Cognito – Trust-Based Social Network with Spam Detection

Final Project Report

TU856

BSc in Computer Science

Eoghan Martin

C18342116

Supervisor – Jack O’Neill

School of Computer Science

Technological University, Dublin

08/04/2022

Abstract

It’s no doubt that social media has become a place that is unsafe at times and can be filled with spam, cyber bullying, discrimination and more. The issue lies in the lack of accountability and the ease with which people can hide behind a blank profile or pretend to be someone else. New research shows that only 59% of accounts on Instagram in the US are real people, the rest consists of bots, inactive accounts and mass followers (accounts with over 1,500 following) **(1)**. It has been well documented at this stage that social media has a big impact on people’s mental health and with rises in hate from fake accounts and blank profiles this issue is only getting worse.

As well as this spam accounts are becoming more and more prevalent online, it has been reported that an estimated 13% of accounts are fake on Instagram and 14.6% of twitter accounts are fake **(2)**, this causes a lot of problems both in the rise in scams, catfishing, spam promotions etc. These problems are never going to go away fully but new solutions to minimise them are necessary .

The goal of this project is to propose a new way of dealing with this issue by allowing a social network built on peer-based trust and using other algorithms and techniques to be able to catch spam accounts early. The development of this social media application will take into consideration what people want to see in such an application.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

[illegible]

Eoghan Martin

08/04/2022

Acknowledgements

I would like to thank my project Supervisor Jack O’Neil for his continuous support and guidance throughout the project and the Project Coordinator Damian Gordon. I would also like to thank my family and friends for their help and encouragement throughout.

Table of Contents

List of Figures

1. Introduction

1.1. Project Background

Over the last few years there have been exponential rises in spam accounts and people who are not who they say they are. This has become a huge issue within the area of social media; one incident in particular was the Euros 2020 final where bouts of racist abuse were directed towards players on the English team following their defeat. This was a terrible incident but the horrific nature of this led to people asking important questions about the security of these social networks and the impact they have on people and their mental health. In some cases, identities of the accounts were easy to find, and those people were held accountable but some accounts were given free rein to pollute the timeline with horrible tweets without consequence (3).

This raised the question of whether there was any way of stopping these spam and fake accounts from being allowed onto social networks or removing them once found. There have been a number of ways this has been done on twitter in detecting spam accounts such as the *anomaly detection* method (4). This method is used to detect tweets that don't follow the usual behaviour of tweets, 'Behaviour is considered dangerous if it differs from normal'. This is an important thing to consider when considering possible solutions to this issue, which will be discussed in Section 4.3.1, later in this report.

Other methods for detecting spam accounts on Instagram have been experimented with using machine learning techniques, such as the method proposed by Sheikhi (5). In this experiment, Sheikhi employed bootstrap aggregation (bagging) with a decision tree model in a fake account detection dataset and achieved a 98% success rate on a hold-out test set.

1.2. Project Description

The goal of this project is to create a secure social network where all users are verified by their own peers, the user can determine whether they trust an account or not and this trust is then stored using a graph database. This will give users power in the network to decide who is trustworthy but also gives them responsibility and accountability.

Cognito will be a Social Media Application that will be more secure and safe than current social media networks. Once users log in they can try to find friends to add and build their network and choose to trust their friends etc. thereby building confidence in their profile. Once the user builds their network and begins to see other users with mutual friends they will begin to see personalized trust levels for all these users. This can help determine whether or not to accept requests etc.

Over the last year or two there have been a lot of instances where spam accounts or fake accounts are being used to spread hate and misinformation without consequence, like the racist abuse to certain players after the Euro 2020 finals and many more instances. The idea of this application is to hold everyone accountable, therefore a trust-based method would help prevent similar situations in future.

As well as this there is the issue where a bunch of spam accounts might choose to make multiple accounts and decide to trust each other. This brings in a new issue where the number of trusted connections and personalized trust levels of users can seem safe but are artificially manufactured through spam users building their network through creating a spam cluster of users. To solve this issue algorithms using graph theory and spam behaviour analysis will be used.

Along with this security is the need to balance having free speech on the application which is important as it is intended to be modelled after the real world, but like in the real world users will be able to say what they want but not without consequence for the user themselves and also their trusted connections. This will work by users only trusting accounts that they truly trust because if this person does something or turns out to be a spam account, they are also accountable and linked to that account, therefore damaging their reliability on the network. This added element will get users to think carefully before creating trust connections, because it could affect them as well.

1.3. Project Aims and Objectives

Objectives:

- Create an Application that reflects the real world
- Establish trust connections between users
- Define levels of trust and type of trust between users
- Spam Detection algorithm
- Standard social media features

The main objective of this project was to try to reflect the real world as much as possible and implement the consequences that actions have in that world unlike the social media world.

The next objective will be creating a system wherein the users will be able to choose which accounts they trust and will be able to see who trusts who, this will promote users to build their own network in order to gain the trust of others. This will then all be recorded using a graph database in order to map the relationships between users and who they trust and who their friends trust etc.

Like in the real world we have varying levels of trust for all of our friends, and this can change over time, and this will be another aim of the application, to allow users to define how much they trust certain people if at all and be able to change this over time. This will show users how much they should trust someone based on how much their friends trust them. Users will also be able to report users they suspect as spam accounts or that they do not trust to be real and the spam algorithm will check the user and their trust connections. It is also important that standard social media features will be implemented. This will include posting, messaging, sending trust requests etc.

1.4. Project Scope

The scope of this project is to create a secure social network for users that is safe from spam accounts and fake accounts and will allow users to be responsible for who they choose to trust on the network. This will include having a database that will allow for queries to find spam account clusters, will allow users to be build their relationships on the application and see who they should or shouldn’t trust.

Although the end goal of this project was to have a finished polished application, the complexity of the project comes from the backend, so this was prioritised throughout the project. I was hoping to complete everything including a front end that was built using react in order for the application to be used on all devices. There are also extra features like group chat messaging or photo sharing that would possibly be implemented in future and will be re-visited after the completion of my degree.

1.5. Thesis Roadmap

This Section will provide a summary of every chapter in this dissertation.

Research

This chapter will explore the research behind what users would love to see in a system that would make them feel safe and secure, it will also explore the different technologies that were researched in terms of how my system will be developed. Some existing previous final year projects have also been included that have some similar ideas, one detecting bot accounts on twitter and the other managing a NoSQL database and service provider.

Design

The Design chapter will dive into the design process why the design for the system was chosen. I will start with the basic System Architecture followed by Graph Database diagrams as well as Class Diagrams and Use Cases. These will all contribute towards giving a general idea of the initial designs for the development of the system.

Development

The Development chapter will discuss not only the development of the project but also the future development of the system. This will include steps taken into the development process including setting up and populating the user databases, the logic layer development, the API layer development and the presentation layer development.

Testing and Evaluation

For testing and evaluation I will discuss how the application was tested and evaluated. This will include evaluation of different testing techniques and what technique was best for my particular application. I will also discuss how I got users to try the system and my evaluation of their opinions and suggestions.

Conclusion and Future Work

The final chapter of this report will discuss the initial project plan and any changes made along the development process as well as drawing all conclusions from the project and will discuss further work that could be done and for this application.

2. Literature Review

In this chapter the research undertaken for this project will be discussed. Alternate solutions similar to this project, technologies researched for the development of the system and the survey research that was done will be discussed. There will also be discussion on how the survey with user opinions and ideas would affect the design and development of the project.

2.1. Alternative Existing Solutions

While researching into the idea of graph databases in social networks it was found that a popular social media network uses similar ideas to this Project. LinkedIn (6) is a social network for the working world where users can interact and build contacts for their careers, apply for jobs, contact employers and post about different achievements in your careers and much more.

They use a similar idea of connections between users, when you log in to LinkedIn and start to make connections you can see users who are a certain degree of separation from you. This is in the form of 1st, 2nd, 3rd, where 1st would be a friend, 2nd would be a friend of a friend *etc*, an visual representation of this is seen in Figure 1. 1 . This will work in a similar way to the system being developed but some features will be added including your personalised level of trust and more, to cater to the wants and needs of users on this particular system.

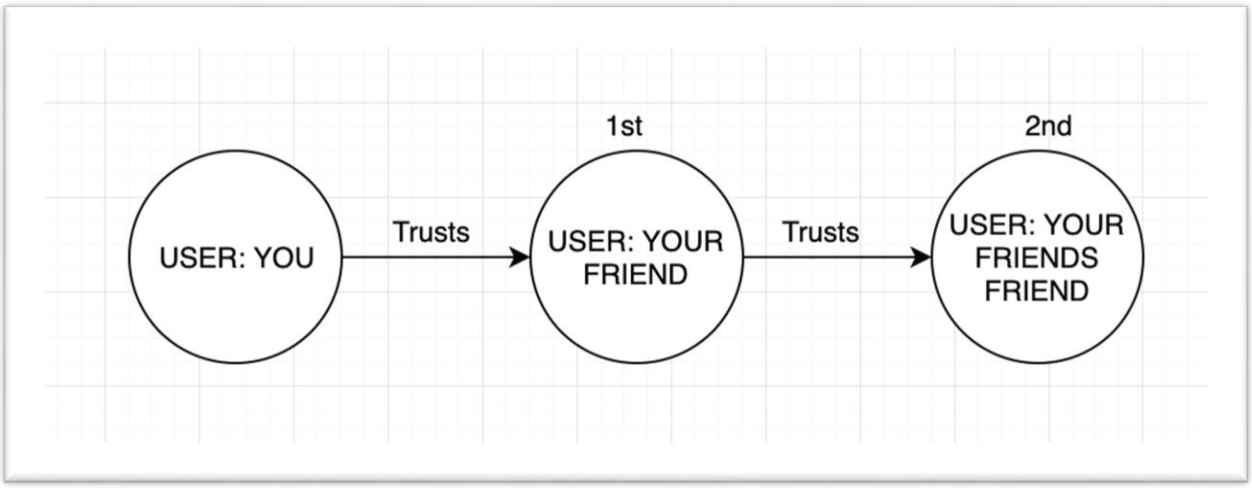


Figure 1. 1 Degree of Separation Visual

2.2. Technologies Researched

In this section I will discuss the technologies researched for use in this project

2.2.1. Graph Databases:

In many ways social networks already uses the same idea as graph databases, there are *nodes* (users) and the relationships between these nodes, this means there is no real need for us to convert these graphs into tables like in normal relational databases (7). Similarly in real life this is how real relationships work in theory, there are people (nodes), and they have relationships with other users. For this particular application this is very important because we focus a lot on the relationships between people and the trust levels they have, as well as community detection techniques that are not easy to implement in relational databases. This is the main reason for the choice to use a graph database to store users for the project. There are many different available NoSQL resources that use graph databases like Neo4j and Orient DB.

Graph DB Free: (8) Graph DB free is a free database solution, it can support up to 100 million nodes locally. It is deployed using Java and SPARQL is used to query the database in memory. This database software was one of the ones considered when deciding what to use for this project, it had all the necessary requirements for my project however it supports only low query loads and is suited for smaller projects which is okay for the moment but in future it may not be suitable if the system were to grow.

Orient DB: (9) This database software is a database software that can store up to 220,000 records per second in common hardware. It is a document based database and relationships are managed as in *Graph DB Free* with direct connections among records. Orient DB is also good for modelling data with complex relationships, which is not exactly something that's a requirement for my system, as my relationships between nodes would all be quite similar.

Neo4j Community Edition: (10) Neo4j is a very well-known open-source graph database which is implemented in java. It is a disk-based, embedded, fully transactional Java persistence engine, that stores data in graphs rather than tables. It delivers extremely fast read and write performance while protecting data integrity which is very important in this user-based system. As well as all of this Neo4j has a huge community of people who can help beginners and online forums and videos to help starting off for beginners.

2.2.2 Relational Databases

Although it was important to store users in a graph database the users posts, messages and notifications will be stored in a relational database. The idea would be to take the username at an application level from the graph database and use this to store users posts, messages and notifications when certain actions are taken by the user. This will instil referential integrity in the database. As part of the research into what relational database I would use I researched PostgreSQL and MySQL.

PostgreSQL: (11) PostgreSQL is an open source relational database system that has a good reputation for reliability and performance. It is the most professional of the relational database systems and is highly reliable and scalable. There is a big community of support that helps with common issues found when developing using it and this is important as it can speed up the cost of the development process.

MySQL: (12) MySQL is one of the most recognised systems, it is a widely compatible Relational Database Management System that can runs on all major platforms. It is an easy-to-use system that provides comprehensive support for developing applications that can speed up the development process.

2.2.3. Server-Side:

PHP: (13) PHP is one of the most widely used server-side language, that works very well and is reliable. It is open-source and supports cheap hosting, rapid development and it is easy to find PHP resources. It is also very easy to deploy, making it a very suitable server-side language for this project. As the systems grows however there might be problems found such as scaling issues that may not occur with other server-side languages.

Node.js (JavaScript): (14) Node.js is a newer server side and it is the quickest growing server side language. Node.js works very well compared to other server-side languages in real-time applications which would be beneficial in a social media network as a lot of functionality such as chats etc is in real time. This language is essentially the same as JavaScript which means you only have one language for front-end and back-end.

Java: (15) Java is a very popular language that is used in the server-side. For smaller applications it can be overkill as you can achieve the same results with something simpler but is suitable for large enterprise applications where it is widely used. It has great performance on a large scale and for a social media application down the line, this would be useful. It is also easy to pick up if you have the prior knowledge of java and its concepts.

