
Final Year Project Title

Aaron Moran

Conor Shortt

Thomas Kenny

B.Sc.(Hons) in Software Development

MAY 7, 2021

Final Year Project

Advised by: Kevin O'Brien

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



Contents

0.1 Abstract	4
1 Introduction	5
1.1 GitHub URL	7
2 Context	9
3 Methodology	10
3.0.1 Agile Methodology	10
3.1 Testing	12
3.1.1 Test Plan	14
4 Technology Review	15
4.1 ReactJS	15
4.2 NodeJS	16
4.3 MongoDB	17
4.4 Express	17
4.5 Styling and Bootstrap	17
4.6 Simplicity	18
4.7 Docker	18
4.8 Github	19
4.9 Heroku	20
4.10 Heroku vs AWS	20
4.11 Stack Comparison: MEAN vs MERN	21
4.12 Agile vs Waterfall	23
5 System Design	27
5.1 Front-End	28
5.1.1 Forms	28
5.1.2 Displaying Lists and Feeds	33
5.1.3 Likes and Comments	35
5.1.4 Notifications	36

<i>CONTENTS</i>	3
5.1.5 Reading List	37
5.1.6 Profiles	38
5.2 Backend	39
5.2.1 Retrieving Data	39
5.3 User Interface Implementation	43
5.4 Heroku Deployment	43
6 System Evaluation	44
6.1 Wire Framing Designs	49
7 Conclusion	50
7.1 What we enjoyed	50
7.2 What we learned	51
7.3 System Evaluation Conclusion	51
7.4 What we would do differently	52
8 References	53

About this project

0.1 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. This dissertation focuses on the work carried out to design and develop this project over the course of the 20/21 academic year. In detail, we show our development process, testing and the structure we used in developing this website. For this paper, We have researched relevant papers, journals and books related to the software we have used in the implementation of this final year project. The in depth research we have conducted gave us more knowledge of the software used and helped shape our understanding of system design. The development stack we have decided to use is the MERN stack. The MERN stack consists of MongoDB, ExpressJS, ReactJS and NodeJS. By using this stack, it allows us to spread the development process between each member of the team. We chose the Agile Method for our project as it was flexible and suited our project as a couple of changes were made throughout its development. This method gave us the ability to recognize features and requirements and upon this we discussed the approach we would take to tackle the solution. Our testing would not be up to par without the use of Agile as this method provided us with structure and consistency. The main ambition of a social media website is to give a foundation for users to interact with each other. The website must have a low learning curve and be visually appealing. As a team we believe this is demonstrated with our website ***zoosh***.

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. This project was undertaken by a team which consists of three members - 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 gained valuable experience carrying out testing which will stand to us when we are working within industry we will have to carry out extensive testing on important features we may be working on.

We felt that a social media website was a project that we could all contribute to. 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, 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.

As a result of 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 make a digital platform for clubs and societies alike. 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 an 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

• 3.0.1 Agile Methodology

For the development of this application, we decided the best approach to take was to use Agile. Agile is considered the modern methodology for software development due to its flexibility. Agile is a methodology focused around iterative development and by using this method requires a team to work collaboratively and consistently. Agile is more suited to developing web applications as it encourages greater collaboration flexibility. We chose Agile over the Waterfall model as it was not suitable for the application we were developing and proved to be too rigid. 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:

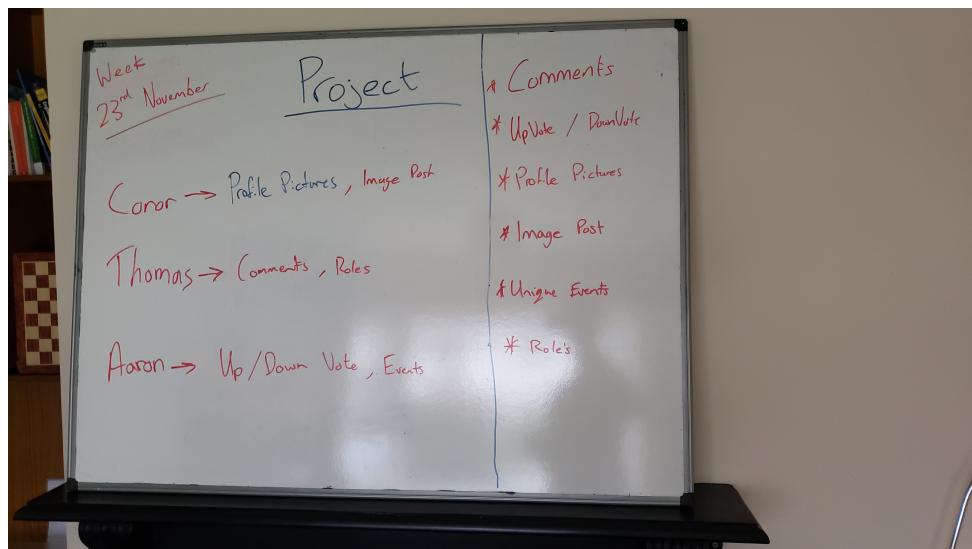
- **Functional** - Probably the most important aspect of every website. It must work quickly and as expected. Page loading times must be kept as low as possible, especially when working with large amounts of data - this was a big issue that we encountered when dealing with images.
- **Easy to Use** -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

3.1.1 Test Plan

These are the features we wrote up on our test plan.

Features to be tested:

- TC.001: Register an Account
- TC.002: Login to Website
- TC.003: Create a Post
- TC.004: Delete a Post
- TC.005: Follow a user
- TC.006: Unfollow a user
- TC.007: see followed users posts
- TC.008: Comment on a post
- TC.009: Like a post
- TC.010: Unlike a post
- TC.011: Add post to Reading List
- TC.012: Remove from Reading List
- TC.013: Check notifications
- TC.014: Create a Society
- TC.015: Check user profile

A link to our test plan can be found here. [Link to Test Plan](#)

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] React was the obvious choice for us as it suited the needs of our application. The learning curve for React is low compared to others like Angular. We carried out extensive research into MEAN vs MERN stack which can be viewed here [21](#)

4.2 NodeJS

Using NodeJS has many benefits within web and mobile development, some of the main advantages of Node are its high performance, its extremely scalable, its easy to learn and implement and has a large community to provide help and guidance for any issues you may be facing. NodeJS has been extremely important in the functionality of our website as it handles all of our server-side functionality, 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 performs all of our Create, Open, Read, Delete and Update functionality and handles all of our form data and passes it to the correct collections within our MongoDB database.

