third-Party Libraries*, and *Trends*.

**Performance**

In terms of performance, React is definitely better for performance reasons. Angular implements two way data binding and a digest cycle, this enables synchronization with the underlying data layer. This is extremely costly when hundreds of data items are updated dynamically. React has better state control, where change is detected using a unidirectional flow, and works better when hundreds of thousands of records need to be rendered and updated. React is also easier and more intuitive when handling events.

**Architecture**

Comparing the architecture of React and Angular is apples and oranges. React is a JavaScript library that makes UI rendering a breeze. And Angular is a JavaScript framework. As a framework, Angular enforces MVC design, which creates better organized code. And React is more flexible, leaving it to the developer to organize their code, although this may potentially make it harder to maintain.

**Third-Party Libraries**

In comparison to Angular, React needs support from Third-Party libraries to enable proper functionality. For example, to make HTTP requests, React needs to import the Axios module. Angular has a built-in HTTP service wrapper that makes HTTP requests a breeze.

**Trends**

According to Google Search Trends, Angular was initially in trend up until about 2018, React and Angular were fairly even in terms of popularity. After that though, React has taken over as the more popular frontend design framework. This is seen in figure 4.1.

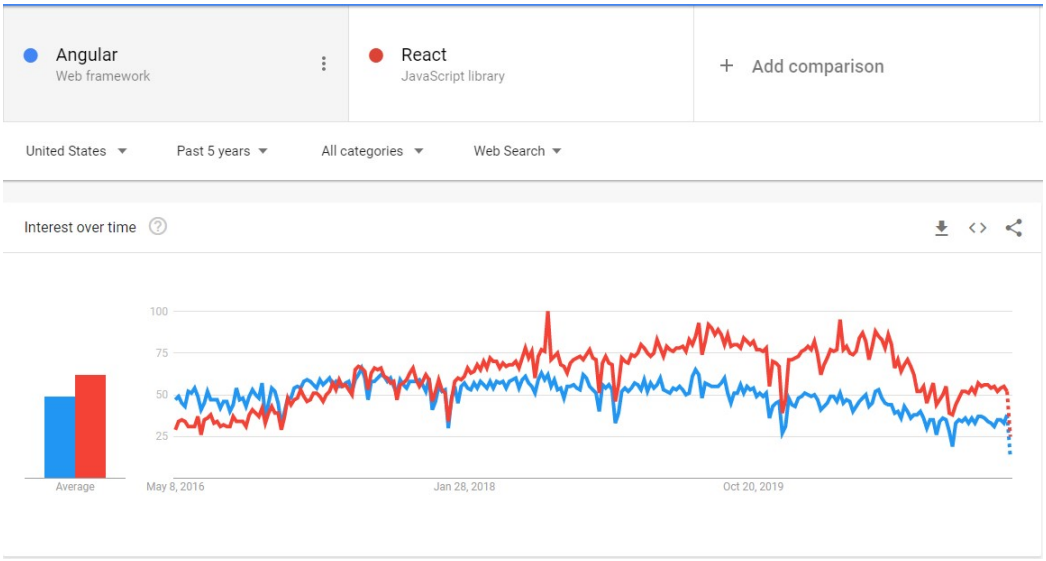



Figure 4.1: React vs Angular

# Chapter 5

# System Design

The structure which we have implemented into our project was by dividing up the Back-End and Client (containing Front-End components) into separate folders. This way we could easily distinguish which areas we were working on as both folders contain a large amount of scripts which relate to their desired areas. We decided to structure our project this way as from our research this seemed to be the most standard and mainstream structure that is being used within the industry. Inside of each folder the command 'npm install' must be used to install all the project dependencies, previously we had the Server and Client combined together within the same folder. This proved to be quite messy and meant the amount of dependencies being installed at the same time would slow down the process of development. By dividing them into separate folders it meant the members working on Back-End content did not need to install dependencies which were being used by the members working on the Front-End components.

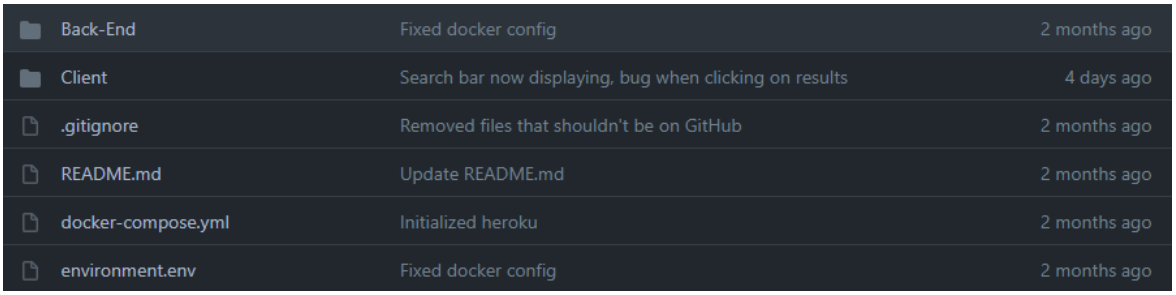



Figure 5.1: Project Github Structure

**Class Components**

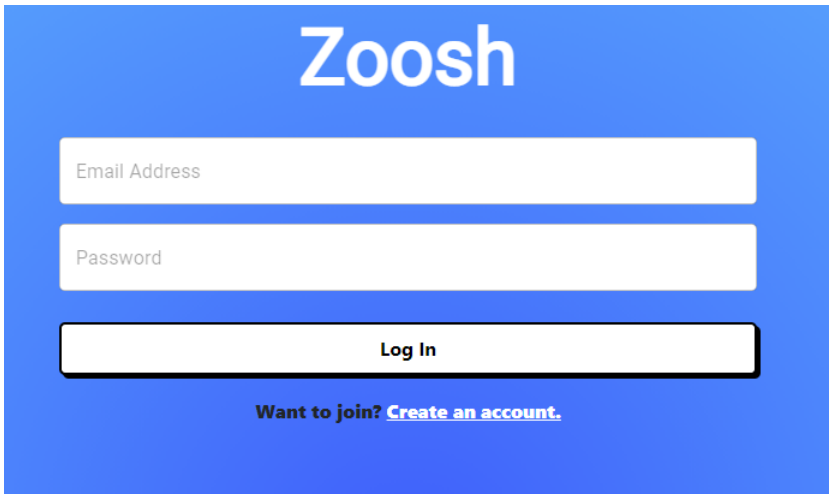
All of our JavaScript files have been implemented using class components. The reason as to why we chose Class components instead of Functional com-

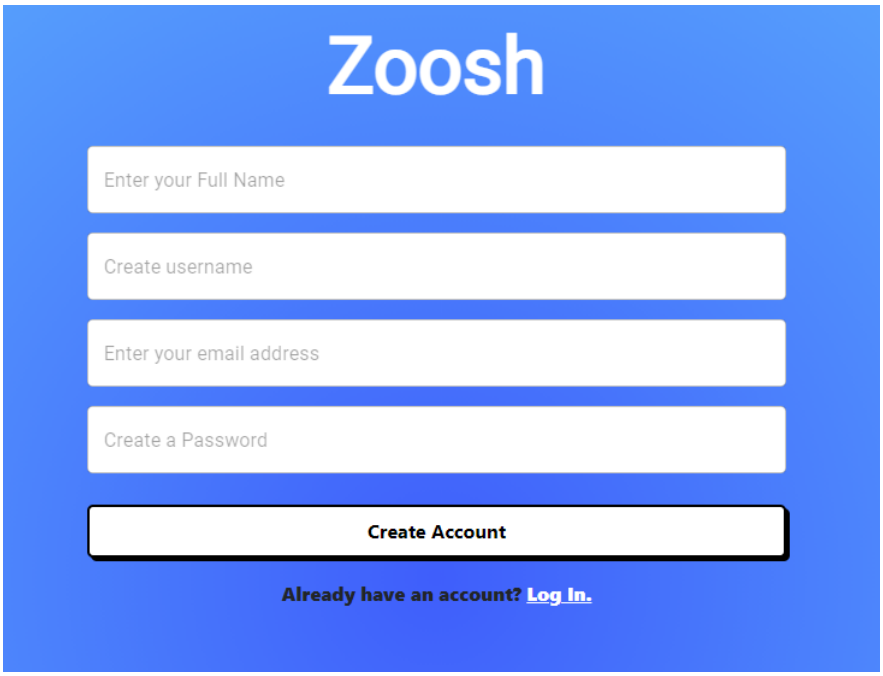
ponents was because we were dealing with many state changes within our application and needed to utilise the support and logic that class components provide. The main differences between the two types of components are their syntax, Class components were the most familiar to us as it has a similar structure to Java.