2.2.4. Front-end:

Native Apps: (16) Native applications are developed using the languages and technologies specifically designed for certain systems such as iOS and Android. These languages have a lot of great benefits for users but also some drawbacks in the development process. Developing a native application improves the user experience as it has specific platform-customised UI/UX components and can provide better security due to various layers of protection of the operating system. Although these benefits are great there is also the cost of developing these apps in comparison to hybrid applications. In the case of this project users should be able to access this system from a laptop or a smartphone, including both IOS and Android devices. This would mean an application would have to be developed on each separate device, which takes a lot longer. As well as this with new features being added, these features need to be added on each code-base which can complicate things.

Hybrid Apps: (16) A hybrid application is a combination of both native and web apps. It is developed in just one language and can be run on multiple operating systems. You can use more well-known and widely used languages such as JavaScript, HTML and CSS. There are huge advantages to this meaning you can develop a multi-platform system a lot quicker and things such as new features can be added a lot more easily as there is only one codebase to change and the changes will be implemented across all platforms. There are some drawbacks in hybrid applications however such as the run time if you have a complex application. Other issues such as security and less freedom of development as the code is not being developed to perform specifically on one platform.

2.2.5. Software Development Methodologies:

Agile Development Methodology: (17) Agile is an iterative approach to development where solutions and requirements evolve throughout the development process. Agile's big benefit is the minimization of risks when adding new functionality. There are various different agile methods such as scrum, crystal and extreme programming. In each of these methods the teams developing will develop software in iterations, seen in Figure 2. 1. This can be very useful as defects can be found very early on in development. However, it requires a huge time commitment from users and can be labour intensive for developers.

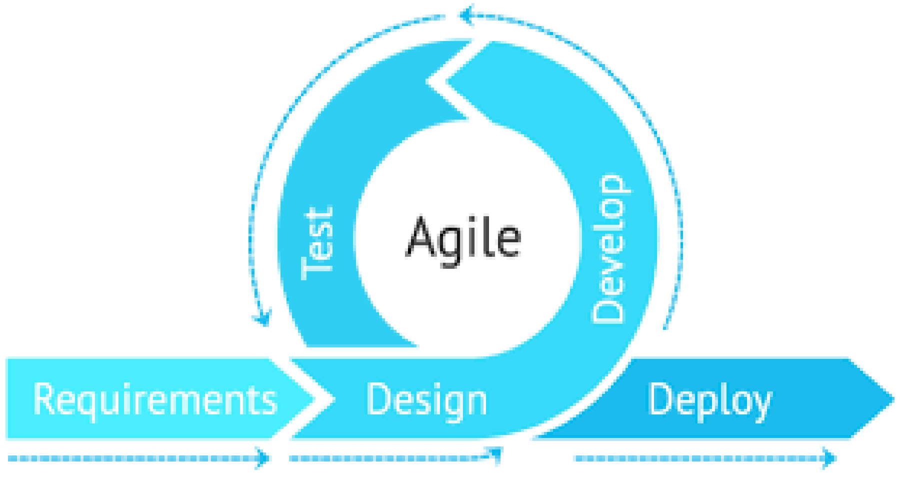


Figure 2. 1 Agile Dev Diagram

DevOps Development Methodology: (17) This methodology focuses on organizational change that allows collaboration between the different departments in charge of specific parts of the development life cycle. Although similar to Agile the key difference is DevOps brings development and operations teams together whereas Agile is an approach that focuses on collaboration, customer feedback and small rapid releases. DevOps is great for lowering the rate of failure of new releases and will implement continuous development, seen by Figure 2. 2, to ensure reliability which is great for customer satisfaction. Although some customers don’t want consistent updates to their systems and sometimes there is extensive testing required that doesn’t allow for this methodology.

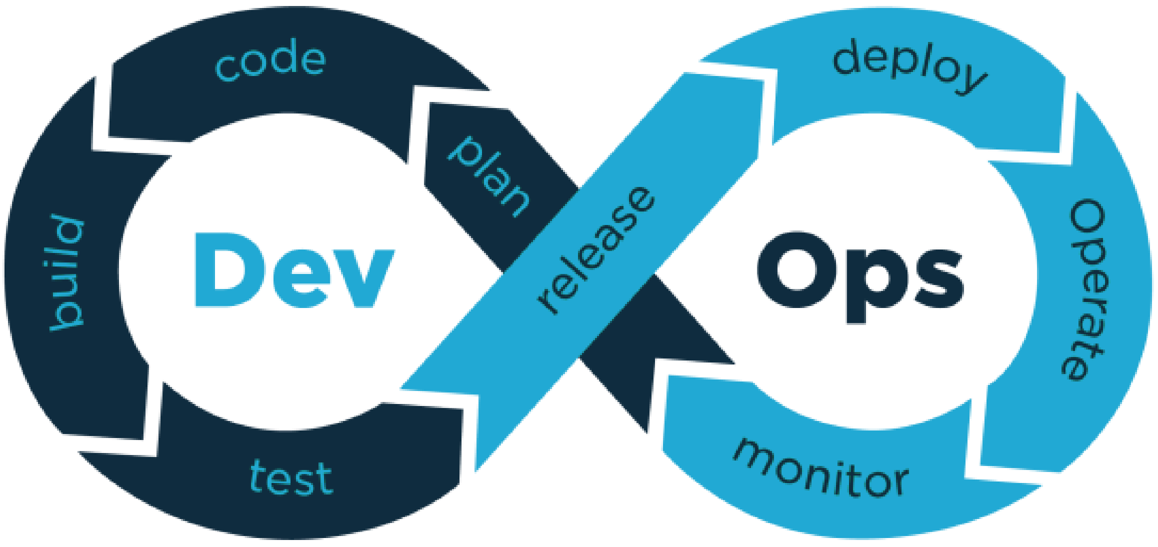


Figure 2. 2 Dev Ops Diagram

Waterfall Methodology: (17) This is the most traditional software development method. It is a linear model that has phases such as requirements, design, implementation, verification and maintenance, seen in Figure 2. 3. Since the approach is linear, each step is completed in full before moving onto the next, so this methodology does not really support easy modification of the system once complete which can be useful in some cases but has its cons. The nature of the waterfall method makes it easy to understand and to manage, it is useful for projects with clear objectives and stable requirements. In a lot of cases, however, it is slow and costly (18).

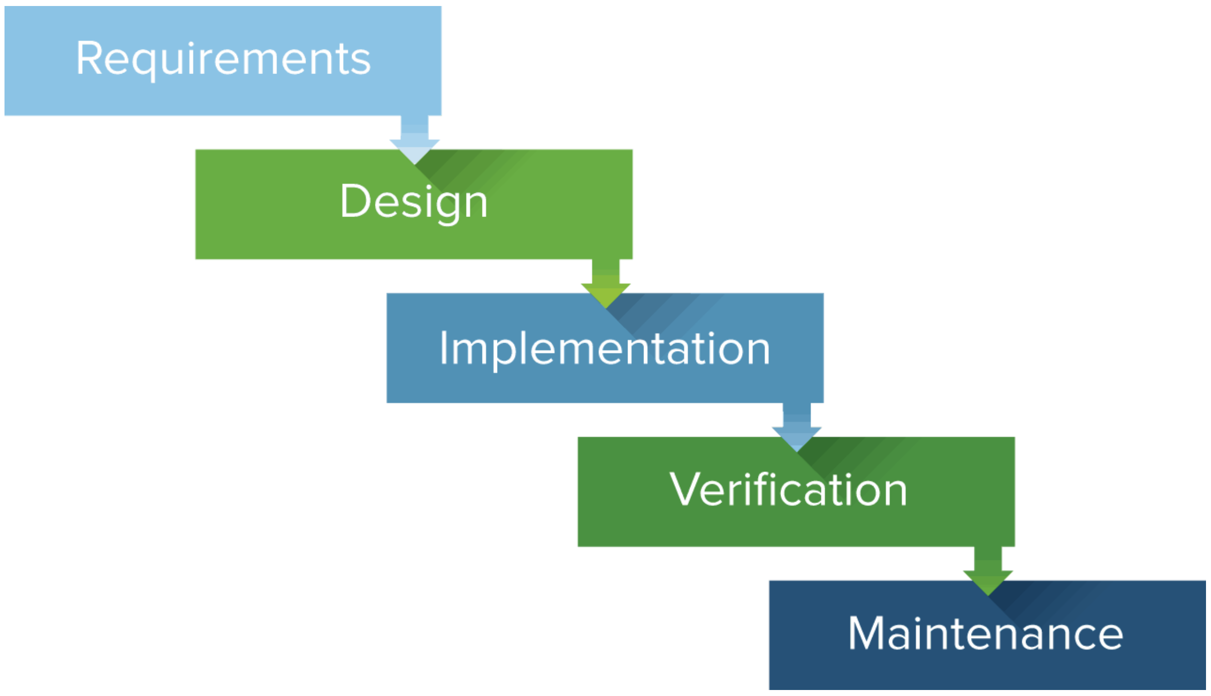


Figure 2. 3 Waterfall Diagram

2.3. Survey Research

I conducted a survey (19) as part of the research for this project in order to find out exactly what users wanted and needed in a social application; this can be seen in Appendix A. The survey included some questions that were thought about in the designing of this system and what users would actually want to see on this system.

The first question that was asked was what users look for themselves when trying to determine if a profile is real or not on social media, the responses can be seen in Figure 2. 4. The top two (tied) responses were "pictures" and "mutual followers", this was a surprise because mutual followers can be understood a bit more but with pictures, anyone can post pictures of anything but my thoughts would be there is more to it when it comes to events people have been at and the times they post pictures etc.

The next two – *verification ticks* and *following size* – were lower and this was also a bit of a surprise but it seems that because "verification ticks" are mainly used for celebrities and public figures but when verifying their friends they know that they won't have a tick because they might not be a celebrity or public figure. "Following size" was next which is understandable to some degree. Huge following sizes are generally seen as a good sign when deciding if an account is real or not.

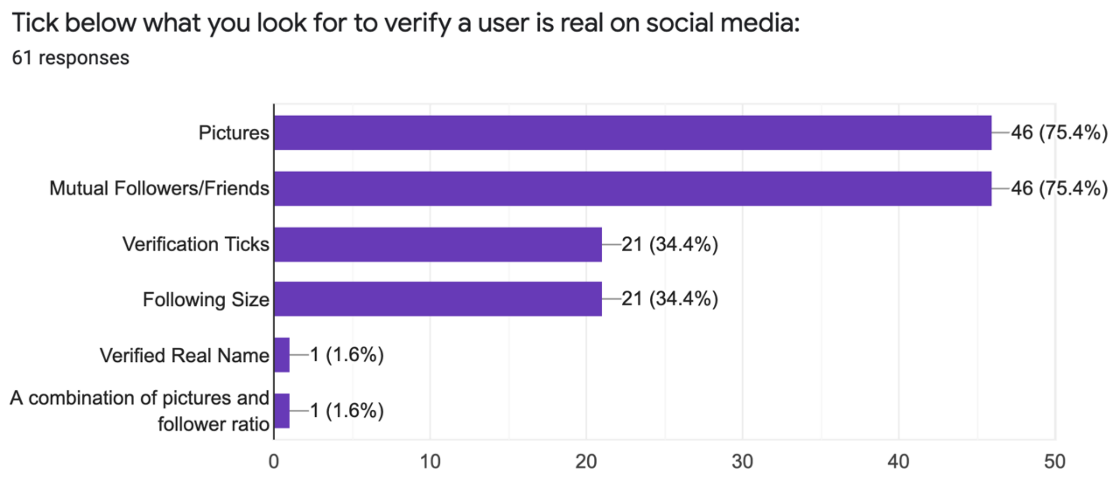


Figure 2. 4 Survey – User Verification

One of the key parts in the system was providing some sort of spam user detection. This will find spam users automatically and limit or remove them. The algorithm might not accurately remove 100% of spam users and in some cases might accidentally remove legitimate users, so I asked in the survey if users would be okay if 1% of the users that were removed were legitimate, seen in Figure 2. 5. Almost half answered yes but there was some scepticism about this which underlines the importance of building an algorithm which does not produce false positives.

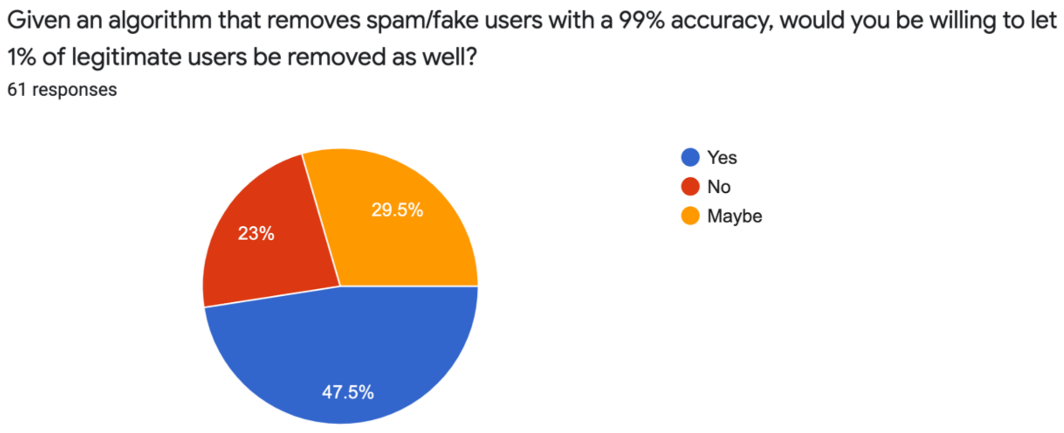


Figure 2. 5 Survey – Removal of legitimate users question

This question in Figure 2. 6 asks *would you be willing to let more than 1% be removed* and it shows that only around half of the people who voted yes in the previous question would be okay with this. This shows that the algorithm would need to have a false positive rate of at most 1% in order for the majority of users to be satisfied.