Reasons as to why we chose Node are because they have become a big player within the web application industry. Major companies such as Uber, Netflix and PayPal all implement NodeJS into their applications as it has increased their performance and simplifies Back-End programming [?]. This gave us enough confidence to choose NodeJS to handle our server-side environments.

NodeJS also provides developers with its own package manager known as 'npm'. This is the most common and most popular package manager used for web and mobile application development as it allows you to install third party libraries to speed up and help with the development process. We have utilised the node package manager within our project for many of features and functionalities, through this we were able to install third party libraries such as React Bootstrap for Front-End design and Bcrypt which is a password hashing function to keep our users passwords secure.

```
PS C:\Users\Aaron\OneDrive\Desktop\Final Year\FYP\zoosh\Back-End> node server.js
[Server]: :LISTEN:4000
```

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 Bootstrap

We have followed all of the community and industry standards in regards to styling and bootstrapping our application. Bootstrap is the most popular CSS framework which allows you to create responsive and mobile first web-

sites. Bootstrap allows you to create nice website designs in a timely manner. Its responsive CSS allows developers to create Front-End components which will resize themselves on many devices such as Mobile, Tablet, Laptop and Desktop. 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.

We used two external third party libraries to achieve specific Front-End components within the application, these libraries were React Bootstrap and Material UI. Both of these third party libraries can be installed through the node package manager and are out of the box components that do not need cascading style-sheets or any editing. These libraries are extremely useful for a fast implementation of visual components although have many drawbacks as they cannot be styled. Majority of our application has been styled using our own internal cascading style-sheets where each component can have its own class name to determine its style. This way we could design the application to how we wanted it to look and achieve the most user friendly application.

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 Heroku vs AWS

Heroku is a platform as a service (PaaS) that allows developers to deploy and host applications in the cloud. It is also one of the first cloud platforms, and has been in development since 2007. The main reason we chose to deploy

our application on Heroku was because of the simple nature of the Heroku service. It's extremely easy to configure, and there is an abundance of documentation and Stack Overflow posts relating to it. In comparison to Heroku, AWS is a secure cloud services platform that provides IaaS, PaaS, and SaaS. The configuration and deployment of a service on AWS is quite complicated in comparison. The advantage of AWS is that it can provide much higher computational demands, but as we were only developing for a very small number of users in mind, Heroku fitted our purposes perfectly. Another major factor was that Heroku offers a free deployment service, where you can deploy up to 5 applications at any one time. AWS would also generally require a dedicated DevOps person to manage and maintain the infrastructure, and as we only had three team members, this wasn't feasible. A simple benefit that we found when using Heroku was that it has an automatic deployment feature when connected to a GitHub branch. And so we created a separate branch on GitHub named *heroku-deployment*, and set up the configuration there. Whenever a new push is made to the branch, Heroku automatically re-builds and re-deploys the application.

In a comparative analysis of AWS vs Heroku for Growing Startups, they compared the two under cloud hosting services when deploying the same application, with the same region, and a concurrent user amount of 1000. According to the load test results, Heroku dynos had very inconsistent results, as every test produced a different graph. Conversely, AWS produced consistent results, responding to a lot of requests quickly and then more slowly. Heroku slowed down suddenly at large intervals, possibly due to the fact that the Heroku applications run on shared AWS EC2 instances. Heroku had significantly worse response times, with an average latency of 26ms, in stark contrast to AWS' 12ms latency. The network throughput of AWS was also quite a lot higher, with 648kb/s, in comparison to Heroku's 419kb/s[10].

Despite these findings, and with AWS clearly being superior in terms of performance to Heroku, the configuration and deployment time of an AWS application took significantly longer than the deployment of a Heroku application, which takes minutes, not hours. This is why we went with Heroku, and also because of its free capabilities.

4.11 Stack Comparison: MEAN vs MERN

MEAN vs MERN is a conversation that is becoming increasingly popular in the development community. The MEAN stack development consists of MongoDB, ExpressJS, AngularJS and NodeJS whereas the MERN stack consists of MongoDB, ExpressJS, ReactJS and NodeJS. 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 User Interface components. The only real difference of MEAN vs MERN is React vs Angular. So, in a comparative analysis [11] 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. Since we have many event handling features within our application this was the best option for us.

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 front-end design framework. This is seen in figure 4.1.

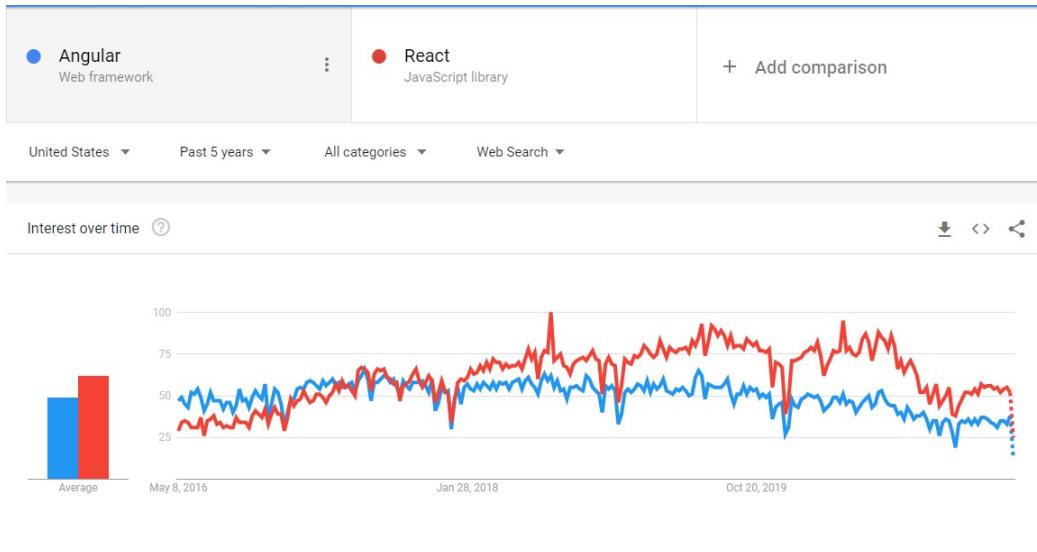


Figure 4.1: React vs Angular

This was a strong indicator for us to choose React as it is highly supported online within communities and their are endless amounts of informative resources that we could learn from to help us complete the project.