## 5.1 Frontend

### 5.1.1 Forms

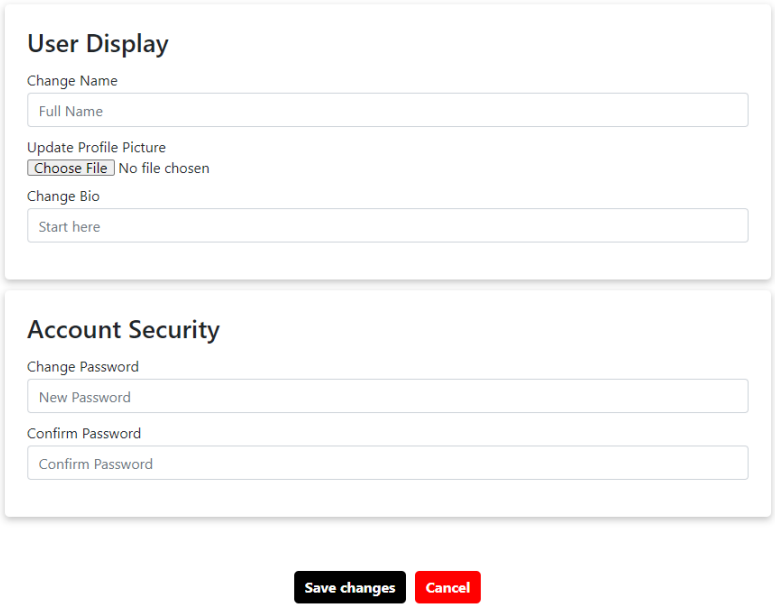
Forms within our application have been created in a simple and easy to use manner. Each text field within our forms are easily understood and do not confuse the user on what it is they are filling out. We decided to make our Login and Register pages very clean and simple with not a whole lot of noise or distraction surrounding it.





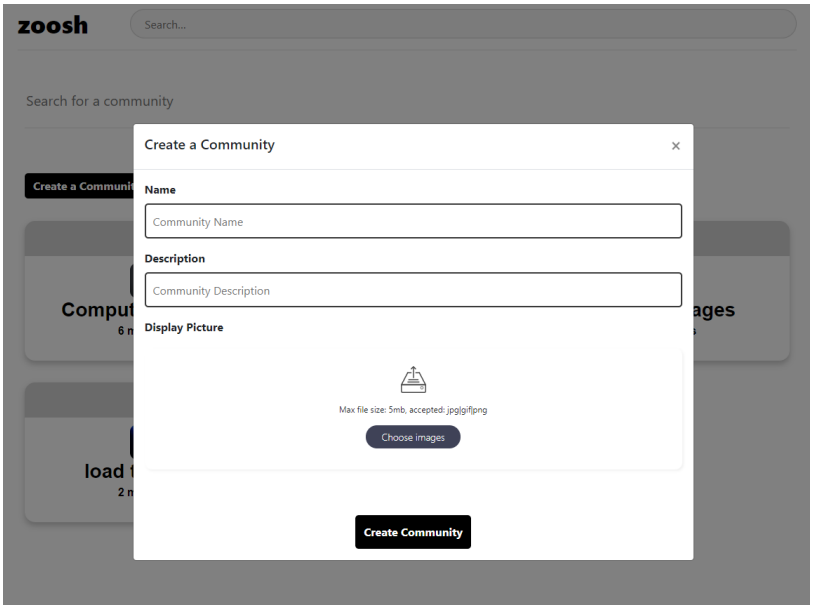

**Edit Profile Form**

When a user has navigated to their account they will be able to edit their profile, the editing options have been limited to Name, Bio, Profile Picture and Password. This form is simple and clean with no confusing aspects within the user interface.



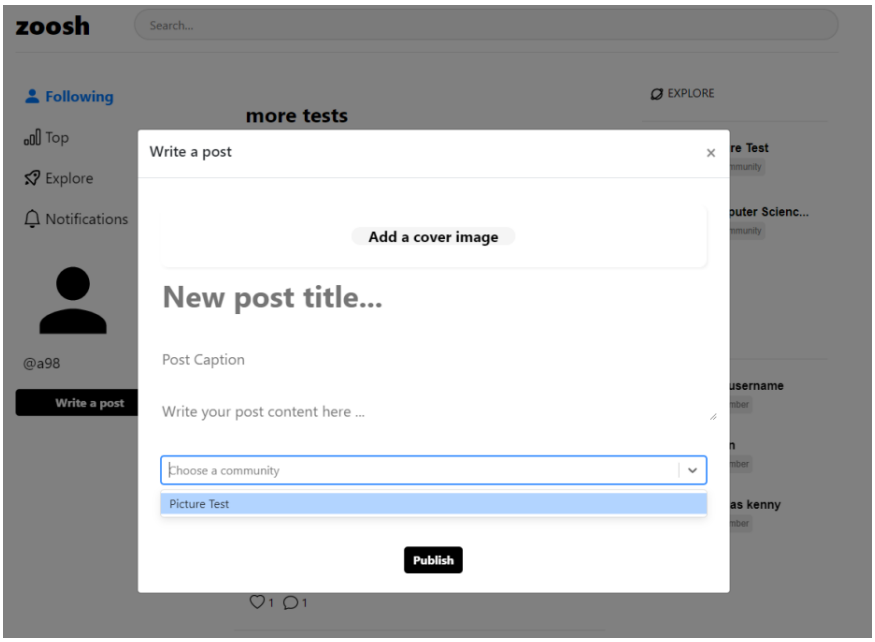
**Create a Community**

If a user wishes to create a community they must navigate to the 'Explore' page for communities and press 'Create Community'. This action will then lead to a modal pop up which will be a form that the user must fill out to create the community. The form is simple and consists of a community Name, Description, Profile Picture.




**Create a Post**

If a user wishes to create a post they can click the button 'Write a Post' on the feed pages and another modal pop up form will appear prompting the user to fill out the required sections for a post. A post must consist of a heading so that users will get a brief insight to what the post is about on the feed pages.

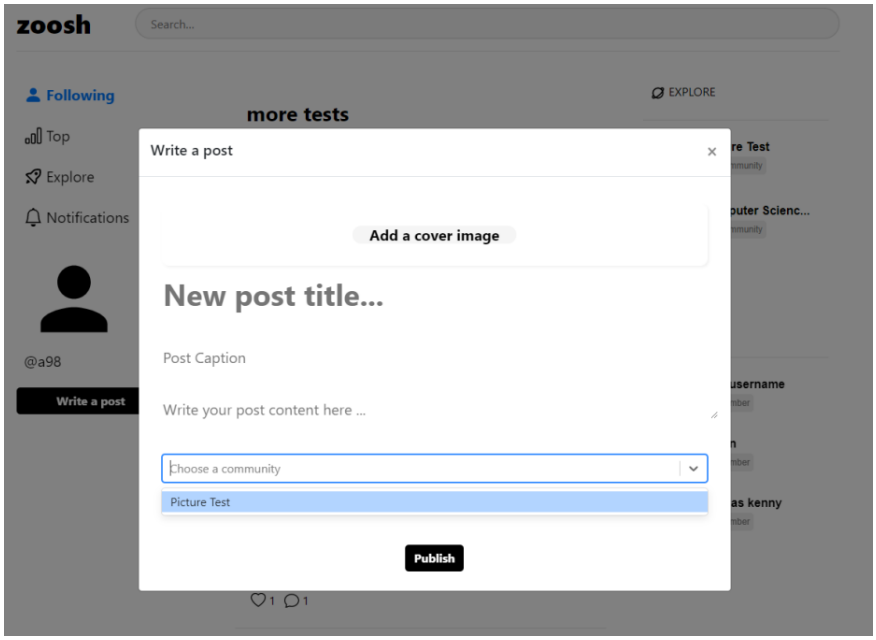



We are storing each community inside of its own collection within our MongoDB database. Each community consists of its general information and a users array to store the members of the community.


```
_id: ObjectId("5fac23d6130cbd06d803fb8f")
> users: Array
name: "Computer Science"
college: "GMIT"
category: "Technology"
address: "GMIT GALWAY"
private: false
score: 353
__v: 0
```