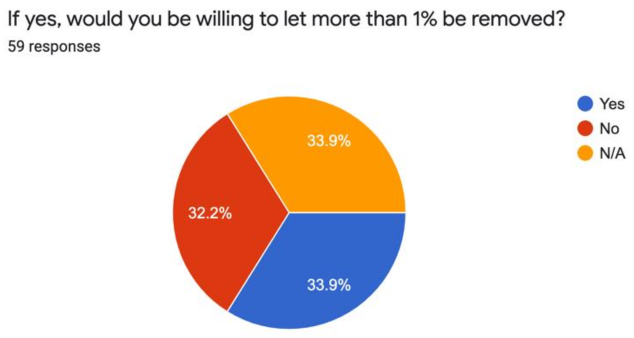


Figure 2. 6 Survey - Removal of legitimate users question (Part B)

What I found most interesting about the question in Figure 2. 7, *If this algorithm found you in the 1% would you still have the same response*, was the change in peoples answers if they were affected by the accidental deletion of legitimate accounts, although the majority would keep the same response 35.6% would not and 15.3% might change it. This was very interesting because a lot of people might not see this happening to them but if it did they might not be as okay with the idea. This is important to consider when designing the algorithm for the application, specifically with the actions taken on these accounts, it is important to note that an algorithm will never catch 100% of accounts so if users are uncomfortable with them being caught in it, then removing users is not be the best option.

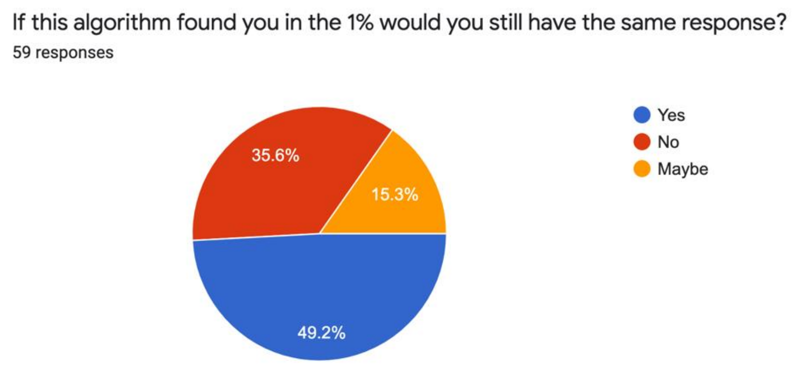


Figure 2. 7 Survey – Removal of legitimate users question(Part C)

Next in Figure 2. 8, it was asked if users need to be held accountable for what they say when it comes down to discriminatory behaviour online. An overwhelming majority of people would want to see this, this can be done in a range of different ways, such as limiting functionality for these users, permanent ban, temporary ban *etc.*

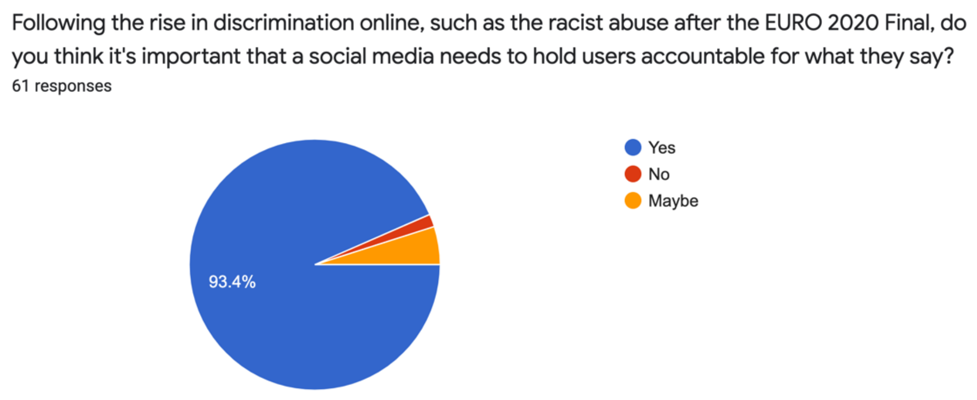


Figure 2. 8 Survey – Accountability Question

A follow-up question to this was asked which was what kind of action should be taken on these profiles and a lot of answers were received, the majority of them were to ban the account, but there were some other ideas that were interesting as well. Some people said there should be a strike system allowing users more than once chance which could be very effective. Others suggested suspension of account or limiting their capabilities temporarily and much more. I think there was a lot to think about when it came to this specific problem and these suggestions led me to think that some sort of ban whether temporary or permanent was in order, perhaps a ban on doing certain things on the application, in the final application users were flagged who seen as spam, in future development the users will be given a soft ban and will be given a permanent ban if users continue with this behaviour.

I know there has been some scepticism in what companies were doing with this kind of biometric data but I know that this system would only use it for verification and would not store any data which would lessen security needed for GDPR. In this case, seen in Figure 2. 9, 85.2% of survey responses were yes meaning an overwhelming majority would be okay doing this in order to get rid of bot accounts on the social network, other options could be using captcha to eliminate bot accounts.

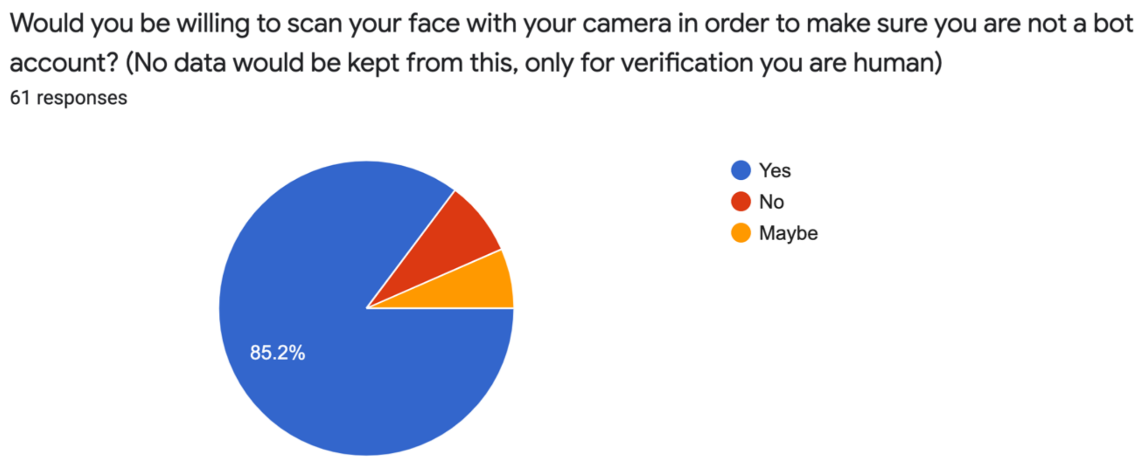
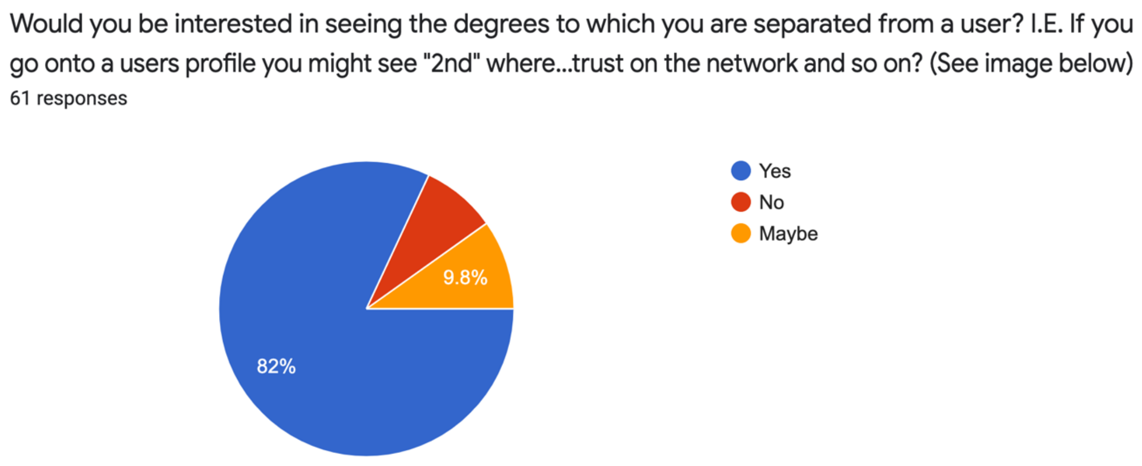

Would you be willing to scan your face with your camera in order to make sure you are not a bot account? (No data would be kept from this, only for verification you are human)
61 responses

Figure 2. 9 Survey – Face Scan Question

The last question I asked was to check if people would be interested in seeing the degrees of separation between them and other users, as if people didn't want this, I wouldn't implement it in the system but, as can be seen from the responses in Figure 2. 10, the majority of people would want to see this information.



Would you be interested in seeing the degrees to which you are separated from a user? I.E. If you go onto a users profile you might see "2nd" where...trust on the network and so on? (See image below)

61 responses

Figure 2. 10 Survey – Degrees of Separation Question

2.4. Existing Final Year Projects

In this section I will discuss some existing final year projects researched for this project.

2.4.1. Mongo Stratus: MongoDB Manager and Database Service Provider by Vlad Zat (20)

The goal of this project was to provide a flexible and easy-to-use web service for developers to use, in order to setup and manage MongoDB Databases. The idea would be to allow users to make these databases without having to set them up manually. The users would be able to utilize an easy-to-use interface where they can view, change and create data in their databases.

I think what makes this project so complex is the number of features that are worked with, some of which were no used before by Vlad, this included building a user interface for the user to be able to view modify delete data, creating an API that supports advanced querying accessed through a modern Web Application.

This project had four layered architectures with database tier, server-side tier, API tier and client-side tier. The strength of this project is the ease in usability and modern looking user interface for clients, this makes it very easy and attractive for clients to use this service instead of manually building their own Databases and would be very useful for people not as experienced in databases. The weaknesses of the project are in creating a secure connection to the database and restricting access to the database based on IP. This will give a lack of needed security for clients to use the service in confidents.

2.4.2. Detecting Bot Twitter Accounts using machine Learning by Emmet Hanratty (21)

A machine learning program that will be used to detect bot accounts on twitter, similar to my project this project's aim is to get rid of the large amount of spam and bot accounts on twitter and other social media platforms. This will be done to help differentiate these bot accounts from normal twitter accounts. There is a web-based interface where users can input URLs from twitter, and it will tell them if it is real or not.

The complexity of this project lies in the machine learning aspect of this, and the various different algorithms and the training required to be able to analyse and teach the system how to determine whether the user is a bot account or not.

At the logic tier there are four parts, Twitter API which is used to get data needed to pass to the Twitter Computation Section. Model Training which trains the Machine Learning model. Twitter Computation which determined whether or not it is a bot, and the algorithm results which outputs the result using JavaScript Chart. The presentation layer is done using Django for the Web UI Client and the project is developed in python using the Django Framework.

The Key strength to this project is the complexity of the training elements for machine learning.
The main weakness of this project would be that the follower prediction functions slowly, this could be due to the fact the accounts followers need to be fetched one by one and slow internet connection.

2.5. Conclusions

In this Chapter I discussed the research behind the different technologies that I could have used to develop this application. I discussed different languages and technologies at each tier, this included databases, server-side languages, front-end technologies. As well as this I discussed the different software methodologies I could have used in my application. Finally I discussed the survey results from my google forms survey and how they might impact on the design of the system.

Agile software development methodology was the decided software methodology, this is because it would be the most suited to this project because this project will initially start off with the basic functionality but a lot of new features will be needed as the system grows and it is important that these new features are reliable. User Satisfaction is also very important for this system and the user is always involved in Agile and this combined with the reduction of risk will allow for a better user experience.

From this research some choices were made in how the project would be designed. For the backend data tier, it was decided to use Neo4j database software, this is because of its extensive supports and forums, the familiarity with Java and the easy integration into applications. PostgreSQL was the decided database that would be used for storing users content, because of its ease of use and good documentation for support. It was also decided that Spring Boot API using Java would be used to for the logic tier of the application in order to facilitate communication between the front-end and the database. The front-end technology that will be used is React Js, as it is easily adaptable to different devices and is fast and scalable. As well as this the survey helped decide different features users would want to see, such as displaying the degree of separation, what users want to feel secure such as the actions taken on accounts that have been reported or have posted hate speech etc. The final spam detection algorithm was accurate for finding spam accounts with certain behaviours but will need to be tested once deployed in order to determine its accuracy properly, actions will then be added on accounts caught in the algorithm.

3. Design

In the previous chapter all of the research that went into the designing this system was discussed and in this chapter will be focusing on the conclusions and designs following all of this research. This will include the Software Methodology that was settled on along with diagrams including the software architecture diagram, graph database diagrams, use case diagrams and class diagrams, as well as the front-end design and screenshots of what the front-end system will look like.