4.12 Agile vs Waterfall

Agile Method

The Agile Methodology suited us much better than Waterfall in modern terms of software development and in a team collaboration project, the reasons for this are because it is a flexible method and changes can be made to the design and code structure during the last minute of development. Although Waterfall is considered one of the most commonly used methodologies within industry, Agile is being adopted more and more within the industry [12].

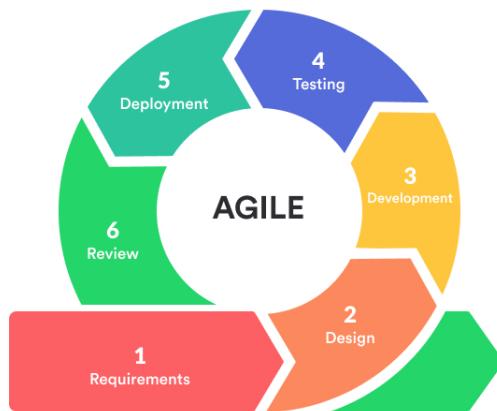


Figure 4.2: Agile Methodology
[13]

The first stage of Agile is the requirements, figuring out what features and functionalities to implement into the application and most importantly what purpose the application will serve. During this stage there will be a lot of communication with the client to ensure the project scope is understood on both ends. The design stage is extremely important as implementing a design a client does not want or like will result in repeating the entire process again, although within the Agile method designs can be changed throughout the cycle and adaptations can be made at any stage [12]. During this stage the User Interface / User Experience Designers will work on creating wireframes and mock up designs to be passed onto the programmers for development. These two teams will work together to make sure the designs are possible and that they suit the customers needs. Once the project is complete testing will commence to ensure the software is bug free. The application will then be provided to the client through a demo of the software or a live application. The client will then review the application and if they are dissatisfied the process will repeat from the very beginning. Throughout the entire cycle, Agile allows the team to make changes at any point throughout its development process. If changes need to be made this methodology allows for new ideas or designs to be brought to the table at any point of development. [12]

Waterfall Method

There are many differences between the two software development methodologies that have become more evident through our extensive research. Agile is considered the modern methodology compared to waterfall. Waterfall consists of a development process from stage to stage, the development team will only move onto the next stage once the previous stage has been fully completed. By using the waterfall methodology you are fully defining the project at the beginning and carry out each step until the project is complete without making any changes. This methodology can make the project too rigid as changes are not allowed to be made with the design so the coding and testing must meet the requirements of the solution. The name 'Waterfall' was given to this methodology as it refers to the downward flow of water as progress within this methodology is carried out in the same format [12].

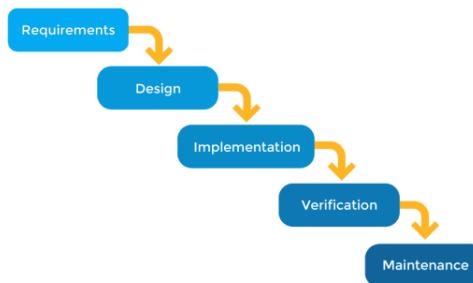


Figure 4.3: Waterfall Methodology
[12]

Waterfall is broken up into the same six stages as agile, these stages consist of Requirements, Analysis, Design, Coding, Testing and Acceptance. Once the requirements have been decided upon at the beginning then you can proceed to the next stage. Analysis is then performed on the requirements to determine the resources that are needed to tackle the project. The projects pseudocode is then written in the next stage so that the programmers know how to complete the project without coming across any design patterns that might stunt the projects progression. The next stage consists of the programmers coding the project based on the previous stages flowcharts and algorithms decided upon. It is important that each stage is correctly implemented and fully decided upon, if the previous stages before coding have failed to achieve their goal then major problems will be faced. Once the code has been complete there needs to be an extensive amount of testing to make sure that the code that has been written works without any bugs. The final stage of the waterfall method is making sure that the Client is satisfied

with the final product, in some cases where the client may not be happy with the application you have created for them the entire process must start over and return to the first stage of the methodology. This ensures that extensive meetings with the client before any development stages begin are extremely important to avoid wasting time and resources. [12]

Class Components vs Functional Components Class and functional components both hold their strengths and weaknesses. Class components handles state changes whereas functional components are just JavaScript functions that accept props as arguments and they can be reused throughout an application. There are many uses for functional components and they have become increasingly popular within the ReactJS community but we settled on class components as we could utilise its simplicity and robust structure.

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

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.

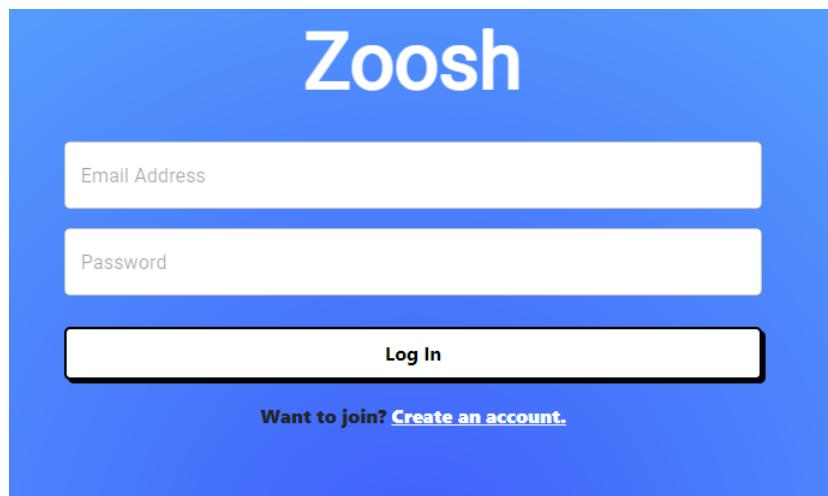
Back-End	Fixed docker config	2 months ago
Client	Search bar now displaying, bug when clicking on results	4 days ago
.gitignore	Removed files that shouldn't be on GitHub	2 months ago
README.md	Update README.md	2 months ago
docker-compose.yml	Initialized heroku	2 months ago
environment.env	Fixed docker config	2 months ago