**Create a Post**

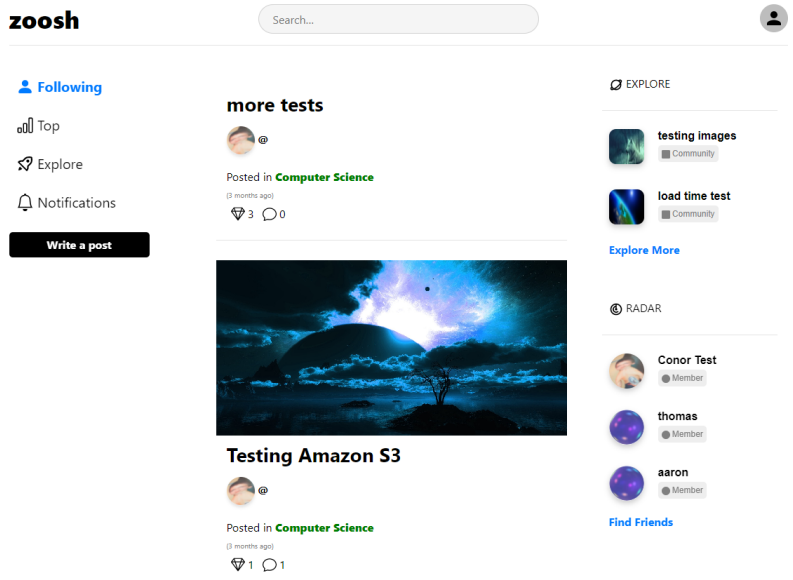
If a user wishes to create a post they can click the button 'Write a Post' on the feed pages and another modal pop up form will appear prompting the user to fill out the required sections for a post. A post must consist of a heading so that users will get a brief insight to what the post is about on the feed pages.

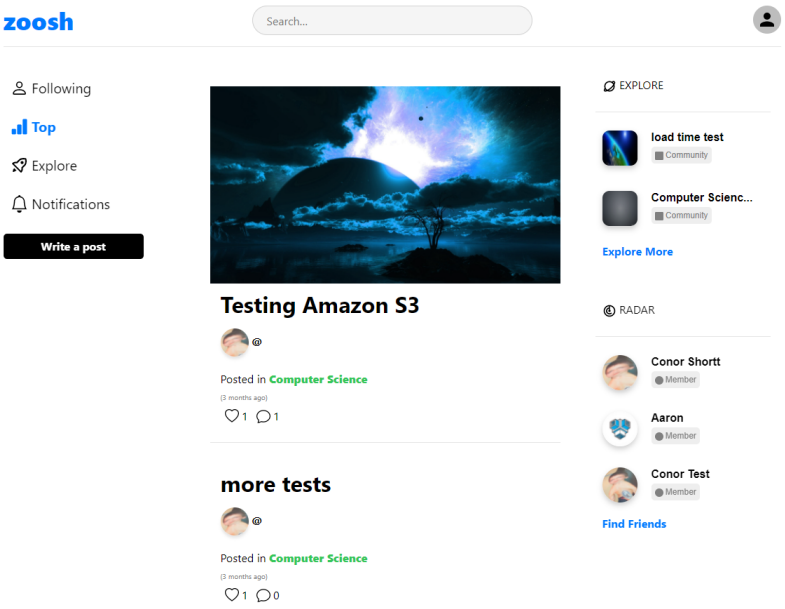



We are storing each post inside of its own collection within our MongoDB database. Each post consists of its the posting users id, a comments array to handle communication within the unique post, the communities id which it was posted under, a likes counter, the general post details such as the title, content, thumbnail and timestamps to determine when the post was created.

### 5.1.2 Displaying Lists and Feeds

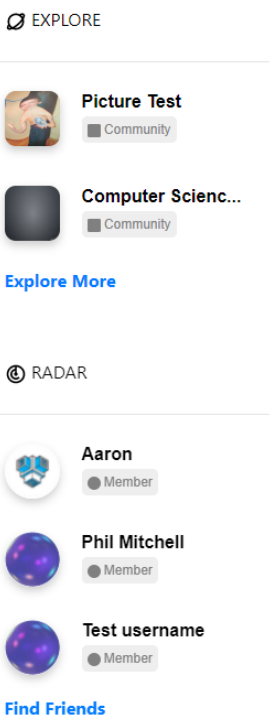
The feeds are divided into two sections, the default section is to display the posts of users that you follow and the second section will display the top posts within the website. Below is an example of both screens that will appear for users when they navigate throughout both feeds.



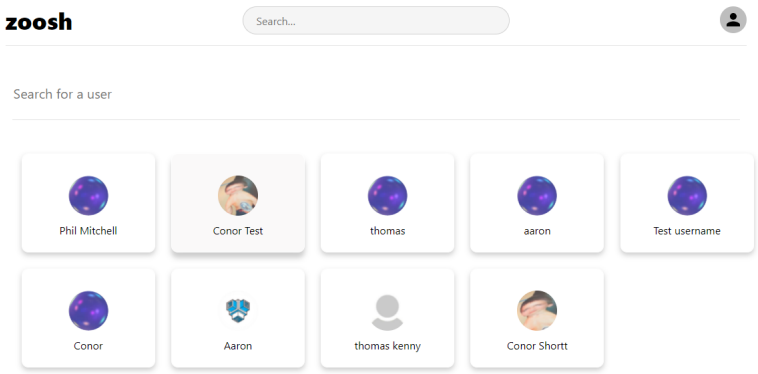



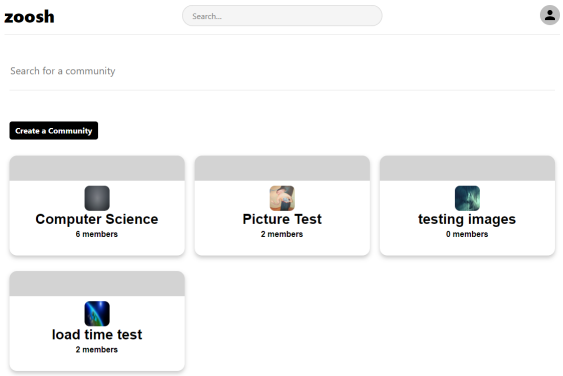
**Lists**

The displaying of lists within our application allowed us to re-use a lot of components which was very important to stream line our projects architecture. Within our application there are recommended users and communities lists which are being randomly displayed from the entire database of users. Communities are displayed as round edged square and they will have the 'community' tag display under their profile, if a community page does not have a profile picture it will default to a grey background. Users avatars are displayed as a circle and they have a 'member' tag beneath their profile, and if a user doesn't have a profile picture they receive a default purple image.




Another example of lists which we have implemented into our application is displaying users and communities within the database. These two lists have been separated into two pages and you can search the entire database for a user or community you wish to follow or join.




The two lists will filter as you type to the correct search result and you can navigate to their pages through their profiles card. Upon initially loading the communities or user list pages, skeleton grey loading icons are displayed in place of the communities. Once the communities are retrieved from the database, the state is updated and the communities are displayed in place of the skeleton icons. The communities list is looped through in the React render function by utilizing the map function that is common to Javascript. These lists are randomized each time the user enters the communities list by using the .shuffle method.

