

---

# Final Year Project Title

---

**Aaron Moran**

**Conor Shortt**

**Thomas Kenny**

B.Sc.(Hons) in Software Development

APRIL 26, 2021

**Final Year Project**

Advised by: Kevin O'Brien

Department of Computer Science and Applied Physics

Galway-Mayo Institute of Technology (GMIT)



# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Context</b>	<b>5</b>
2.1	Filler . . . . .	5
<b>3</b>	<b>Methodology</b>	<b>6</b>
3.0.1	Testing . . . . .	8
<b>4</b>	<b>Experiments</b>	<b>9</b>
<b>5</b>	<b>Technology Review</b>	<b>14</b>
<b>6</b>	<b>System Design</b>	<b>15</b>
<b>7</b>	<b>System Evaluation</b>	<b>16</b>
<b>8</b>	<b>Conclusion</b>	<b>17</b>
<b>9</b>	<b>References</b>	<b>18</b>

# About this project

**Abstract** This project is an Social Media platform where you can create communities, post up discussions and follow your friends. . . . etc etc

**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 our interest in social media we decided early on 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 UI. We were confident from the start with the team we have we could develop a website that could meet the standards of the websites out in the world today. Before any developing took place, it was crucial we carried out in depth research into the top social media websites , how we were going to develop it , type of database to use and too draw up a timeline of goals/targets.

# Chapter 2

## Context

- Provide a context for your project.
- Set out the objectives of the project
- Briefly list each chapter / section and provide a 1-2 line description of what each section contains.
- List the resource URL (GitHub address) for the project and provide a brief list of the main elements at the URL.

### 2.1 Filler

# Chapter 3

## Methodology

About one to two pages. Describe the way you went about your project:

- Agile / incremental and iterative approach to development. Planning, meetings.
- What about validation and testing? Junit or some other framework.

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:

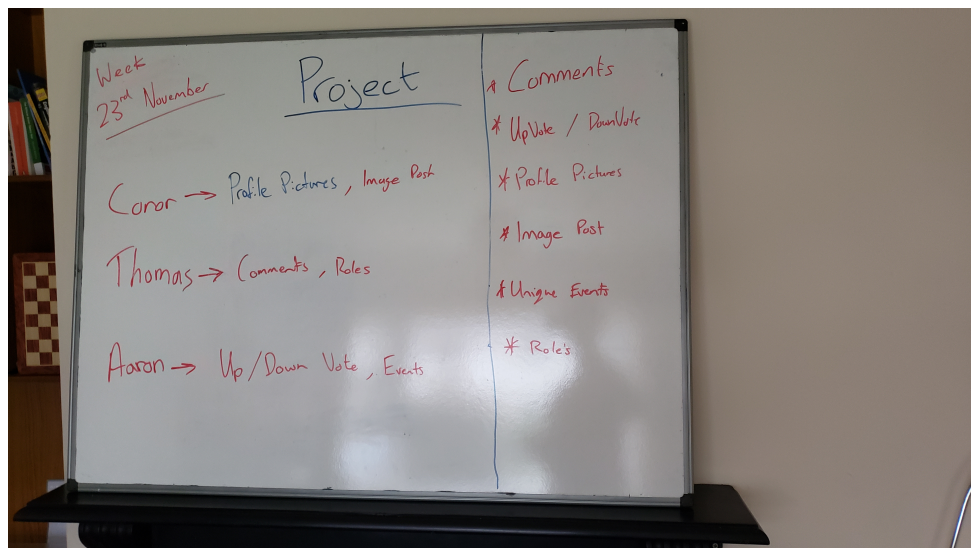
- **Functional** - Probably the most important aspect of every website. It must work quickly and as expected. Page loading times must be kept as low as possible, especially when working with large amounts of data - this was a big issue that we encountered when dealing with images.
- **Easy to Use** -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

### 3.0.1 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 two main testing methods used are White box testing and Black box testing.

- Selection criteria for algorithms, languages, platforms and technologies.

Check out the nice graphs in Figure ??, and the nice diagram in Figure ??.



# Chapter 4

## Experiments

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 [4.1].



```

> {
  "_id": ObjectId("5fac23d6130cbd06d803fb8f"),
  "users": Array(
    0: {
      "_id": "5fbfdc340beba0351c030873",
      "fullname": "thomas kenny",
      "email": "thomas@gmt.ie",
      "societies": Array(
        0: {
          "pic": "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAQABAAQ/2wBDAAEBAQEBAQEBAQE..."
        }
      )
    }
  ),
  "name": "Computer Science",
  "college": "GMIT",
  "category": "Technology",
  "address": "GMIT GALWAY",
  "private": false,
  "score": 353,
  "__v": 0
}

```

Figure 4.1: 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 [4.2]. 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.