3.1. Objectives

When considering the design of the application, it has a number of different important objectives and requirements to think about, as discussed in Chapter 1 the main objectives are:

- Create an Application that reflects the real world
- Establish trust connections between users
- Define levels of trust and type of trust between users
- Spam Detection algorithm
- Standard social media features

The first three objectives in this list all relate to each other in that establishing a way of implementing trust between users, trust levels and types, will help in creating an application that reflects the real world. This will be discussed further in section 3.4 how this will work in the database. Other important objectives are finding spam accounts, standard social media functionality, creating an interactive front-end for users to allow them to engage in standard social media functionality. This chapter discusses how the application has been designed in order to accomplish these objectives.

3.2 Use Cases

There were a number of use-cases needed in the initial designs of this application to show the basic functionalities within the system. These include:

- Logging In
- Registering
- Viewing Profile
- Editing profile

The *Logging in* use case seen in Figure 3. 1, includes authorization from the server to make sure the user's login details will match the database for their profile. Registration includes adding the login details and user information into the database and also includes a face verification process in order to verify the user is human in future. The last of the basic functionalities are grouped together. The user can view their profile and is given the option to edit their profile if they want, this is shown by the Extends arrow.

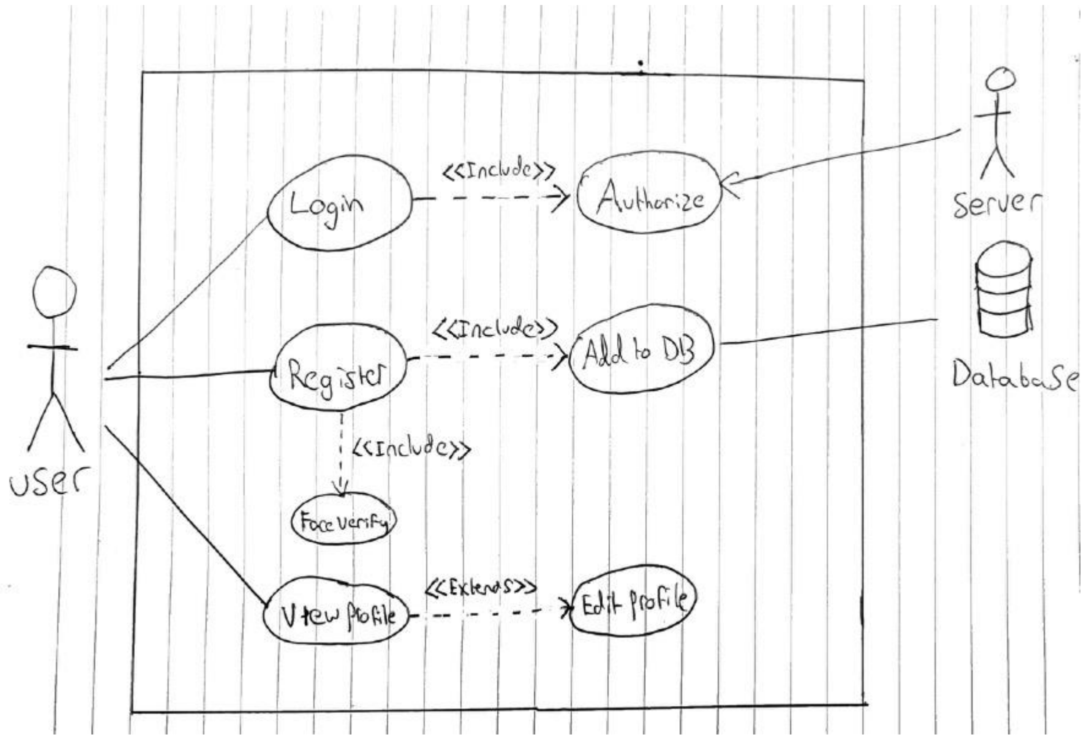


Figure 3. 1 Use Case for login, register and profile viewing

The next use case diagram shows the process of searching and creating connections with other users. You can see below in Figure 3. 2, User A can search for users and they have the option to send them a trust request or not. User B will receive a trust request and extending from the request is two options to trust or not trust user. If User B chooses to trust User A, they will form a trust connection and it will be added to the database.

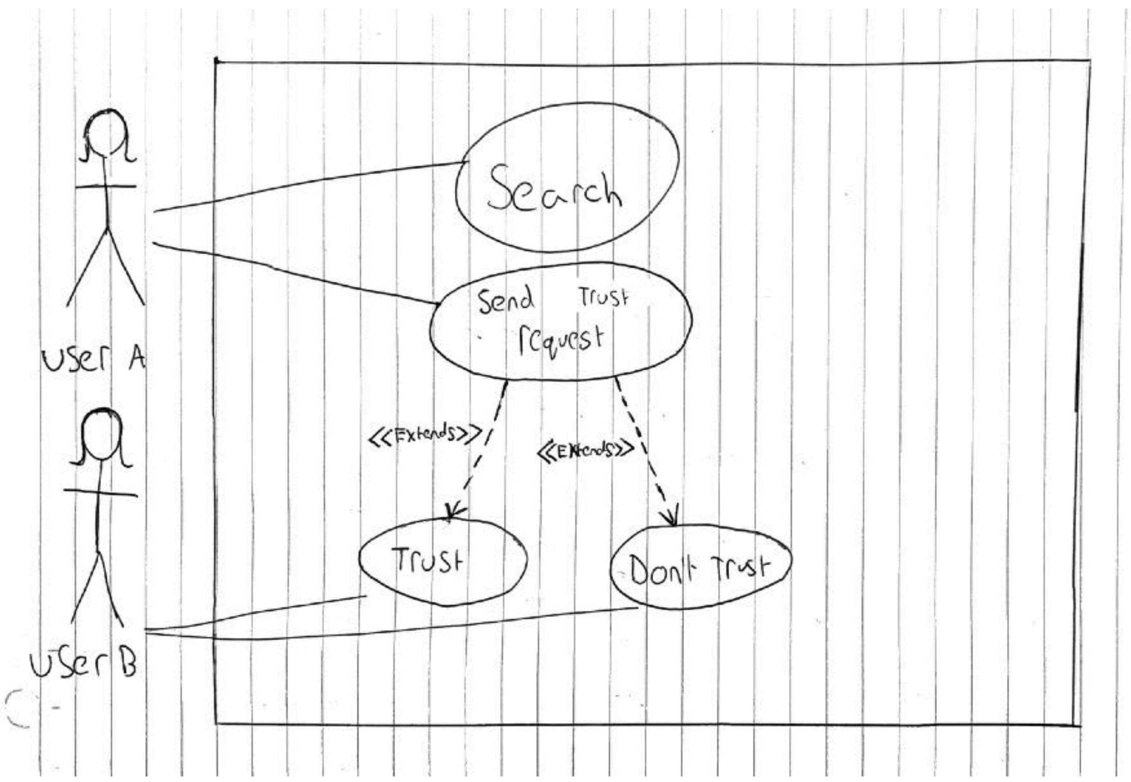


Figure 3. 2 Search Use Case

3.3. System Architecture

The system architecture as seen in Figure 3. 3 consists of four layers: the presentation layer, the API layer, the logic layer and the data layer. In the presentation layer you can see React JS. The API layer uses Spring Boot API coded in Java which will be used for client-server capabilities and communication between the user, the logic layer and the Database . The Data layer consists of Neo4j, as well as a relational database, PostgreSQL. The Neo4j Database is responsible storing users, their information and their relationships. The PostgreSQL database is responsible for the managing of user posts, messages and notifications.

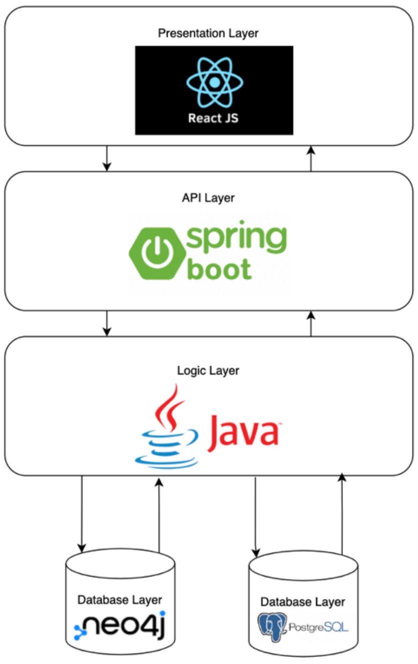


Figure 3. 3 Software Architecture

There were different important classes needed for the application. Starting with the four user Class diagrams in Figure 3. 4 that are the diagrams designed to accommodate for the database tables, in order to retrieve queries, update data, store new data etc. These hold the same attributes as the ER Table diagrams in Figure 3. 10, Figure 3. 11 and Figure 3. 12.

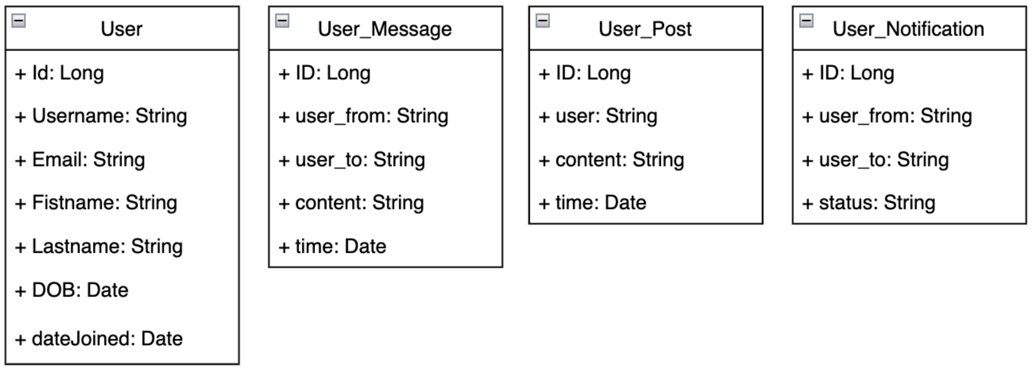
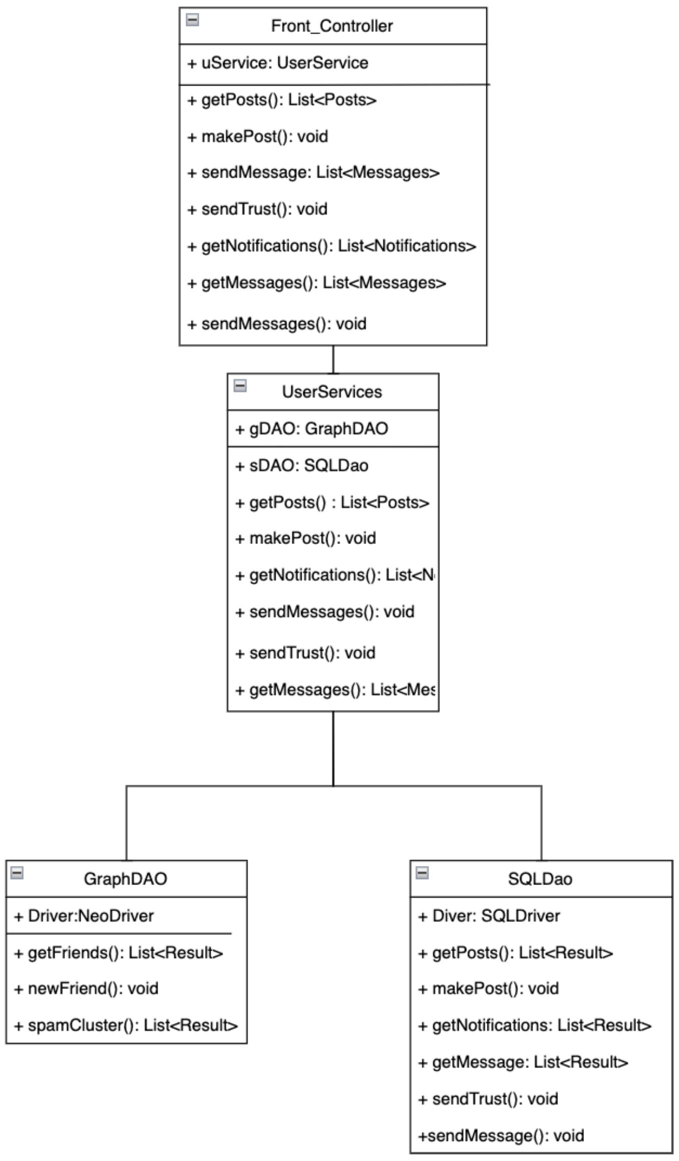


Figure 3. 4 User Class Diagrams

The last set of classes designed are seen in Figure 3. 5. They represent the programme logic needed for user requests from the front-end. The Front Controller will take in the API request and then will direct it towards a user service passing any needed parameters. The user service will then request this from the DAO (Data Access Object), will extract the results and return them to the front controller as lists of the necessary types for transmission to the front-end. This is the basic functionalities of the application although there was additions in the final system.



Front_Controller
+ uService: UserService
+ getPosts(): List<Posts>
+ makePost(): void
+ sendMessage: List<Messages>
+ sendTrust(): void
+ getNotifications(): List<Notifications>
+ getMessages(): List<Messages>
+ sendMessages(): void
UserServices
+ gDAO: GraphDAO
+ sDAO: SQLDao
+ getPosts() : List<Posts>
+ makePost(): void
+ getNotifications(): List<N
+ sendMessages(): void
+ sendTrust(): void
+ getMessages(): List<Mes
GraphDAO
+ Driver:NeoDriver
+ getFriends(): List<Result>
+ newFriend(): void
+ spamCluster(): List<Result>
SQLDao
+ Diver: SQLDriver
+ getPosts(): List<Result>
+ makePost(): void
+ getNotifications: List<Result>
+ getMessage: List<Result>
+ sendTrust(): void
+sendMessage(): void

Figure 3. 5 User Request Diagrams

3.4. Database Design:

In this section I will discuss the Database Designs for the application.