### 5.1.3 Likes and Comments

Early on when designing our website we wanted to incorporate likes and comments. Researching websites we knew this was a feature that is integral with a social media website. To set up the comments and likes we added an array for comments and a variable for likes. We tested having a collection for comments to see if would increase performance but it had no effect. With

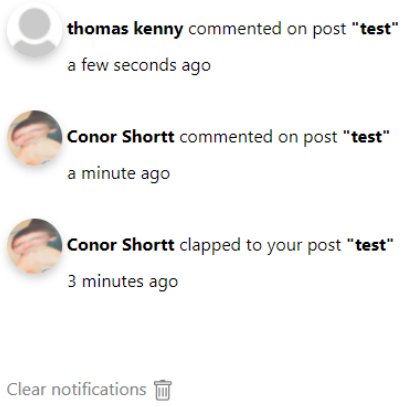
the test carried out we went forward with the array of comments. When a user adds a comment it posts an object to the comments array as shown.


We can then easily load these comments in order of creation on each discussion page. Since we have the user id's stored in the comment we could then create a link to that persons profile, linking each comment to a user.

When a user likes a discussion it will update the likes in the discussion and then display the current amount of likes. When a user first loads on to a discussion it checks to see if you have already liked it or not. If you have liked it the clap symbol will change to a dark color indicating it's being liked. If a user clicks it again it will unlike the post and will minus one from the current likes.

### 5.1.4 Notifications

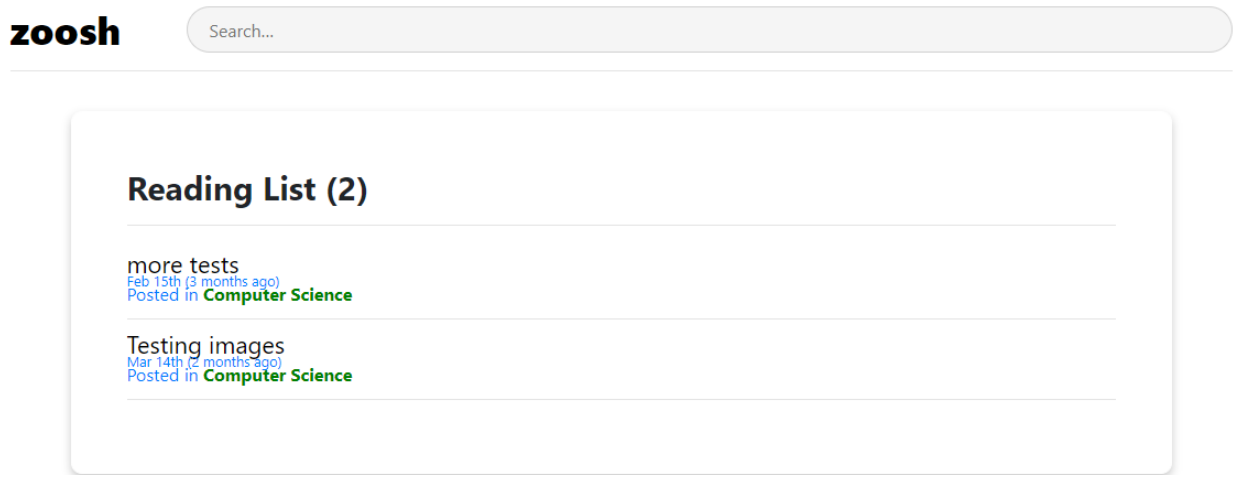
Notifications was one of the last components to be implemented. Using a social media website notifications are an essential as you will know when somebody has liked your post or made a comment. This was a feature we wanted to incorporate from the beginning. For this feature to work we needed to have the foundation of users being able to like and comment under a post. When these features were implemented we drew up a plan on a whiteboard of how we were going to tackle this feature. It would be relatively simple, when a user makes a comment/like on a post we would send that persons id,the user who created the posts id, the discussion id, and a message like "Liked your post" to the notifications collection. Once the notifications were in the database we could simple grab the data via the id and display this on a persons notifications page as shown here.




It was important to have the function of clearing the notifications for efficiency as if the notifications built up it would reduce performance. These notifications are also given a timestamp of when they were created, this is done so we can order them from newest to oldest. The notifications feature underwent extensive testing to ensure it would be efficient as possible. Originally we had the notifications be an array under each user. As with each new notification the user collection would grow to a point where it slowed down load times. This is when we decided to redesign the structure and create a new collection for notifications, in doing so only a handful of records are being loaded in at a time increasing performance.

### 5.1.5 Reading List

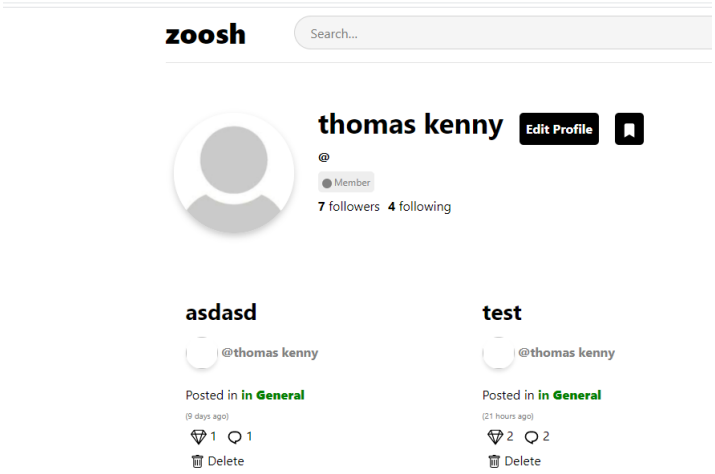
This was a feature we thought early on would be a feature to complement the website. When a user goes to a discussion they can click a button to add to their reading list. In doing so they have the opportunity to view this post at a later time.The the user adds a discussion to their reading list it's only saving the id of the discussion so it works at an optimal speed. When we are retrieving the data we can use the id to search for the discussion. The number displayed is simply counting the size of the array. A view of the reading list can be seen below.




To remove the a post from the discussion list they can go back to the post and click on the button again to remove it. Once they have removed it from the list a message will appear confirming you have removed it.

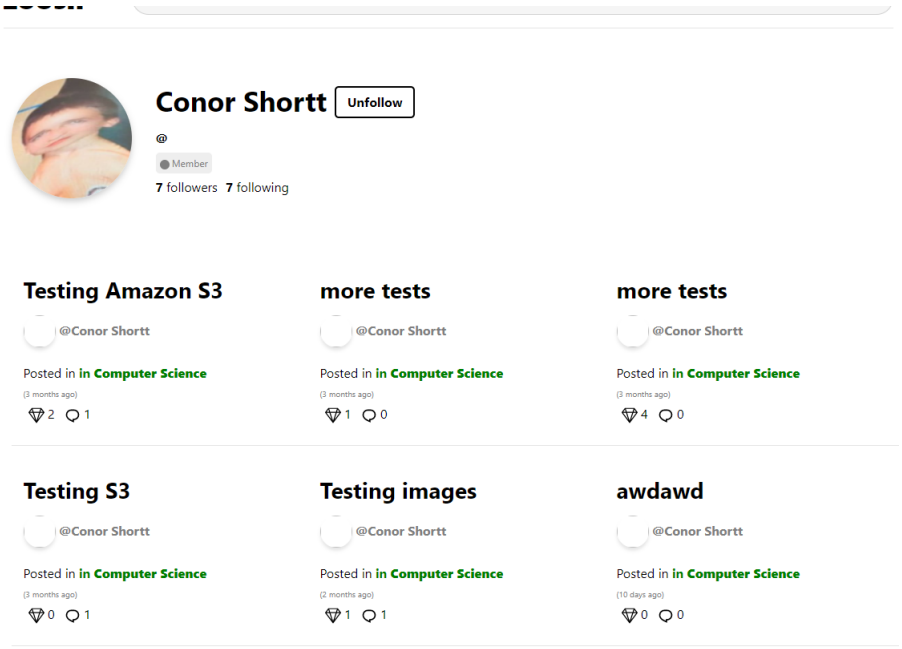
### 5.1.6 Profiles

We wanted the website to feel unique to each user, this is why profiles are so important. When a user visits their own profile they will see a some option only they can see for example, deleting their posts and editing their profile. This is done through a check before the page loads in. When a user is logged in the id is saved to local storage, this id is then compared to that of the profile page you are on. An example of a profile page is shown below.