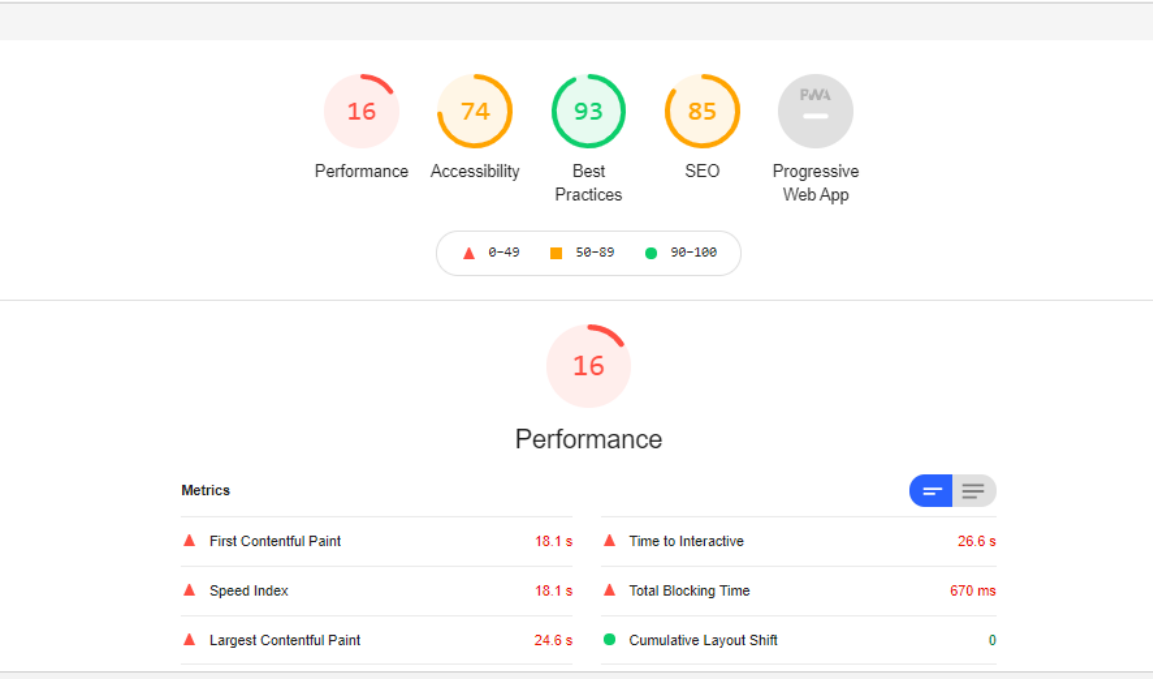


Figure 4.2: Poor Lighthouse scores

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 [4.3]. 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 [4.4].

Name	Status	Type	Initiator	Size	Time
get-discussion-feed	200	xhr	xhr.js:177	189 kB	2.17 s
get-users-radar	200	xhr	xhr.js:177	98.4 kB	1.06 s
get-user-details?id=5fa9624ea92818192ccabe92	200	xhr	xhr.js:177	72.2 kB	827 ms
get-user-details?id=5fa9624ea92818192ccabe92	200	xhr	xhr.js:177	72.2 kB	807 ms
1.chunk.js	200	script	100	3.0 MB	422 ms
css2?family=Paytone+One&display=swap	200	stylesheet	100	510 B	113 ms
css2?family=Noto+Serif&display=swap	200	stylesheet	100	632 B	112 ms
css2?family=Rock+Set&display=swap	200	stylesheet	100	382 B	110 ms
css2?family=Open+Sans&display=swap	200	stylesheet	100	630 B	55 ms
css2?family=Rigorous&display=swap	200	stylesheet	100	447 B	54 ms
css2?family=Montserrat&display=swap	200	stylesheet	100	596 B	54 ms
OnkaC9P7MfrHjZoFtm2CnTgPs.woff2	200	font	css2?family=Paytone+One&display=swap	19.8 kB	34 ms
css2?family=Roboto&display=swap	200	stylesheet	100	644 B	34 ms
main.chunk.js	200	script	100	75.4 kB	29 ms
get-explore-societies	200	xhr	xhr.js:177	570 B	26 ms
main.34a5cda73f804d100860.hot-update.js	200	script	100	3.7 kB	16 ms
bundle.js	200	script	100	6.7 kB	15 ms