3.4.1 Graph Database:

This is a basic graph diagram representation of what the database would look like and how it would function in terms of trust connections. The users will connect with others on the network by creating trust relationships. This will create connections between nodes in the database. In Figure 3. 6 below you can see a straight-forward representation of the degrees of separation in the database when users create connections. At the base layer you can see a user who trusts eight different users, this is the first degree of separation or the direct trust connection on the network. The next degree is all of the users who are trusted by the previous degree of separation, e.g. the friends of your friends. This will continue to go out creating different degrees of separation from the centre user creating different degrees of separation that will be displayed on users profiles.

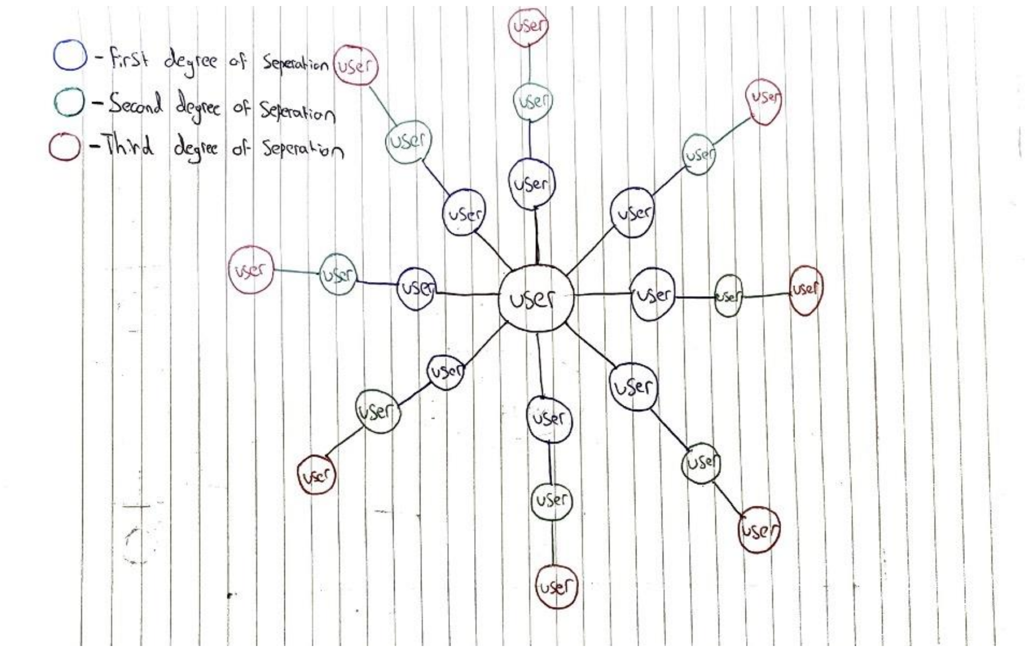

○ - First degree of Seperation
○ - Second degree of Seperation
○ - Third degree of Seperation
user

Figure 3. 6 Degree of Separation example using nodes

In some cases in the system users will have no trust connections, as you can see in Figure 3. 7 below. The users John, Tim and Ella all have trust connections and let’s say for example one of them stumbles upon Conor, they will get a warning that he has no trust connections. Conor in this case would need to build his connections in order to be seen as more trustworthy on the network, this is an interesting problem and a problem that may be seen across all social media applications, where users have no following and no credibility at the beginning. It’s definitely worth considering in future possible other ways to give users the initial credibility, such as verification ticks using ID verification, facial recognition etc.

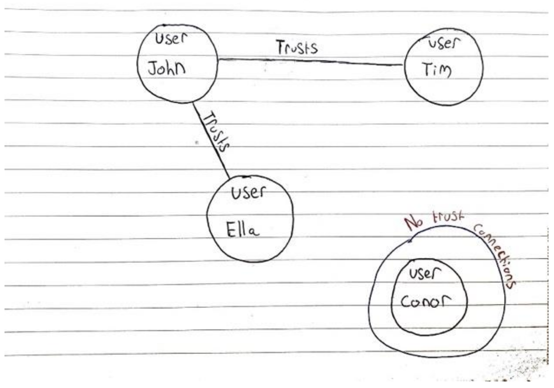


Figure 3. 7 No Trust Connections Example

I did consider alternative seen in Figure 3. 8 for the databases diagram, where Users would be able to see the content and profiles of their friends' friends instead of just their friends, but after that they would need to connect with other users to see their content. This is a possible future option for the user if they would want to see this or allow this for their profile.

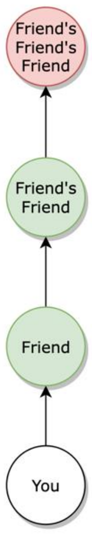


Figure 3. 8 Alternative User Access Design

A big part of my project relies on the idea of trust between users and authenticity of accounts within the network to allow for a safer environment without spam or so called 'Trolling', i.e. users who would hide behind or blank accounts and send hateful comments or try to provoke other users for their own enjoyment. This meant figuring out a way to get rid of accounts that were created for these reasons had to be considered.

One of the ways this was done was by using graph network analysis to find spam clusters. As you can see from Figure 3. 9 there are two different groups of nodes with connections to each other, on the left you can see legitimate users and their connections and on the right you can see a block of spam or fake accounts. This shows a way an algorithm could be created to detect these accounts. In the diagram all of these accounts trust each other and no one else, if one is found to be a spam account they can all be found easily by looking at their connections. There is also the issue of people accidentally trusting spam accounts but other parts of the algorithm like looking at patterns common in spam users and also dates they have joined etc would help with this.

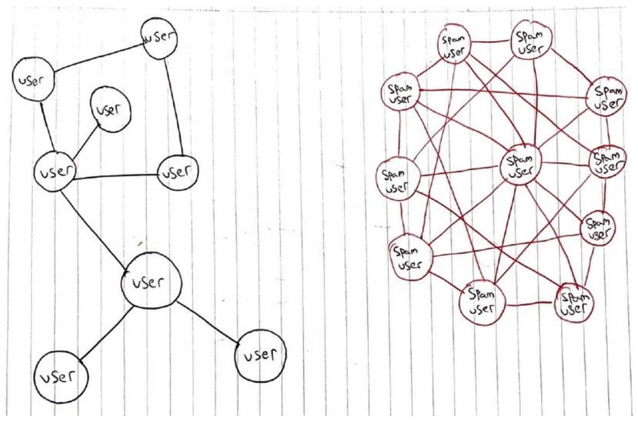


Figure 3. 9 Normal vs. Spam User

3.4.2. Relational Database:

As part of this project there was a need for storing data such as messages, posts and notifications. These are all essential parts of a social network. The solution to this problem was to use a Relational Database. The reasoning behind this is that for spam detection using community detection and graph theory algorithms it was needed to use a graph database because that is what they do best. However, although I could have possibly implemented messages, posts and notifications within the graph database I decided to use a Relational database, as this is the area where relational databases work best.

To do this a number of entity relational tables we're created starting with the user messages table shown in Figure 3. 10. This table has five different attributes "ID", "user_from","user_to","message" and "time". These elements hold all the necessary information to store messages between users.

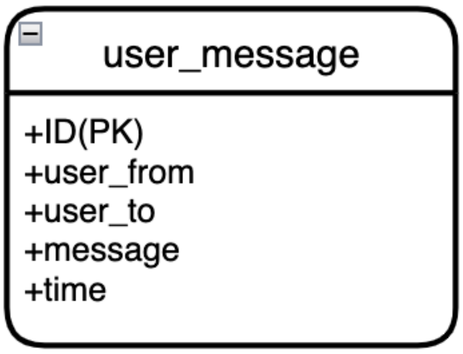


Figure 3. 10 User Message ER table

Next the user notification table was designed, seen in Figure 3. 11, which consisted of four attributes "id","user_from","user_to" and "status". The purpose of this table was for sending trust requests in particular so there was no need to store a type of notification. The "status" attribute would be used in order to determine if a notification was rejected and this could be useful in terms of the spam detection which will be discussed later.

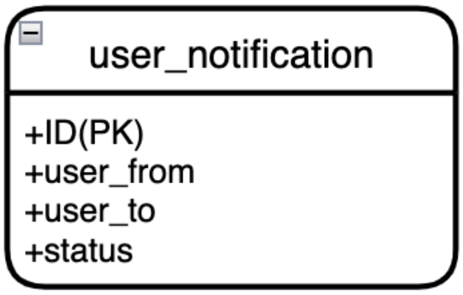


Figure 3. 11 User Notification ER table

The final table, seen in Figure 3. 12, was the user posts table which has a simple structure consisting of a n "ID", "user","content" and "time". These were the only attributes needed to store posts although there could be further adaptation to this to accommodate for likes, comments etc. These tables have no connections to a user's table as the users are stored in the graph database, however, referential integrity is enforced through the application layer.

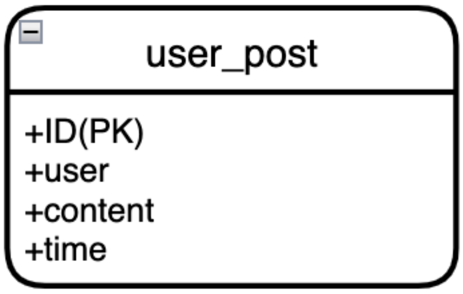


Figure 3. 12 User Post ER table

3.5. Front-End Design:

This section will discuss the design for the front-end and show the side-by-side of the final application compared to this design. The designed flow of the final application will also be discussed.

Starting with Figure 3. 13, when the user is logged out, they will see this screen. On the left we see the design for this screen and on the right we see what the final application looks like for this screen. There is one button on this screen that is used to login which will redirect you to Figure 3. 14 which is where the user will login into the application.

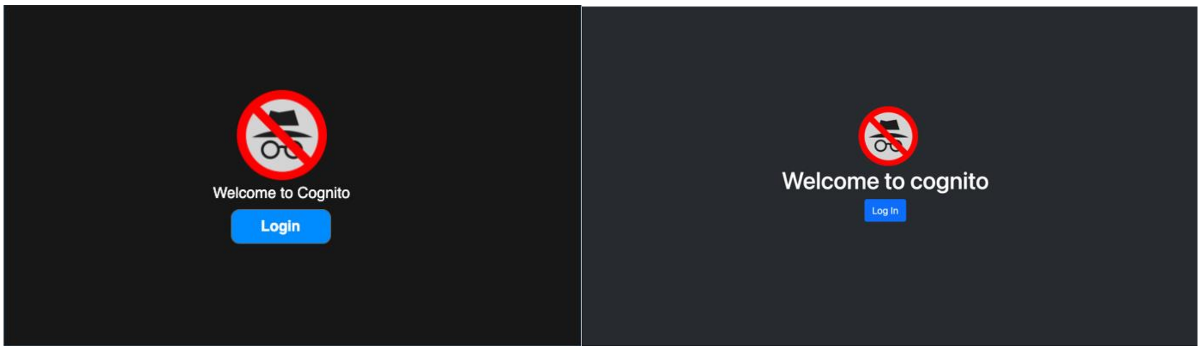


Figure 3. 13 User Logged out Example – Design on left, Final on right

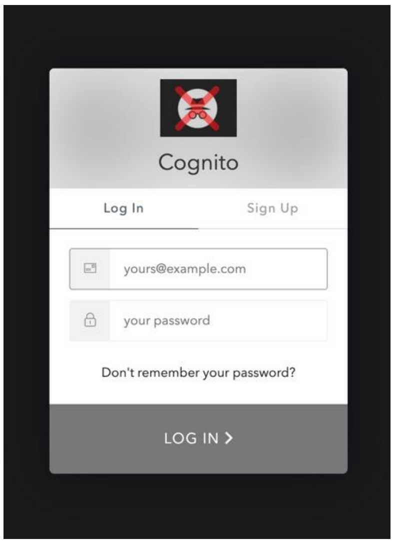


Figure 3. 14 Login Box

Once logged in, users will see a Navbar on the top of the page with a number of options for navigation and other buttons. The navigation through different components of the web app is done via the left side of the Navbar, when the user clicks the dropdown button labelled 'Cognito' they will have a number of options to navigate to depending on whether they are an admin user or regular user. Beside this you will see a search input and this is used for searching for users in the application and will navigate to a search results page. On the right hand side of the navbar there is a link to your own profile of the user that is currently logged in as well as a notification bell to see if the user has any trust requests. The last button on this Navbar is the Logout button which will log the current user out of the application. We can see the design of this in Figure 3. 15 and how this compared to the final application in Figure 3. 16