Each post contains the user id of the person who created it, this is done so we can then show them on the profile page.

If you view another user's profile you will get the option to follow them or view their posts. The follow/unfollow was a feature we felt at the start we needed. All social media platforms have some type of follow/unfollow whether it be adding a friend or making a connection. In our website this works when a user clicks the follow button, that user's id will be saved to your following list and your id will be saved to their followers list. When you view a person page you follow, a check will be done to see if you follow them or not. If you do follow them the button shown will be unfollow and vice verse for someone you do not follow. An example of this is shown below.




## 5.2 Backend

### 5.2.1 Retrieving Data

Retrieving data is an important component of any data heavy website. And as social medias are notoriously data-driven, filled with interactions between users, photo sharing, commenting, and liking, the performance and layout of the back-end became an extremely important aspect of the application. The layout of the back-end was separated very soon after development began, as we noted that the server.js file became increasingly large and complex. We

separated the server components into two sub folders: routes, and models. The routes folder contains all of the routes for making requests to the server. For example, in discussions.js, the *get-discussions* [**??**] route takes in the parameters *fields*, and *limit*. The *fields* variable is used for selecting only the required fields from each document in the database. This greatly increased efficiency when making requests, as the requests weren't returning everything inside the required document. The *limit* variable is used to return only a certain number of documents from the database. There are five route files in the routes folder:

- **Discussions.js**: Contains all of the routes for the creation, reading, updating or deleting of **discussion** data.
- **Forums.js**: Contains all of the routes for the creation, reading, updating or deleting of **forums** data.
- **Notifications.js**: Contains all of the routes for the creation, reading, updating or deleting of **notification** data.
- **Societies.js**: Contains all of the routes for the creation, reading, updating or deleting of **society** data.
- **Users.js**: Contains all of the routes for the creation, reading, updating or deleting of **user** data.

```
// New get discussion query, selected fields are passed in when calling axios.get
discussions.get('/get-discussions', (req, res, next) => {

    var sort = req.query.sort;

    if(req.query.fields)
    {
        var query = DiscussionModel
        .find()
        .select(req.query.fields)
        .sort({sort: -1})
        .limit(parseInt(req.query.limit));

        query.exec(function (err, data) {
            if (err) return next(err);
            res.json({
                discussions: data
            });
        });
    } else {
        console.log("MUST PASS IN REQUIRED FIELD VARIABLES TO ROUTE:/get-discussions ");
    }

})
```

This route is accessed through use of Axios. Axios is a promise-based HTTP client for node.js and the browser. We used it to make HTTP requests to the node.js server, to retrieve data to be used in the front-end. Axios is imported using the *import axios from 'axios'* statement at the top of each React class. It is then utilized using the *axios.get(route, params)* method.

```
// Fetching the users Details
getUserDetails() {
  var user = JSON.parse(localStorage.getItem('user'));
  axios.get('http://localhost:4000/users/get-user-details', {
    params: {
      id: user._id,
      fields: 'forums societies likedPosts username pic'
    }
  })
    .then((response) => {
      this.setState({
        user: response.data.user,
        forums: response.data.user.forums,
        socs: response.data.user.societies,
        likedPosts: response.data.user.likedPosts
      })
    })
    .catch((error) => {
      console.log(error);
    });
}
```

Figure 5.2: Axios Get Request

As you can see above [5.2], the data is requested from the server through axios by calling a HTTP get request. The route corresponds to the route for getting user details as specified in the users file. The response data is then returned from the request and allocated into the respective state variables. This data is then used on the page to display if a post is liked or not by the user, the societies the user is in, etc.

The structure of data contained in the database is described in each of the model files within the models folder. There are five schemas in the models folder:

- Discussion.js - Outlines the database schema for discussions
- Forum.js - Outlines the database schema for forums
- Notification.js - Outlines the database schema for notifications
- Society.js - Outlines the database schema for societies
- User.js - Outlines the database schema for users

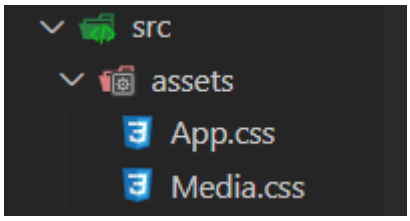
```
// create schema for the database
const DiscussionSchema = new Schema({
    user:String,
    user_id:String,
    title:{type: String , required: true},
    caption:String,
    content:String,
    time: {
        type: Date,
        default: Date.now,
    },
    society:String,
    society_id:String,
    claps:Number,
    comments:Array,
    commentsCount:Number,
    thumbnail_pic: String,
    full_pic: String,
    user_pic: String,
    likes:{type:Number , "default":0},
    slug: {
        type: String,
        required: true,
        unique: true
      },

},{
    timestamps: true,
  }
);       Find related code in zoosh
```

Figure 5.3: Discussion Schema

As you can see in figure [5.3] there are multiple variables contained within the schema for storing data. There are specific variables for discussions such as: title, caption, content, time, full picture (the large scale picture to display when viewing the full discussion page), or the thumbnail picture which is displayed in the feed to reduce cost of displaying multiple images. There are also variables which store the society that is tagged when the post in made.

## 5.3 User Interface Implementation

All of the User Interface components have been implemented using standard HTML and CSS, although for certain aspects such as layout and resizing we were able to utilise React Bootstrapping [11] to achieve the correct layout to streamline between devices. Thanks to the node package manager we can easily install this bootstrapping library and use all of its components. For any designs which cannot be achieved through bootstrapping we have created through our internal assets.




All of the icons that we have used are a free open source icon library which is provided by React [12]. These icons were extremely beneficial as in modern development of web and mobile applications icons can make or break how a user interacts with the application so making sure that the icons were used appropriately was crucial.

## 5.4 Heroku Deployment

Currently our project is being hosted using Heroku. This is the best option for our project as Heroku allows you to host up to five applications free of charge. To deploy to Heroku the application had to undergo a number of changes. To stop this from interfering with the way we were locally developing the application, we created a new branch on GitHub named 'heroku-deployment' which had the correct layout and configuration for deploying to Heroku. After following a tutorial, the server.js now had set up environment variables for dynamic ports depending on which port Heroku chose for the server. The package.json also had to have a "heroku-prebuild" script implemented, which installs client and server packages, and subsequently runs them[13].

# Chapter 6

# System Evaluation

We faced many issues when making sure that our website was robust and never failed carrying out specific functionalities. Below we discuss the problems that we encountered and how we overcame these problems. Making sure that the website is robust is extremely important for keeping users on your website and being an active user, because if it were to fail at carrying out the basic features then this would cause the user to become frustrated with the application and eventually give up on using it.

We tried out many of different experiments throughout the development of our project to improve its user experience and run-times. Between Front End and Back End we tried out many different design patterns and user interfaces that would suit our project and through trial and error found the best suitable solutions to suit our needs. We were able to utilize Googles Lighthouse performance testing tool, and also the Performance tab in Chrome DevTools to examine our load times.

We faced issues while trying to load a large amount of images on a users feed and decided to experiment with ways around this bug. We tried reducing the amount of imported packages and keep the project package small so that it would be less pressure on the website when running. This clean up didn't seem to make any improvements on our website speeds and load times, but was still a successful experiment as it tidied up our code and removed any unnecessary and unwanted external packages.