Name	Status	Type	Initiator	Size	Time
get-user-details?id=5fa9624ea92818192ccabe92	200	xhr	xhr.js:177	354 B	362 ms
get-user-details?id=5fa9624ea92818192ccabe92	200	xhr	http://localhost:4000/users/get-user-details?id=5fa9624ea92818192ccabe92	354 B	283 ms
get-discussions?fields=user+society+time+thumbnail_pic+title+content+likes+comments	200	xhr	xhr.js:177	354 B	270 ms
get-users-radar	200	xhr	xhr.js:177	159 kB	207 ms
get-user-details?id=5fa9624ea92818192ccabe92	200	xhr	xhr.js:177	354 B	96 ms
react_devtools_backend.js	200	script	inject:GoogleAnalytics833	448 kB	60 ms
get-explore-societies	200	xhr	xhr.js:177	573 B	42 ms
main.06d6d0484e7e78a11f53.hot-update.js	304	script	100	202 B	9 ms
main.chunk.js	304	script	100	203 B	9 ms
0.chunk.js	304	script	100	204 B	8 ms
favicon.ico	200	image	Other	3.4 kB	3 ms
en-us.js.html	200	document	JavaScript:3.1.1.min.js	1.1 kB	3 ms
manifest.json	304	manifest	Other	265 B	2 ms
logo192.png	304	png	Other	265 B	2 ms
bundle.js	304	script	100	202 B	2 ms
top	304	document	Other	201 B	2 ms

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

Figure 4.3: Slow Requests (Top) vs Improved Requests (Bottom)

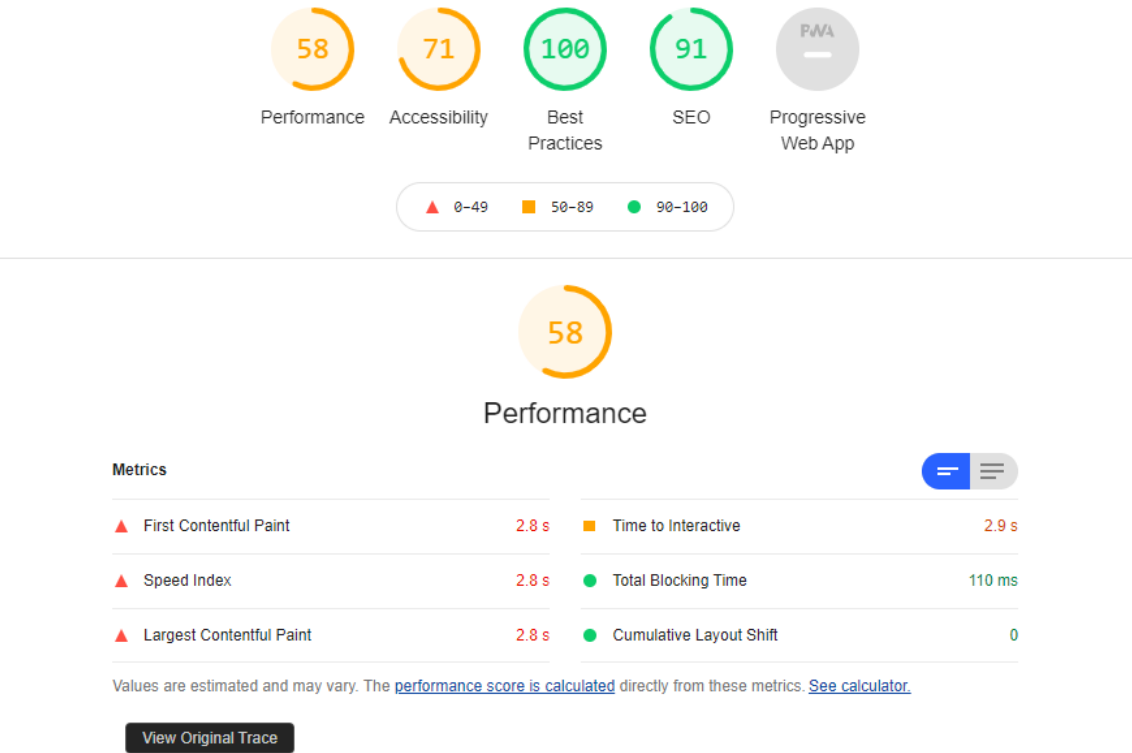


Figure 4.4: Improved Lighthouse Performance

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 our application. This helped us to achieve satisfying results as our applications load times improved dramatically and our Google Lighthouse Performance was showing better results.

- Website Runtimes
- Responsiveness
- Speeds and Performance
- Loading in images via MongoDB/Amazon
- Front End
- Colorways
- UI
- FIGMA, WIREFRAME
- Design Decisions
- Icon imports, Bootstrapping

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, 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.

# Chapter 5

## 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](#).

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.

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.

# Chapter 6

## System Design

## Chapter 7

# System Evaluation



## Chapter 8

# 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.

Ultimately, working as a team to produce this project has helped us to develop and improve on our team working skills and problem solving skills

...

## Chapter 9

## References

# Bibliography