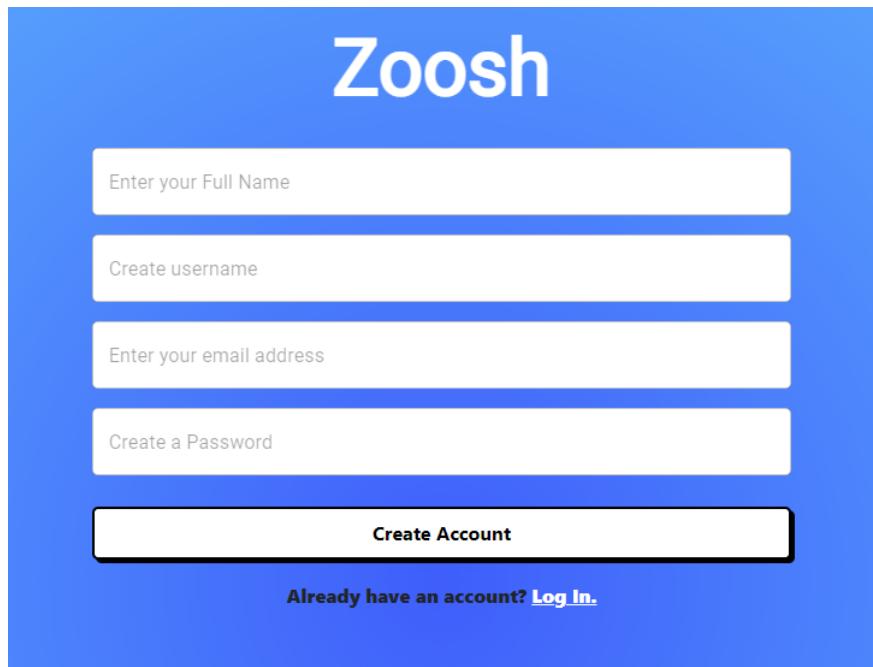
Figure 5.1: Project Github Structure

5.1 Front-End

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.





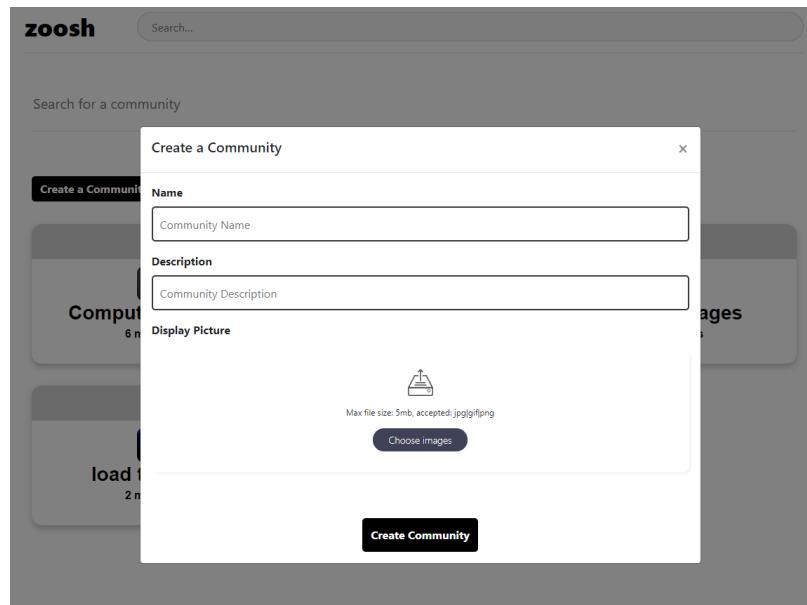
Edit Profile Form

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.

The edit profile form consists of two main sections. The first section, titled "User Display", contains fields for changing the name (with a "Full Name" input field) and updating the profile picture (with a "Choose File" button showing "No file chosen"). The second section, titled "Account Security", contains fields for changing the password (with "New Password" and "Confirm Password" input fields). At the bottom of the form are two buttons: "Save changes" (black background, white text) and "Cancel" (red background, white text).

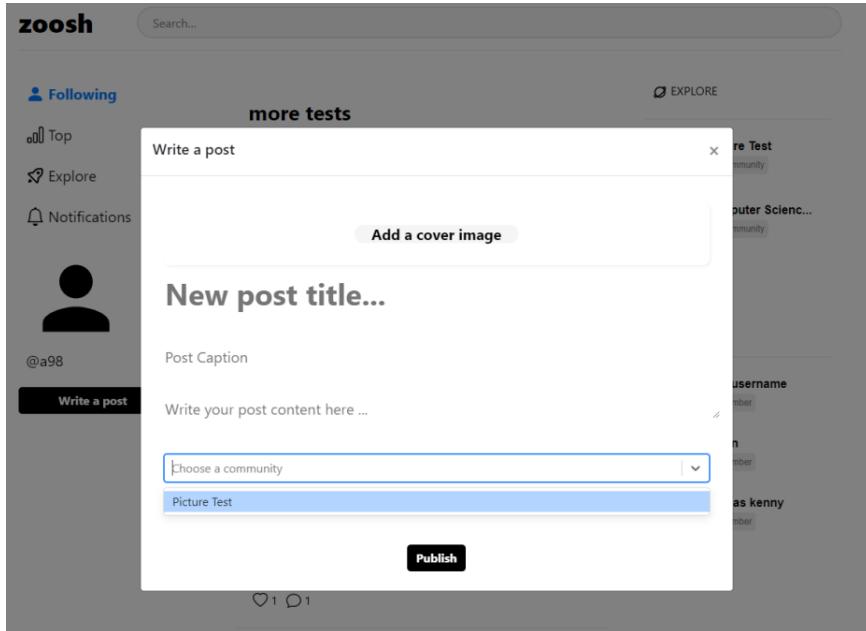
Create a Community

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.