Initially, the database models were extremely inefficient as we had entire objects being stored within one another. For example, the society model was storing the entire list of users within that society as user objects which contained unnecessary information like images, names, date of birth, etc [6.1].

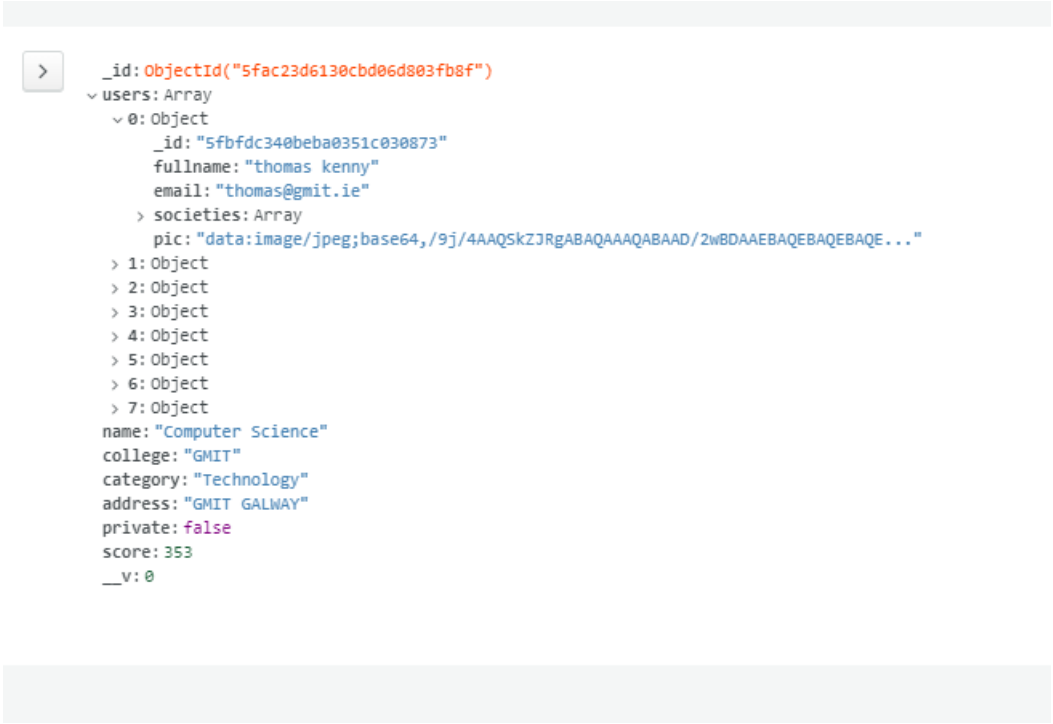



Figure 6.1: Inefficient Society Model

We then vastly reduced the amount of data shared information between collections within our MongoDB database, instead of passing all of the posting users information we only shared the ID keys which would give us full access to the details of the posting user and the posts information. Changing this structure within the communication between our collections saved us many resources and storage space within our database. This only marginally improved load times but was a success as it made our website perform at a higher speed than previously, and we learned a new solution to sharing information within our application which could be applied to many other features and functionalities.

However, even after optimizing the design of the database and making sure that no model contained unnecessary data, the website was still performing extremely poorly on Lighthouse scores [6.2]. As you can see from this figure everything except for performance was scored highly, and as the Chrome DevTools only analyze network requests and front-end components, we presumed the problem was within the structure of the database and as we had already restructured the models, we determined that the queries that were providing data to the get requests were the problem.

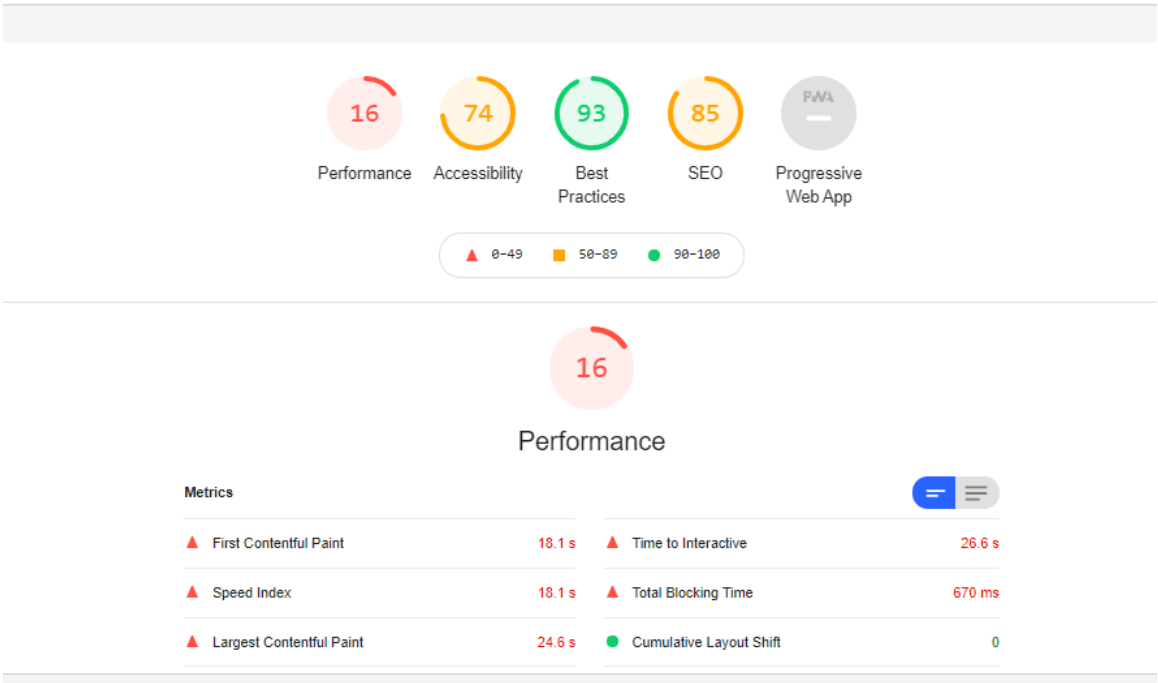



Figure 6.2: Poor Lighthouse scores

After analyzing the performance of the website using the Performance tab in Google Chrome DevTools tool, which allows you to view network requests, we evaluated that get requests for things such as the discussion feed, society list, or any request that required a large amount of data to be fetched was drastically slowing down load times [6.3]. From this figure we noted that the request for discussion feed data, which included all images, comments, title, etc, was taking significantly longer than other requests. After analyzing the queries, we noted that every field in the *Societies* collection were being returned in the request. And after some requirement analysis, we determined that only a handful of these fields were actually being displayed in the discussion feed component. We then parameterized the database queries, so that the requests could pass in a string with only the required fields for each request, and also a limit variable that limited the amount of documents returned from the query. This greatly increased the flexibility of the queries, as before we had a whole host of overloaded methods for getting different data from the database, and now we only needed a handful of queries to perform the same operations. This increased the performance of the website massively, as the discussion feed was now loading eight times faster, and all other requests were loading much faster too. This also greatly improved Lighthouse performance scores [6.4].