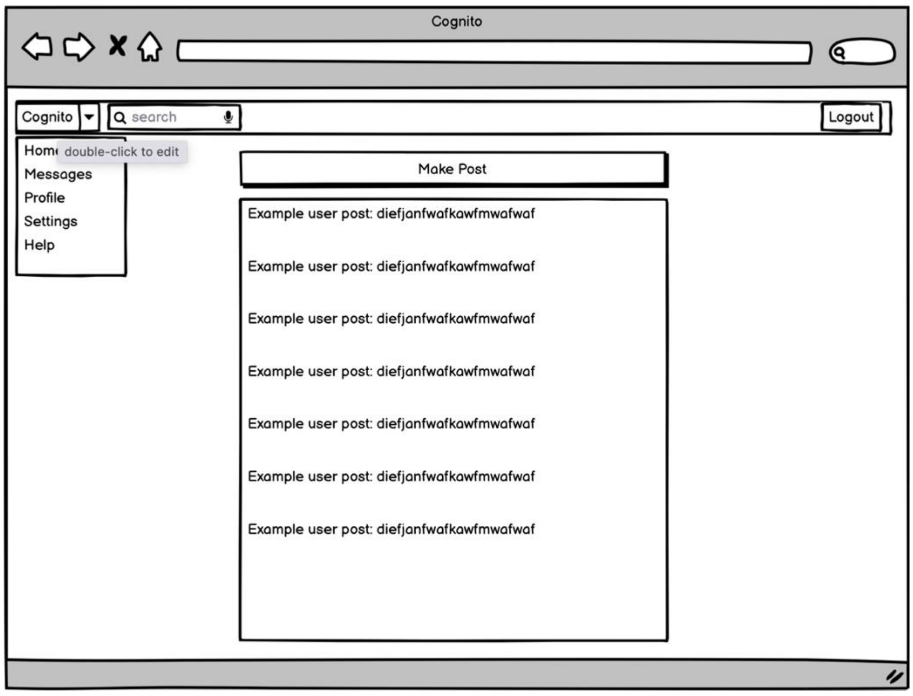
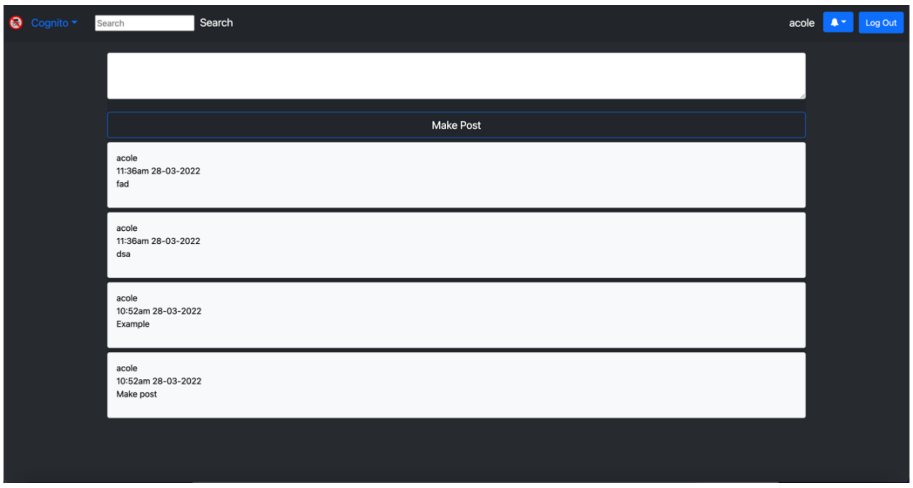

Cognito
Cognito
search
Logout
Home
double-click to edit
Messages
Profile
Settings
Help
Make Post
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf

Figure 3. 15 Home with Navigation



acole
11:36am 28-03-2022
fad

acole
11:36am 28-03-2022
dsa

acole
10:52am 28-03-2022
Example

acole
10:52am 28-03-2022
Make post

Figure 3. 16 Final App Home

For the Search component users are prompted to type queries into the search input field on the Navbar. When the user searches they will be redirected to a Search Results page as shows in the wireframe Figure 3. 17. This will show the list of users that have first names, last names or usernames matching the inputted query. Users will see the cards of basic information about the users returned from the query and can click in to view that specific users profile in more detail. We see what this looks like in the final application in Figure 3. 18.

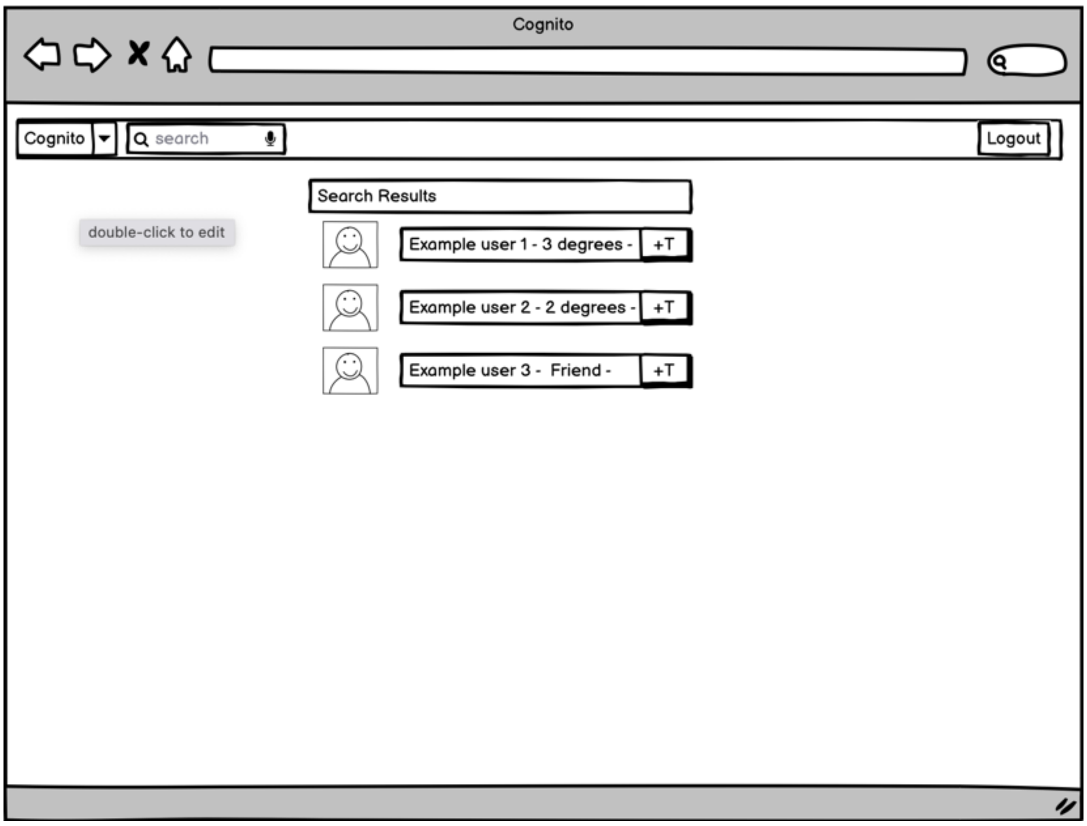
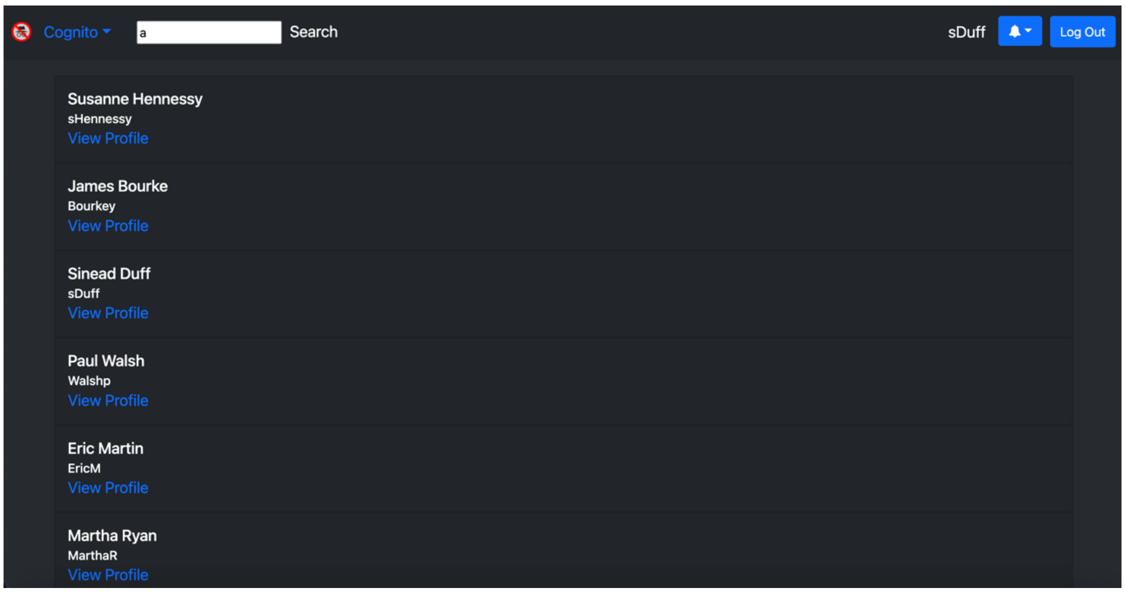

Cognito
Cognito
search
Logout
Search Results
double-click to edit
Example user 1 - 3 degrees -
+T
Example user 2 - 2 degrees -
+T
Example user 3 - Friend -
+T

Figure 3. 17 Search Results Page



Susanne Hennessy
sHennessy
View Profile

James Bourke
Bourkey
View Profile

Sinead Duff
sDuff
View Profile

Paul Walsh
Walshp
View Profile

Eric Martin
EricM
View Profile

Martha Ryan
MarthaR
View Profile

Figure 3. 18 Search Results Final App

The last front-end wireframe is shown in Figure 3. 19. This shows the profile view when users look at other user profiles and also their own. The degrees of separation are shown on the right if the user is not the current suer logged in otherwise it shows a map of the current logged in user’s friends. There will be options to send trust request to users and possibly other interactions as well such as blocking, reporting and (if connected already) removing that connection.

We can see how this design in the final application adapts to the profile the user is viewing. The map on the right-hand side of the screen shows a graph displaying all of the user's friends if viewing their own profile (Figure 3. 20). If viewing another user's profile, a *separation map* is shown, allowing the user to see how they are connected to the profile in question (Figure 3. 21). Users at more than 5 degrees of separation from each other are not considered connected. From this screen, users can edit their own profile, untrust a connected user or send a trust request to an unconnected user (Figure 3. 22). This screen also allows users to report other users to be flagged for follow-up by an admin.