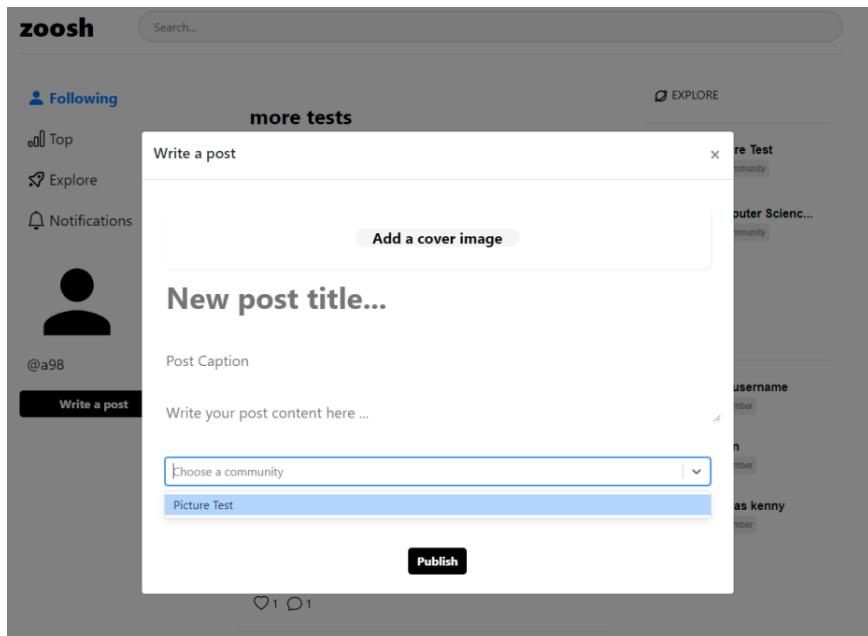


We are storing each community inside of its own collection within our MongoDB database. Each community consists of its general information and a users array to store the members of the community.

```
_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

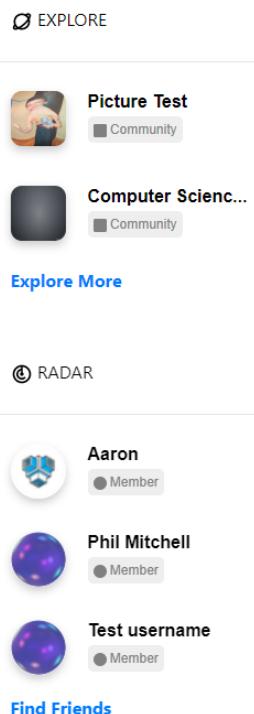
The feeds are divided into two sections, the default section is to display the posts of users that you follow and the second section will display the top posts within the website. Below is an example of both screens that will appear for users when they navigate throughout both feeds.

This screenshot shows the 'Following' feed on the zoosh app. The left sidebar includes navigation links for 'Following', 'Top', 'Explore', and 'Notifications', along with a 'Write a post' button. The main content area displays a post titled 'more tests' by a user named 'testing images'. The post is categorized under 'Computer Science' and was posted 3 months ago. It includes a thumbnail image of a landscape at night with a tree on a hill, a caption 'Testing Amazon S3', and engagement metrics (3 likes, 0 comments). To the right, there's an 'EXPLORE' section featuring other posts like 'load time test' and 'RADAR' feed, along with a 'Find Friends' section.

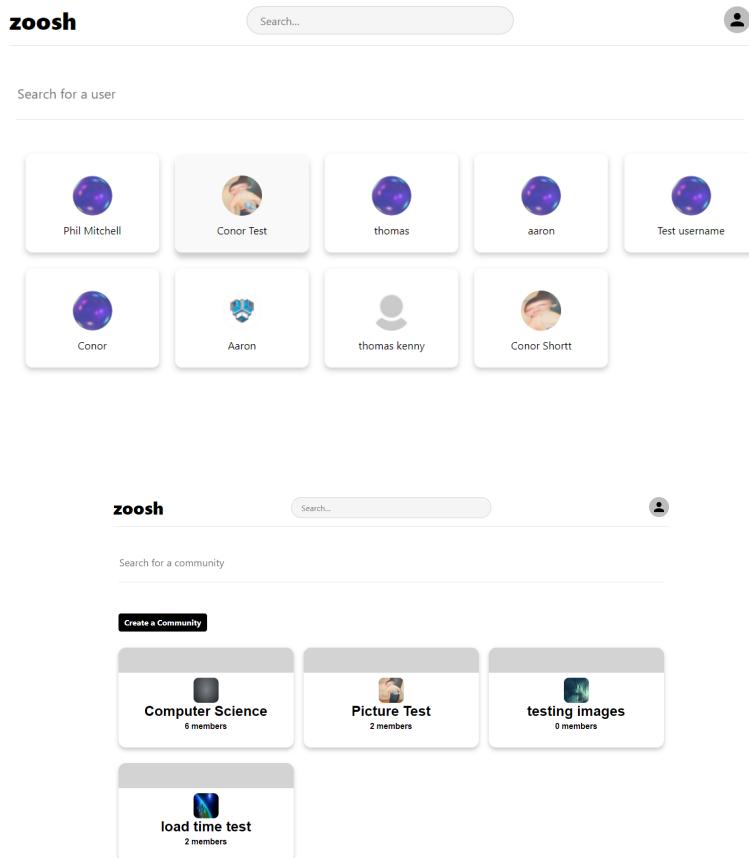
This screenshot shows the 'Top' feed on the zoosh app. The left sidebar includes navigation links for 'Following', 'Top', 'Explore', and 'Notifications', along with a 'Write a post' button. The main content area displays a post titled 'Testing Amazon S3' by a user named 'Conor Test'. The post is categorized under 'Computer Science' and was posted 3 months ago. It includes a thumbnail image of a landscape at night with a tree on a hill, a caption 'Testing Amazon S3', and engagement metrics (1 like, 1 comment). Below this, there's another section titled 'more tests' with a similar post by 'testing images'. To the right, there's an 'EXPLORE' section featuring other posts like 'load time test' and 'Computer Scienc...', along with a 'Find Friends' section.

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.