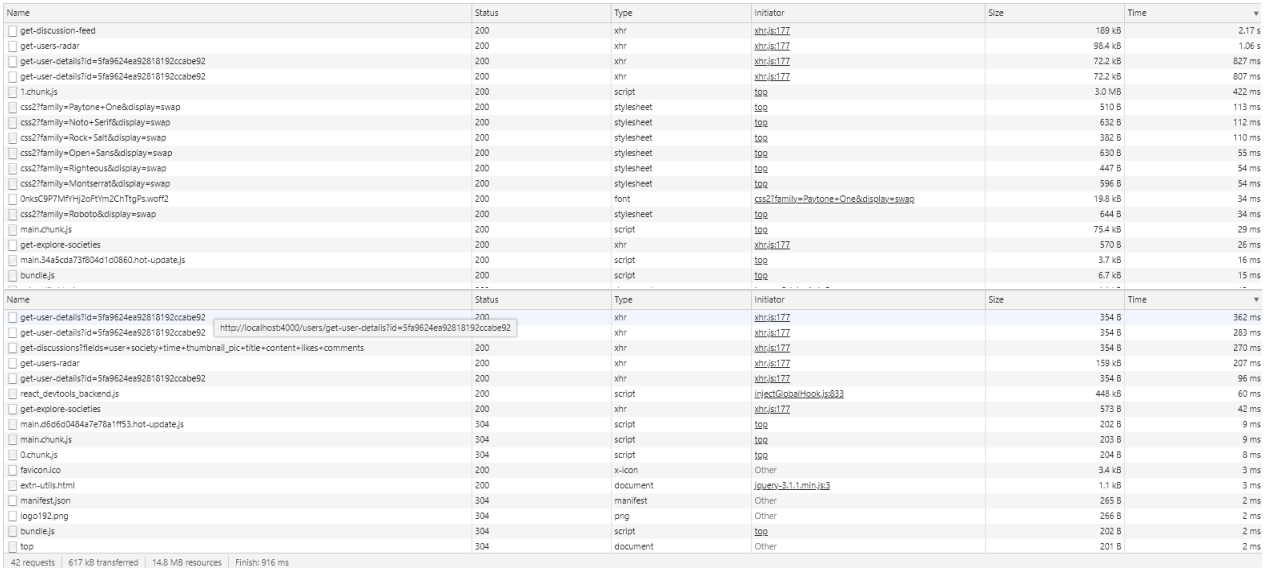



Figure 6.3: Slow Requests (Top) vs Improved Requests (Bottom)

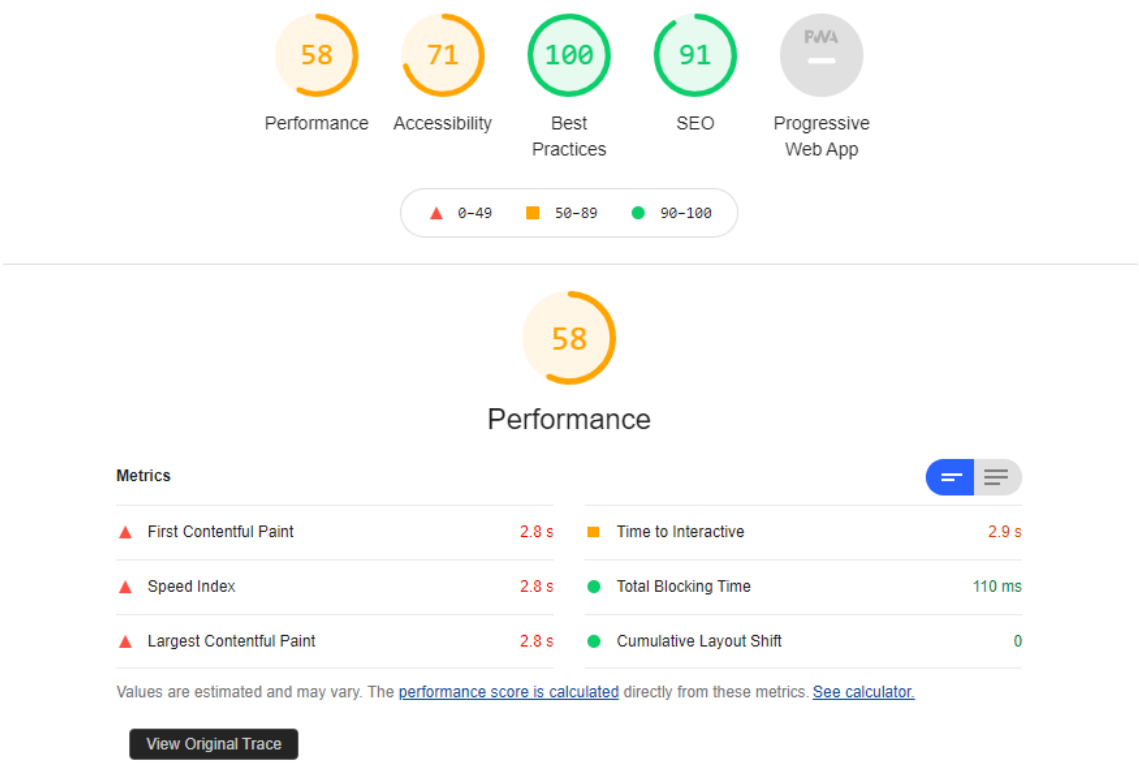



Figure 6.4: Improved Lighthouse Performance

However, this wasn't the entire issue. Although the performance was better than before, there was clearly still more problems at hand as it was still quite low comparative to other websites, such as Facebook which had a performance of 80 when analyzed with Lighthouse. Through much deliberation and many days of discussion, and after looking through many forums and articles for building fast React applications, we determined that the storage of large photos was slowing down the load times dramatically. We found that the best practices which are being used in industry is to store the large photos on an external server such as Amazon AWS and link them back to the required application. We decided to take this approach and after researching further, the consensus seemed to be that Amazon S3, a simple cloud storage service could be used to solve our large image problem. This took quite a while to implement as we had to set up a .env file which contained all of the secret keys, ip's, etc. We also had to implement a new package called Multer to handle the uploading of files to our S3 bucket. Once the image file gets uploaded to the S3 bucket the URL of it is then stored on our MongoDB database, drastically reducing load times and increase performance to an acceptable standard[6.5].

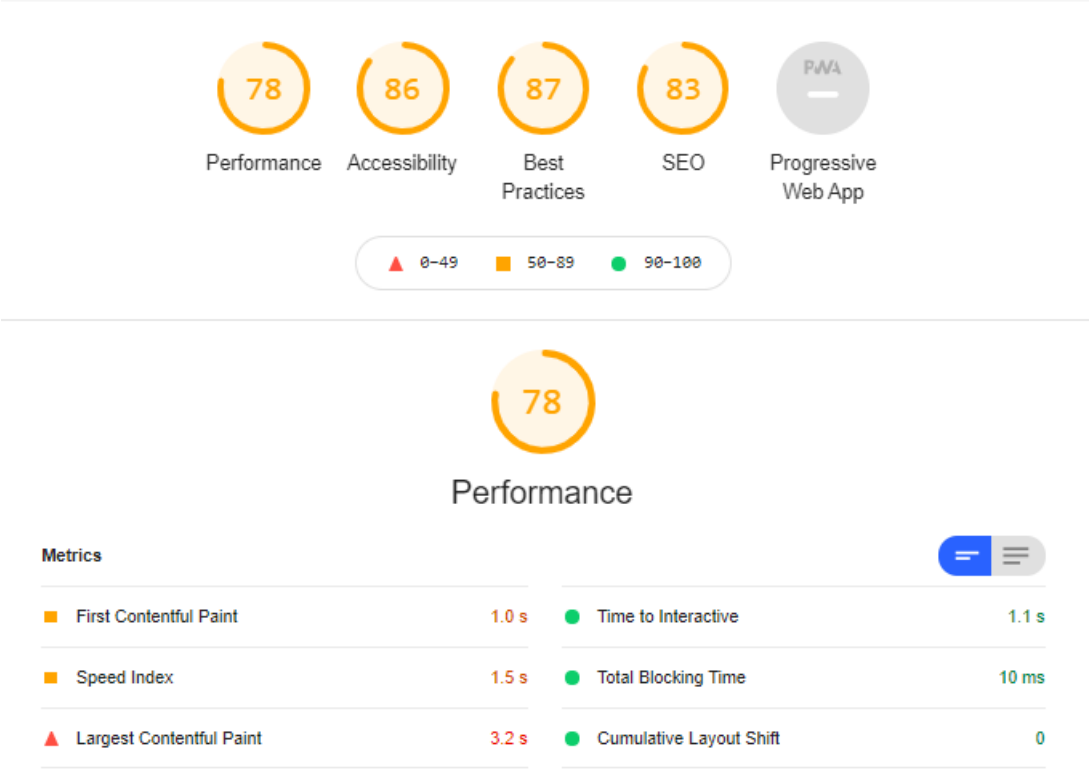



Figure 6.5: Highest Lighthouse Performance

## 6.1 Wire Framing Designs

Throughout the development of this project we tried out many different User Interfaces and Front-End designs to try and find which was the most suitable layout for our project. We used software such as Figma and Wire framing to design and test our designs which we could implement. Figma is a browser based User Interface and User Experience platform that allows you to create and prototype designs within minutes [14]. Once we had three official designs we came to a conclusion on which one would be most suitable for our project to match its features. The design has taken inspiration from existing social media platforms and has been tweaked to suit the websites functions.