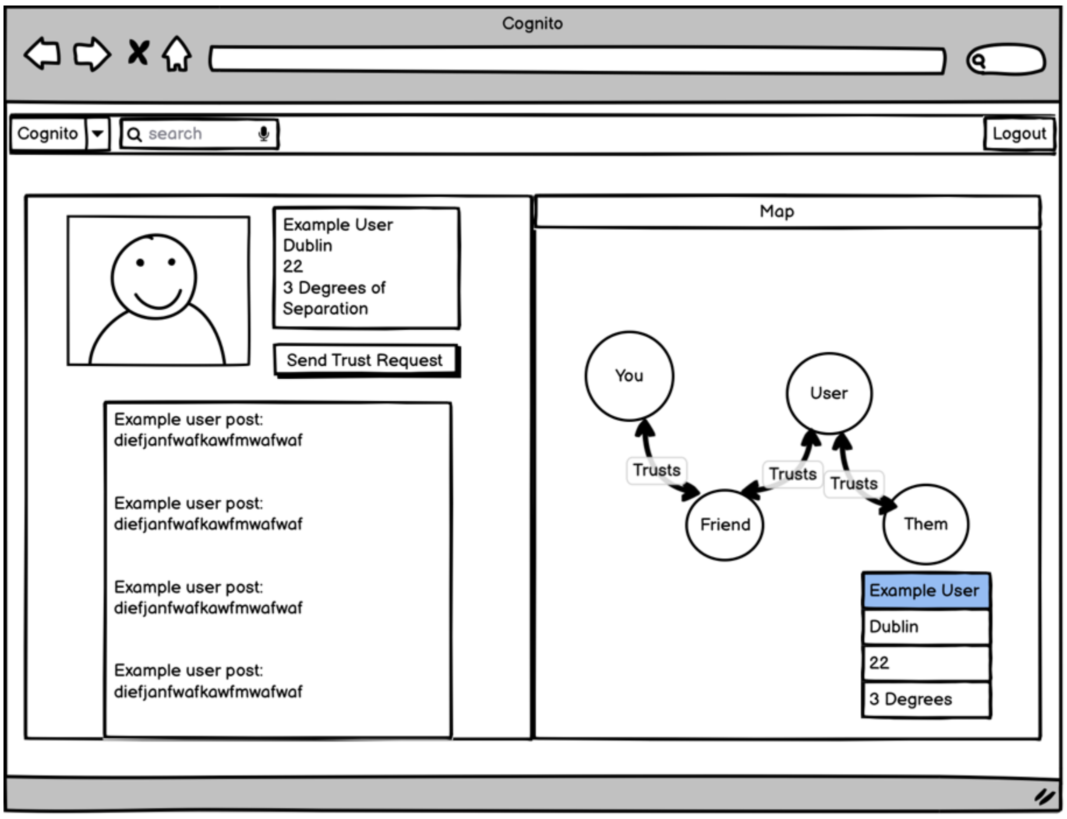
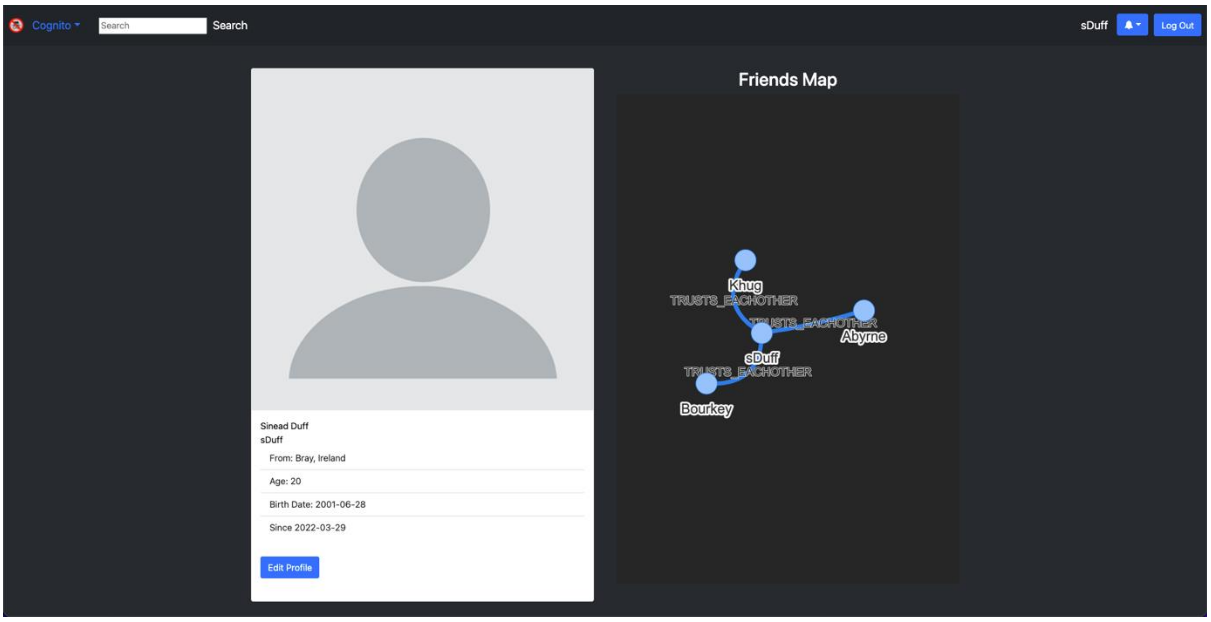
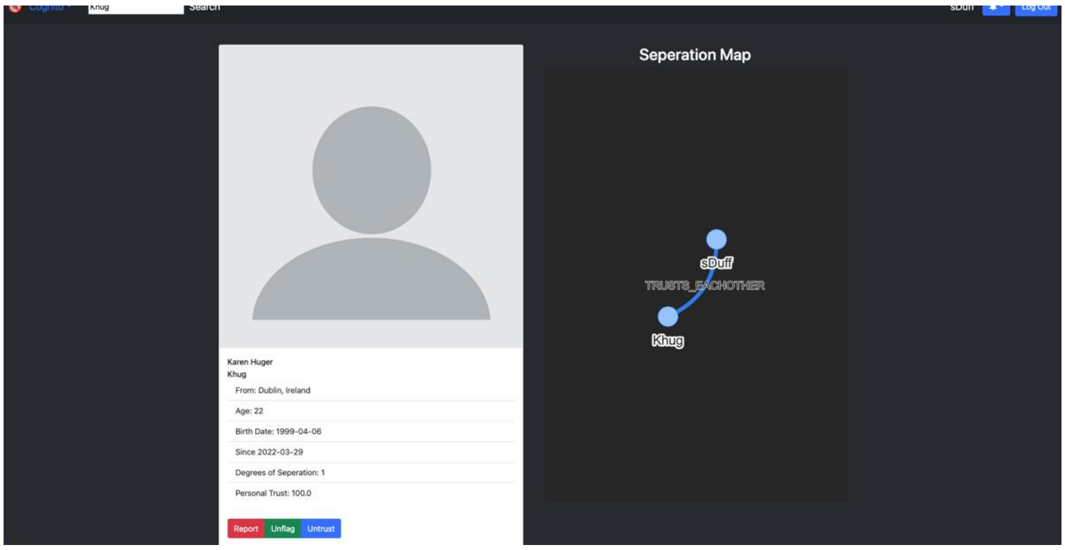

Cognito
Cognito
search
Logout
Example User
Dublin
22
3 Degrees of Separation
Send Trust Request
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Example user post: diefjanfwafkawfmwafwaf
Map
You
User
Trusts
Trusts
Trusts
Friend
Them
Example User
Dublin
22
3 Degrees

Figure 3. 19 Profile View



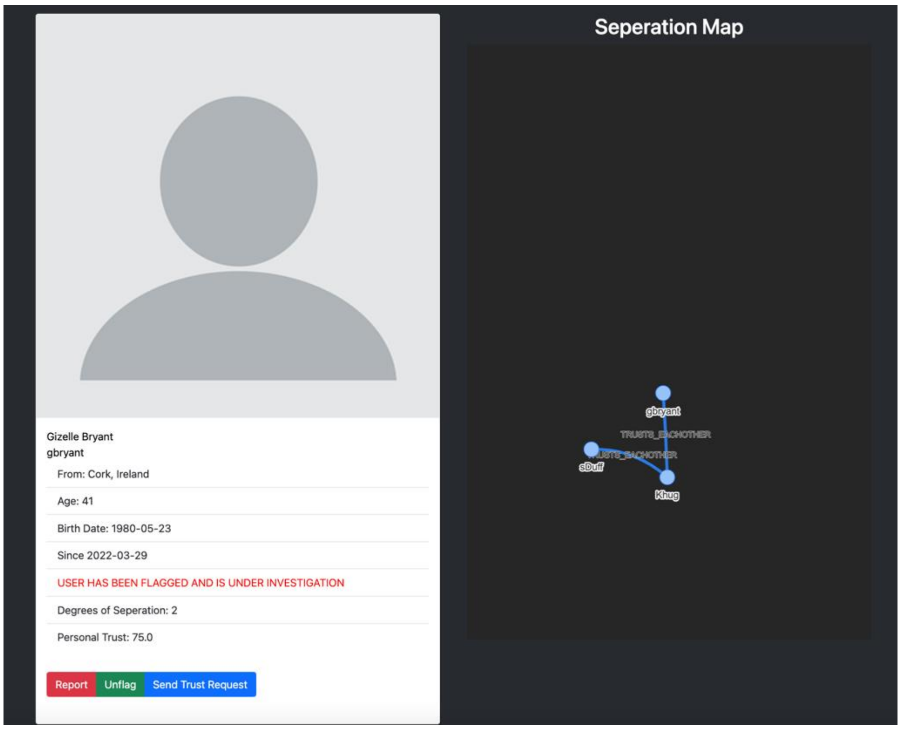
Sinead Duff
sDuff
From: Bray, Ireland
Age: 20
Birth Date: 2001-06-28
Since 2022-03-29

Figure 3. 20 Users own profile Final App



Karen Huger
Khug
From: Dublin, Ireland
Age: 22
Birth Date: 1999-04-06
Since 2022-03-29
Degrees of Seperation: 1
Personal Trust: 100.0

Figure 3. 21 Trusted Connections Profile Final



Gizelle Bryant
gbryant

From: Cork, Ireland

Age: 41

Birth Date: 1980-05-23

Since 2022-03-29

USER HAS BEEN FLAGGED AND IS UNDER INVESTIGATION

Degrees of Seperation: 2

Personal Trust: 75.0

Figure 3. 22 User not trusted Profile Final

3.6. Conclusions

This chapter looked at the design of the system, beginning at a high level view of the architecture, and on to look at the design of the data tier, logic tier and front-end in detail. The identified use-cases motivating the overall app design were explored in detail. In the next chapter I will be discussing the development process based on the designs discussed above.

4. System Development

This chapter will discuss the final application that has been developed at each tier of the multi-tiered architecture. This will include code snippets and database visuals. As well as this there will be discussion on the changes of the system from the initial design and why these changes took place. The link to the GitHub Repository of this project can be found at https://github.com/Eoghan-2000/cognito.

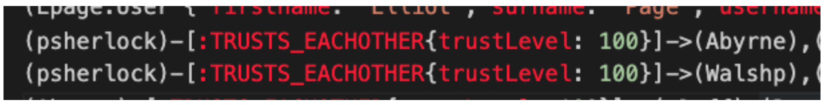
4.1 Data Tier

In this section I will be discussing the database layer and the reasons behind the database designs that were implemented for the application. The database was a key part of this project as the project relies heavily on the use of graph databases and graph theory as well as relational databases for important social media application functionality. This was implemented using Neo4j, a graph database software and PostgreSQL, a relational database. As well as this the Auth0 database was used to store users credentials.

4.1.1. Graph Database Development:

In this sub-section the implementation of the graph database using Neo4j will be discussed. Neo4j provides amazing data visualisation within the application and this allowed me to see what users would look like on the system and show different node relationships such as spam relationships, normal relationships and nodes that didn't have any relationships

Users were given attributes that would be useful for spam detection algorithms and patterns and also information that they might usually have on their social media platforms. These included first name, surname, username, email, date of birth, date joined, age, location and flagged. Then trust relationships were created between user nodes and gave them a trust level of 100 between directly trusted users as seen in the cypher query in Figure 4. 1. This was created for use with the spam cluster algorithm discussed in Section 4.3.1.



```
(psherlock)-[:TRUSTS_EACHOTHER{trustLevel: 100}]->(Abyrne),
(psherlock)-[:TRUSTS_EACHOTHER{trustLevel: 100}]->(Walshp),
```

Figure 4. 1 Trust Defined for test users

As seen below in Figure 4. 2, you can see the output you get when you return all users after they are created and their relationships are defined. On the left you can see a cluster node, I created these relationships in the database so every node within the cluster trusts each other. On the far right, you can see four seemingly normal users, two having only one connection and two having none. On the right of the cluster in the middle of the image there is five normal users that have some similar connections as well as one connection each to the spam cluster.

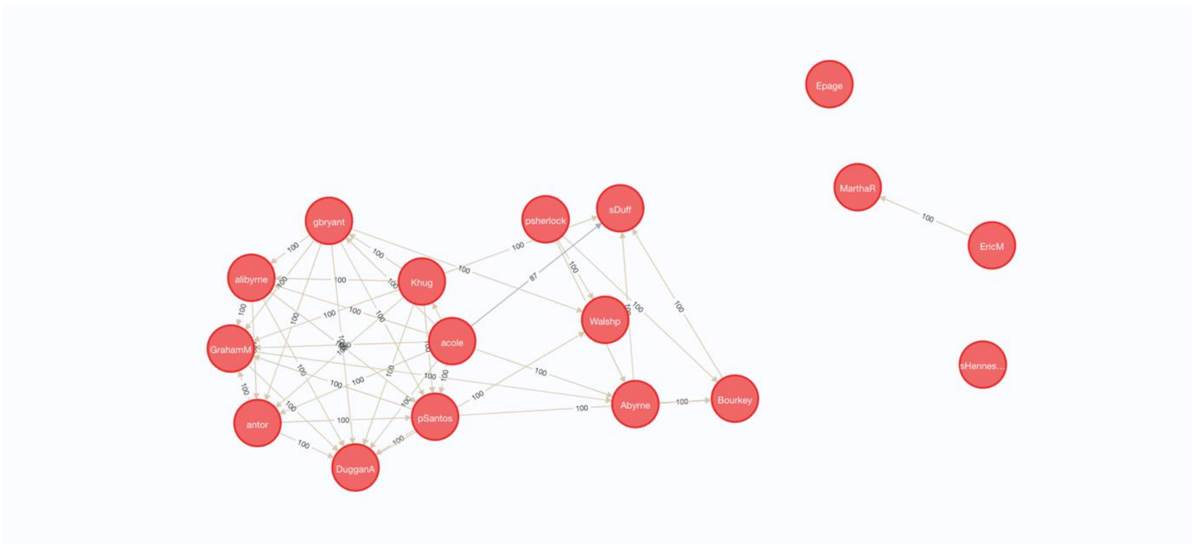
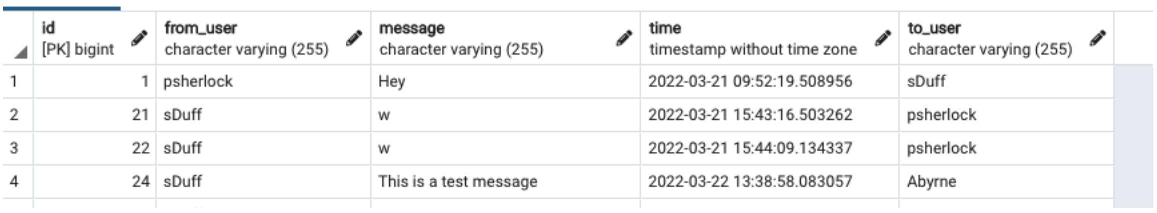


Figure 4. 2 All Users in the databases after insertion of example data

4.1.2. Relational Database Development:

PostgreSQL was also used in database development and the table creation was implemented using Spring API which will be discussed in Section 4.2. The design for the tables was important for the purpose of use within the system when attempting to get or store posts notifications and messages between users.