```

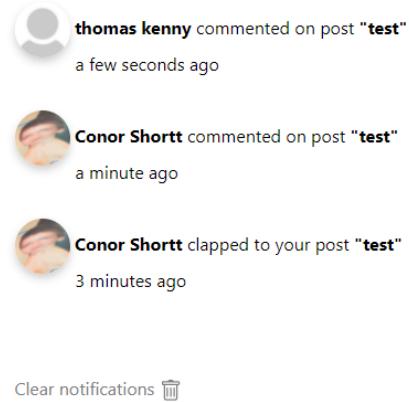
  - comments: Array
    - 0: Object
      user_id: "5fbfdc340beba0351c030873"
      user_name: "thomas kenny"
      comment: "asdasd"
      time: 1614004172935
      user_img: "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAEBAQEBAQEBAQE... "
    likes: 1
    ...
  
```

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 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.

The screenshot shows a user interface for a platform called "zoosh". At the top left is the "zoosh" logo. To its right is a search bar with the placeholder "Search...". Below the search bar is a section titled "Reading List (2)". Underneath this title are two items listed as posts:

- more tests**
Feb 15th (3 months ago)
Posted in Computer Science
- Testing images**
Mar 14th (2 months ago)
Posted in Computer Science

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.

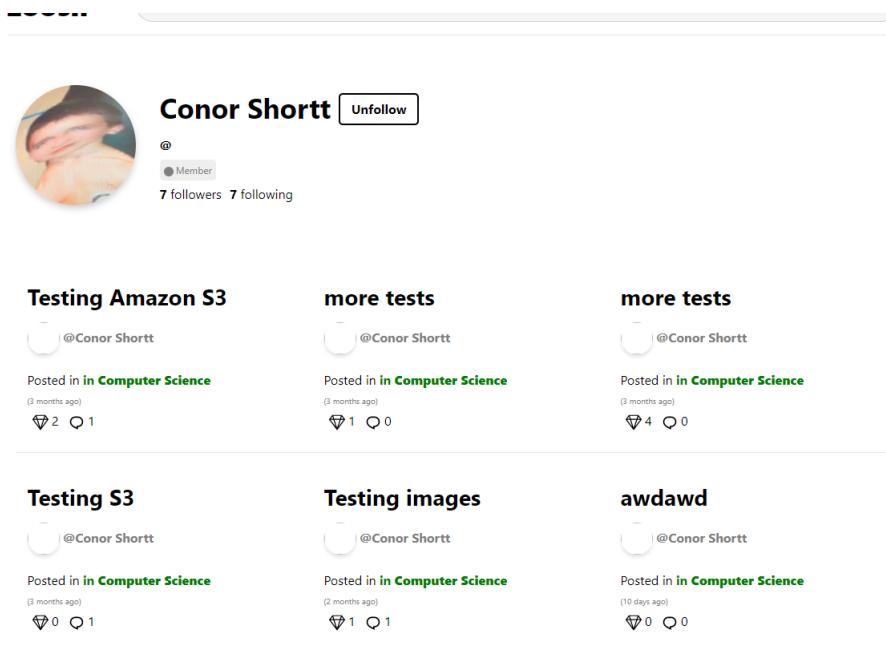
The screenshot shows a user profile page for "thomas kenny". At the top left is the "zoosh" logo. To its right is a search bar with the placeholder "Search...". Below the search bar is a circular profile picture placeholder. To the right of the picture is the username "thomas kenny", a "Edit Profile" button, and a "Bookmark" icon. Below the username is the status "@ Member". Further down, it shows "7 followers" and "4 following".

Below this section are two posts by the user:

Post Title	Author	Posted In	Date	Actions
asdasd	@thomas kenny	in General	(9 days ago)	▼ 1 □ 1 Delete
test	@thomas kenny	in General	(21 hours ago)	▼ 2 □ 2 Delete

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 versa 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

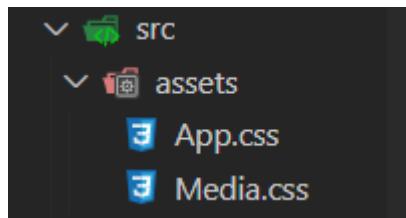
```
// create schema for the database
const DiscussionSchema = new Schema({
  user:String,
  user_id:String,
  title:{type: String , required: true},
  caption:String,
  content:String,
  time: {
    type: Date,
    default: Date.now,
  },
  society:String,
  society_id:String,
  claps:Number,
  comments:Array,
  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,
}
);
```

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 is 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 [14] 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 [15]. 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[16].

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 [Light-house](#) 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].