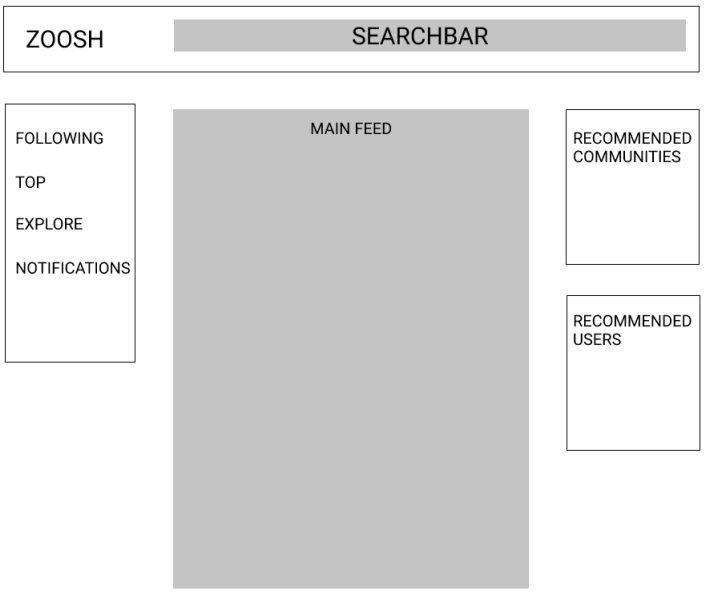



Figure 6.6: Figma Design.

# Chapter 7

# Conclusion

Overall we have achieved our goal of developing an application where users could share opinions and join communities based off of their common interests. Throughout the development of this project we thought of many features we could implement and as we were creating these features we recognized what suited the application best and which features were not necessary as they were just increasing the complexity of the application. We made sure to stay on topic of the social media aspect of the application. If we went off topic and started to add unnecessary features it would increase the difficulty and time of the development process. This is why meetings were so important as we would all be on the same page when it comes to weekly feature implementation.

## 7.1 What we enjoyed

Working on this project was an enjoyable experience as we got to work on an application which we all had fun developing. From the very beginning when we were first capable of communicating through the website rather than any other platform it made us really want to improve the functionality and fluidity of the website. We could all communicate by posting on the feed in real time and comment under each others posts on bugs or issues we were facing. It was also exciting to see this website come to flourish step by step. Week by week as new features are implemented it would be intriguing to see all these features become linked together. Getting the chance to work on an exciting project within a team meant that we were always learning together and constantly challenging ourselves to produce a better website by keeping our standards high.

## 7.2 What we learned

This was valuable experience as we got a sense of working in a professional environment. Working as a team to produce this project has helped us to develop and improve on our team working skills and problem solving skills. Looking back at our tracked progress through GitHub we feel the consistency and team work was there, with over 330 commits during the course of the academic year. Through the development process we increased our ability to carry out extensive research, we used resources such as ReactJS documentation, MongoDB documentation, Stackoverflow, Reddit and YouTube to learn techniques and learn from experts in the industry about ways to tackle problems we may be facing. This was showcased in the beginning when we were looking at other websites. It was important we didn't blatantly copy another website but take inspiration from. This process of research helped our ability of critically thinking into design features and functionality. Testing was another learning experience as we would always complete one features functionality which then led to another feature breaking. It was important that we maintained our process of testing throughout this project to end with a efficient website. This was something we got better at through the development process, at the start we didn't test as much as we would of liked to but with more bugs developing our testing became crucial. This is something we have really taken into consideration for future projects and to thoroughly test features throughout and after development and to not make too many changes within a single commit through GitHub. The experience we have gained using Agile methodologies gave us the confidence that when we work for a company we will have the necessary thought process of working in a professional environment.

## 7.3 System Evaluation Conclusion

- Create a social media platform.
- Achieve a satisfying level of optimization.
- Design a smooth and user friendly website.
- Successful testing performance across all the features.

## 7.4 What we would do differently

Looking back on our development process there will always be something we could of designed our implemented to a higher level. Of course, when we started developing the project we had very little knowledge on how to develop a social media website, as the year progressed we learned new techniques and how to tackle specific features that are considered standard within social media websites. There are a few things we could have done differently from the beginning such as, researching how to implement certain features. Although through trial and error this definitely stood to us as we learned a lot about good and bad programming practices and the correct way to implement specific features. A final thing that we would do differently and we have definitely improved on is our use of time and planning out our sprints, sometimes we would get carried away and make too much progress one week and then a lot less progress the next week and then bugs would begin to appear. If we were more disciplined with our time management we could have faced a lot less problems but this is definitely a positive experience to take from the development of this project.

To conclude, This final year project gave us an opportunity to showcase what we learned over these four years. We gained valuable insights working regularly as a team of three progressing through the development phase using agile methodologies. We were also presented with unexpected challenges whether it be data not pushing to the database or optimization, we did however overcome these obstacles finishing with a website we believe is accessible and intuitive to use that reaches a broad audience.
Ultimately, we are happy with the outcome and final version of the website and are grateful for the knowledge and experience we have gained. We would also like to thank our mentor Kevin 0'Brien for supporting us along the way if we had any issues.

# Chapter 8

# References

[15]

# Bibliography


[1] S. Aggarwal, "Modern web-development using reactjs," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 2349–7688, 2018.

[2] A. Kumar and R. K. Singh, "Comparative analysis of angularjs and reactjs," *International Journal of Latest Trends in Engineering and Technology*, vol. 7, no. 4, pp. 225–227, 2016.

[3] A. Chauhan, "A review on various aspects of mongodb databases," *International Journal of Engineering Research & Technology (IJERT)*, vol. 8, no. 05, pp. 90–92, 2019.

[4] H. Krishnan, M. Elayidom, and T. Santhanakrishnan, "Mongodb – a comparison with nosql databases," *International Journal of Scientific and Engineering Research*, vol. 7, pp. 1035–1037, 05 2016.

[5] N. Singh and S. Srivastava, "Impact of colors on the psychology of marketing—a comprehensive over view," *Management and Labour Studies*, vol. 36, no. 2, pp. 199–209, 2011.

[6] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

[7] A. B. Dhasade, A. S. M. Venigalla, and S. Chimalakonda, "Towards prioritizing github issues," (New York, NY, USA), Association for Computing Machinery, 2020.

[8] R. Li, P. Pandurangan, H. Frluckaj, and L. Dabbish, "Code of conduct conversations in open source software projects on github," vol. 5, no. CSCW1, 2021.

[9] Y. Golubev, M. Eliseeva, N. Povarov, and T. Bryksin, "A study of potential code borrowing and license violations in java projects on github," (New York, NY, USA), Association for Computing Machinery, 2020.

[10] S. Aggarwal and J. Verma, "Comparative analysis of mean stack and mern stack," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 127–32, 2018.

[11] "React bootstrap." `https://react-bootstrap.github.io/`.

[12] "React icons." `https://react-icons.github.io/react-icons/`.

[13] M. Soomro, "Deploying mern app to heroku." `https://coursework.vschool.io/deploying-mern-app-to-heroku/`.

[14] "Figma ui ux tool.." `https://www.figma.com/files/recent?fuid=899606574753017180`.

[15] A. Moran, C. Shortt, and T. Kenny, "Github repository : Zoosh.." `https://github.com/applied-project-2020/zoosh`.