```

_id: ObjectId("5fac23d6130cbd06d803fb8f")
users: [
  {
    _id: "5fbfdc340beba0351c030873"
    fullname: "thomas kenny"
    email: "thomas@gmit.ie"
    societies: [
      pic: "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAEBAQEBAQEBAQ...
    ]
    name: "Computer Science"
    college: "GMIT"
    category: "Technology"
    address: "GMIT GALWAY"
    private: false
    score: 353
  }
]

```

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.

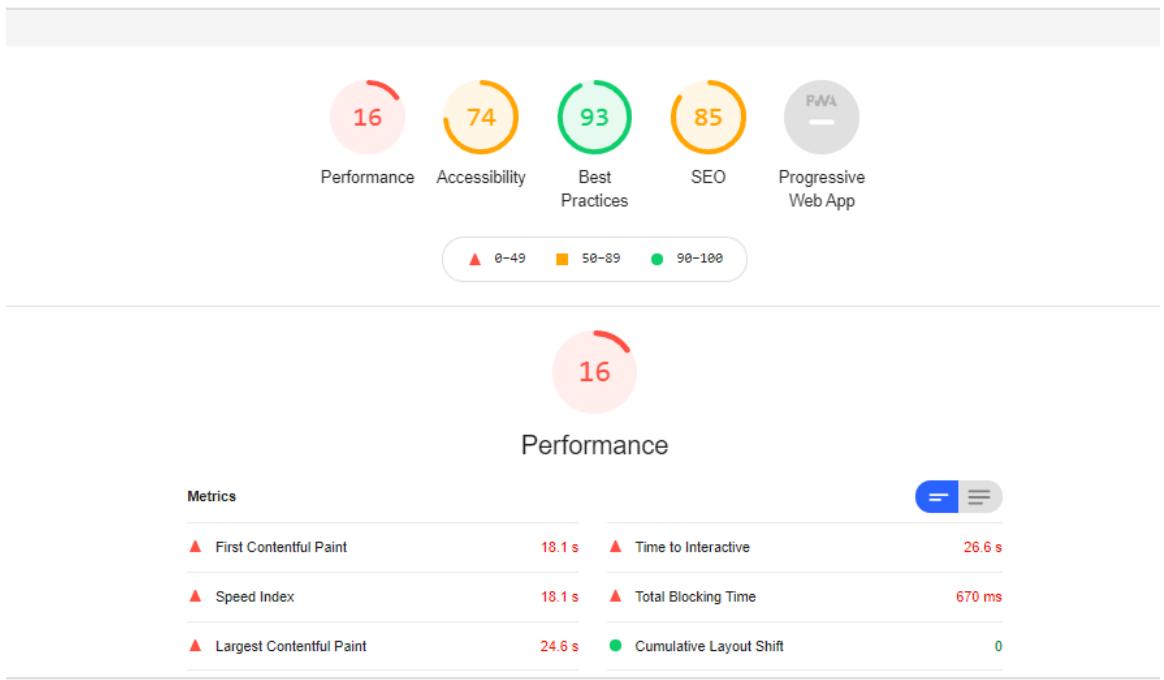


Figure 6.2: Poor Lighthouse scores

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].

Name	Status	Type	Initiator	Size	Time	▼
get-discussion-feed	200	xhr	xhr[§177]	189 kB	2.17 s	
get-users-rear	200	xhr	xhr[§177]	984 kB	1.06 s	
get-user-details?id=5fe9624ea92818192ccabe92	200	xhr	xhr[§177]	72.2 kB	827 ms	
get-user-details?id=5fe9624ea92818192ccabe92	200	xhr	xhr[§177]	72.2 kB	807 ms	
lchunk.js	200	script	top	3.0 MB	422 ms	
css27family=Payone+One&display=swap	200	stylesheet	top	510 B	113 ms	
css27family=Note+Serif&display=swap	200	stylesheet	top	632 B	112 ms	
css27family=Rock+Slab&display=swap	200	stylesheet	top	302 B	110 ms	
css27family=Open+Sans&display=swap	200	stylesheet	top	630 B	55 ms	
css27family=Righteous&display=swap	200	stylesheet	top	447 B	54 ms	
css27family=Montserrat&display=swap	200	stylesheet	top	596 B	54 ms	
Onicapp97family=Oriental&display=swap	200	font	css27family=Payone+One&display=swap	19.8 kB	34 ms	
css27family=Robot&display=swap	200	stylesheet	top	644 B	34 ms	
main.lchunk.js	200	script	top	754 kB	29 ms	
get-explore-societies	200	xhr	xhr[§177]	570 B	26 ms	
main.34a5cd73f804d1d0860.hot-update.js	200	script	top	3.7 kB	16 ms	
bundle.js	200	script	top	6.7 kB	15 ms	
...						
Name	Status	Type	Initiator	Size	Time	▼
get-user-details?id=5fe9624ea92818192ccabe92	200	xhr	xhr[§177]	354 kB	362 ms	
get-user-details?id=5fe9624ea92818192ccabe92	200	xhr	xhr[§177]	354 kB	283 ms	
get-discussions-fields?user=society+time+thumbnail_pic+title+content+likes+comments	200	xhr	xhr[§177]	354 kB	270 ms	
get-users-rear	200	xhr	xhr[§177]	159 kB	207 ms	
get-user-details?id=5fe9624ea92818192ccabe92	200	xhr	xhr[§177]	354 kB	96 ms	
react_devtools_backends.js	200	script	injectGlobalHook.js:833	446 kB	60 ms	
get-explore-societies	200	xhr	xhr[§177]	573 kB	42 ms	
main.6dd0040447e78aff53.hot-update.js	304	script	top	202 kB	9 ms	
main.lchunk.js	304	script	top	203 kB	9 ms	
lchunk.js	304	script	top	204 kB	8 ms	
favicon.ico	200	x-icon	Other	3.4 kB	3 ms	
extra-utils.html	200	document	jquery-3.1.1.min.js:3	1.1 kB	3 ms	
manifest.json	304	manifest	Other	265 B	2 ms	
logo192.png	304	png	Other	266 B	2 ms	
bundle.js	304	script	top	202 kB	2 ms	
top	304	document	Other	201 B	2 ms	

42 requests | 617 kB transferred | 14.8 MB resources | Finish: 916 ms

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

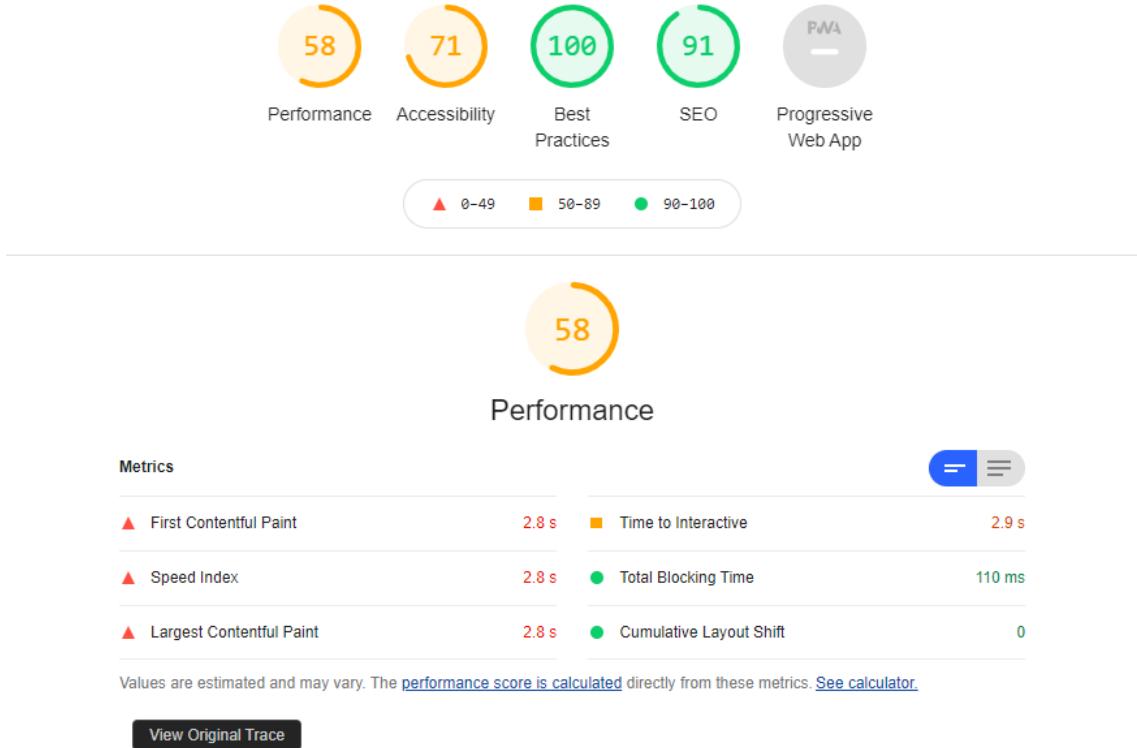


Figure 6.4: Improved Lighthouse Performance

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].

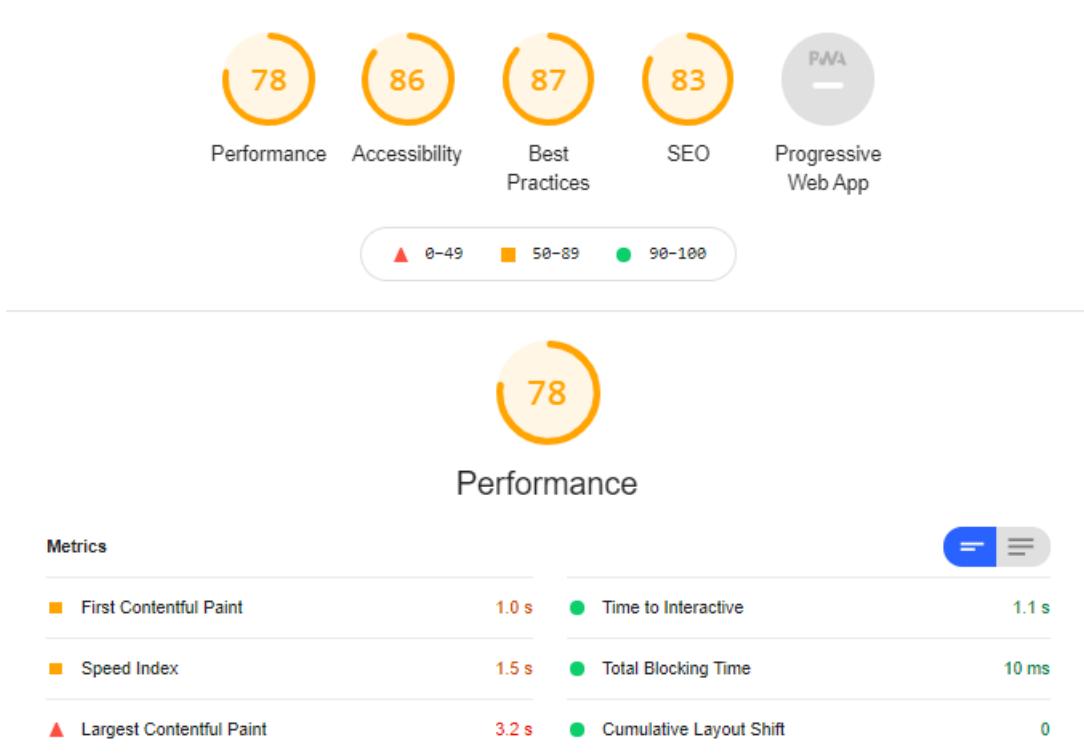


Figure 6.5: Highest Lighthouse Performance

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. Wireframing the designs is extremely important as they clarify the websites features and functionalities. By creating wireframes this will flatten out the design of the user interface so that the team members working on the Front-End understand what it is that needs to be made and what layout they need to create. Figma is a browser based User Interface and User Experience platform that allows you to create and prototype designs within minutes [17]. 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.

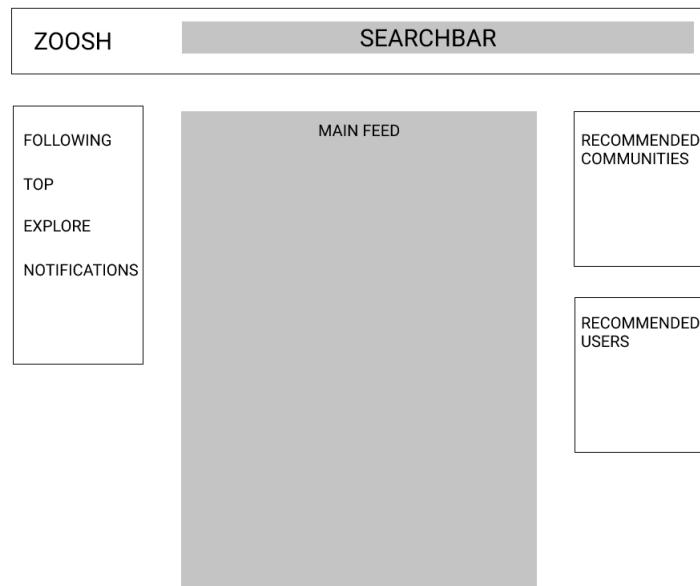


Figure 6.6: Figma Design.

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 have designed or 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 O'Brien for supporting us along the way if we had any issues.

Chapter 8

References

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] C. B. Stapper, “An analysis of heroku and aws for growing startups,” 2017.
- [11] 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.
- [12] “Waterfall methodology diagram.” <https://project-management.com/wp-content/uploads/2020/05/waterfall-project-management-banner.png>.
- [13] “Agile methodology diagram.” <https://hackernoon.com/images/6t8232hs.jpg>.
- [14] “React bootstrap.” <https://react-bootstrap.github.io/>.
- [15] “React icons.” <https://react-icons.github.io/react-icons/>.
- [16] M. Soomro, “Deploying mern app to heroku.” <https://coursework.vschool.io/deploying-mern-app-to-heroku/>.
- [17] “Figma ui ux tool..” <https://www.figma.com/files/recent?fuid=899606574753017180>.