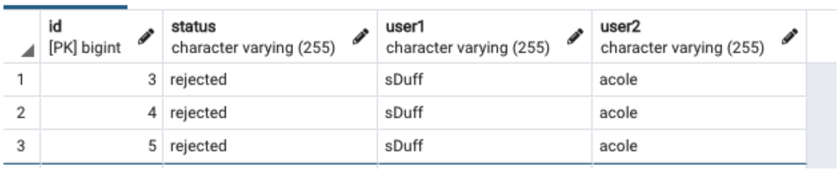
The messages table needed a total of five columns including the ID, from_user, to_user, message and time. As seen below in Figure 4. 3, we can see some example message rows from the database. This shows some message exchanges between users.



	id [PK] bigint	from_user character varying (255)	message character varying (255)	time timestamp without time zone	to_user character varying (255)
1	1	psherlock	Hey	2022-03-21 09:52:19.508956	sDuff
2	21	sDuff	w	2022-03-21 15:43:16.503262	psherlock
3	22	sDuff	w	2022-03-21 15:44:09.134337	psherlock
4	24	sDuff	This is a test message	2022-03-22 13:38:58.083057	Abyrne

Figure 4. 3 User Message Data example

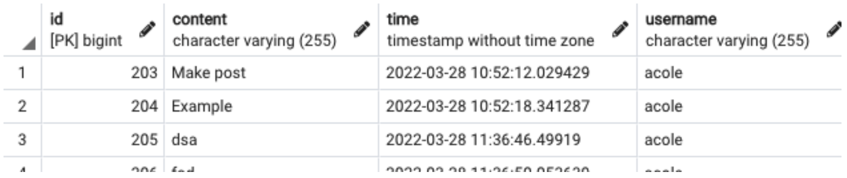
The user notifications data as seen in Figure 4. 4, is an example of what some of the rows look like in the database, it includes an ID, status, user1,user2(which represent who is sending and receiving the notification). This was quite a simple table which initially just had the two users, but then had the addition of status in order to catch spam users sending multiple trust requests to users, this will allow rejected notifications to be counted in order to see accounts with a lot of rejections which would indicate them as spam users. When the notification first is added to the database the status is pending, when a notification is declined the status will be 'rejected'.



	id [PK] bigint	status character varying (255)	user1 character varying (255)	user2 character varying (255)
1	3	rejected	sDuff	acole
2	4	rejected	sDuff	acole
3	5	rejected	sDuff	acole

Figure 4. 4 User Notifications Data example

The final table is the User Post table which as seen in Figure 4. 5, was quite simple in structure it has a post ID, users name, content and time. This was all the necessary columns needed to implement posts.



	id [PK] bigint	content character varying (255)	time timestamp without time zone	username character varying (255)
1	203	Make post	2022-03-28 10:52:12.029429	acole
2	204	Example	2022-03-28 10:52:18.341287	acole
3	205	dsa	2022-03-28 11:36:46.49919	acole

Figure 4. 5 User Post Data example

4.1.3 Issues

There were some issues encountered when creating and populating these databases. Some the main issues was unfamiliarity with graph cypher query language in the Neo4j software.

Neo4j was a tool with which there was learning involved, this is due to the lack of experience using graph databases. However there is a lot of Neo4j documentation including videos, articles, and examples started queries to test yourself. These proved to be extremely important when it came to learning the cypher query language as well as getting in-depth knowledge of how graph databases work and how to take full advantage of the differences between them an relational databases.

4.2 Middle API Tier

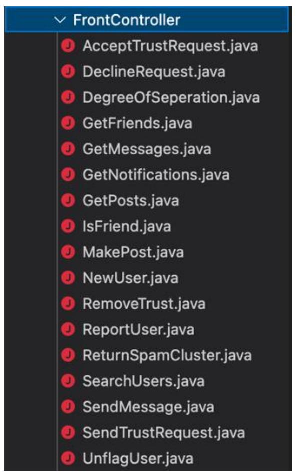
The Middle Tier of the system was developed using Spring REST API which is written in Java. The reasoning behind this middle tier was to facilitate communication between the user interface and the logic-layer. Any user requests come through the front end into the Front Controller classes using Spring and this section the development and reasoning behind using this API. Spring was a new technology undertaken for this project, so there was a learning curve, but because of a lot of previous experience with Java, it was not too difficult to learn.

4.2.1 Spring Boot API

Spring Boot API is a Restful API that allows for a number of different service for web applications including GET, POST, PUT, DELETE actions, Caching, Redirection and forwarding and security. The main features that I used in Spring was the actions in handling requests to get, post, modify and delete data from the front end as well as Database repository interfaces for both Graph and Relational Databases.

4.2.2 Implementing Rest Controllers

As part of the system it was important to have front controllers which take in HTTP requests from the front end and manage what to do with the requests. As seen in Figure 4. 6. the controller files are all in the FrontController Directory. These contain a number of different classes that represent different requests.



```
FrontController
  AcceptTrustRequest.java
  DeclineRequest.java
  DegreeOfSeperation.java
  GetFriends.java
  GetMessages.java
  GetNotifications.java
  GetPosts.java
  IsFriend.java
  MakePost.java
  NewUser.java
  RemoveTrust.java
  ReportUser.java
  ReturnSpamCluster.java
  SearchUsers.java
  SendMessage.java
  SendTrustRequest.java
  UnflagUser.java
```

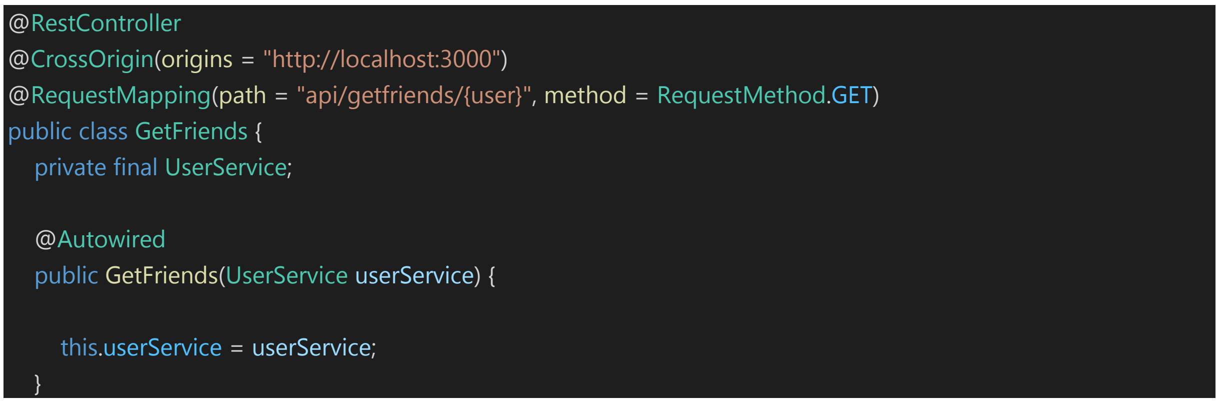
Figure 4. 6 Front Controllers

Within these controllers there are a number of different requests including GET and POST requests. The class structure for these are as seen in Figure 4. 7. The Controller is denoted by the '@RestController' which defines the class as a controller class. There are a number of key elements when it comes to this controller class, starting with the '@CrossOrigin' annotation which defines what addresses are able to request this information from the controller. This is set to the localhost address on which the React Web application is running in this case and this would change in production when the application is deployed.

The next annotation that is needed is ‘@RequestMapping’ which is used to define what request is being called from the front-end. This consists of a path which holds the address of the request so the Spring API knows where to go for the request. Within this address one of attributes denoted by the curly brackets, {user}, allows for when the front-end sends in specific attributes to get specific data, in this case it is the user that is logged in in order to find all of their connections in the database. As well as this the method is defined which tells the API what sort of request it is, in this case a GET request.

The ‘@Autowired’ annotation is used in order to enable dependency injection for a component or feature, this means that you can declare a variable and leave it up to spring to actually create it. For example, the userService class, which is required by the GetFriends controller shown in Figure 4. 8, is automatically passed into the controller whenever a request comes in.

The method with the ‘@GetMapping’ annotation is the method that will be executed whenever a GET request is made and the ‘@PathVariable’ annotation can pull attributes from the path that is needed to execute the function. The main job of the @PathVariable annotation is to determine whether the function should run (by checking the URL the user accessed and seeing if it matches the path specified in the annotation). This Controller will call the UserService class to return a list of strings containing all the usernames of the connections of the passed in user. There are also other controller types such as POST controllers which do not return data to the Front-end.



```
@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping(path = "api/getfriends/{user}", method = RequestMethod.GET)
public class GetFriends {
    private final UserService;

    @Autowired
    public GetFriends(UserService userService) {

        this.userService = userService;
    }
    //calls the user service method to get notifications for a particular user
    @GetMapping
    public List<String> getUserFriends(@PathVariable("user")String user){
        List<String> uList= userService.getUserFriends(user);
        return uList;
    }

}
```

Figure 4. 7 GetFriends Get Controller

There are 19 controller classes in total and they are similar to the controllers shown above, such as ‘DeclineRequest’, ‘DegreeOfSeperation’, ‘SearchUser’ and more that are listed in the directory shown in Figure 4. 6.

4.2.3 Service implementation

There are two services that are associated to both the User Relational Database Services and the User Graph Database Services. These are as shown in Figure 4. 8 and Figure 4. 9. These are used to handle services for both the Relational and the Graph Database requests. As seen below you can see that the Graph user service class has a number of different DAOs which access the graph database DAOs but as well some of the relational database DAOs this is only because they use the results from the Graph DAO to then use that information for different steps in the Spam Detection Algorithm.

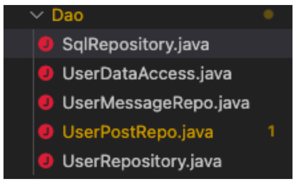
```
@Service
public class UserService {

    @Autowired
    UserDataAccess userAccess;

    @Autowired
    UserRepository uRepo;

    @Autowired
    SqlRepository sRepo;

    @Autowired
    UserPostRepo uPostRepo;

    @Autowired
    SqlRepository uNotifications;
```

Figure 4. 8 Graph Services Class

```
@Service
public class UserSqlService {
    @Autowired
    SqlRepository notifcationRepo;
    @Autowired
    UserPostRepo postRpo;
    @Autowired
    UserMessageRepo messageRepo;
```
Figure 4. 9 Relational Services Class

4.2.4 DAO Repositories

Another useful service within spring boot API is the Database Repositories which work with both graph and relational databases. In this system there are four Database repositories, one for the graph Database and three for the Relational Database as seen in the DAO directory in Figure 4. 10. These work to implement built-in queries as well as custom queries to different tables or entities in the databases. As well as this I have one user access class that uses a '@RestController' annotation to manage more complex Graph database queries.



```
Dao
  SqlRepository.java
  UserDataAccess.java
  UserMessageRepo.java
  UserPostRepo.java        1
  UserRepository.java
```

Figure 4. 10 DAO Directory

Within the Relational Database Repositories, they have similar structure as seen in Figure 4. 11. This is an interface which extends the built-in spring 'CrudRepository' interface. This interface takes two attributes one being the class with the table details in which you are trying to query and the other being a 'Long' to represent the ID column of said table. Once this is set you are able to build custom queries as well as use built-in queries like 'save' which inserts new data of a certain type into the database.

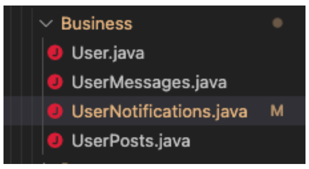
The annotation '@Query' allows you to write custom SQL queries to the database and enable you to pass in parameters to the query with ease. These are the methods that get called from the service classes. When modifying or deleting data its needed to add some extra annotations '@Transactional' and '@Modifying' to let spring know you are you are modifying or deleting data within the database. There are repositories like below for each table within the relational database.

```
public interface SqlRepository extends CrudRepository<UserNotifications,Long>{

    @Query( value = "Select * from user_notifications where user1 = ?1 and status = \'pending\'", nativeQuery =
true)
    List<UserNotifications> getNotifications(String User);

```

```
   @Transactional
   @Modifying
   @Query(value="Delete from user_notifications where user1 = ?1 and user2 = ?2", nativeQuery= true)
   void deleteNotification(String username1, String username2);

   @Transactional
   @Modifying
   @Query(value="update user_notifications set status = \'rejected\' where user1 = ?1 and user2 = ?2 ",
nativeQuery= true)
   void declineNotification(String username1, String username2);

   @Query(value = "Select user2 from user_notifications where user2 in :s and status=\'rejected\' group by user2
having count(*) > 6", nativeQuery = true)
   List<String> getSusNotifications(List<String> s);
```
Figure 4. 11 Relational Repository

4.2.5 User Classes with Spring

Spring also has the capabilities to autogenerate tables within its framework allowing you to use them for the Repositories. In the system there are four User classes relating to the user posts table, user messages table, user notifications table and the graph databases user node, as seen in the directory in Figure 4. 12.



```
Business
  User.java
  UserMessages.java
  UserNotifications.java   M
  UserPosts.java
```

Figure 4. 12 Business Directory

In Figure 4. 13 we see the User class which is similar to a normal user class except we add spring annotations in order to represent the different database attributes. The ‘@Node’ annotation is used to describe the user node within the graph database. We see a number of annotations below in the class attributes which represent the node attributes on the graph database this is signified with ‘@ID’ annotation which specifies the ID of the node and ‘@Property’ annotation which represents the node attributes and their names.

As well as this we can see ‘@Transient’ which is used because of the auto calculation of the age using the date of birth which allows us not to have to store ages within the database. The last annotation ‘@Relationship’ is used to define relationships that users will have with other users.

```
@Node("User")
public class User {
    @Id @GeneratedValue
    private Long id;
    @Property("username")
    private String username;
    @Property("firstname")
    private String firstname;
    @Property("surname")
    private String surname;
    @Property("dateTimeJoined")
    private LocalDate dateTimeJoined;
    @Property("dob")
    private LocalDate dob;
    @Property("location")
    private String location;
    @Property("Email")
    private String email;
    @Property("flagged")
    private Boolean flagged;
    @Transient
    private int age;
    @Relationship
    private List<User> trusts = new ArrayList<>();
```
Figure 4. 13 User Class

The other type of class is the Relational Database Entity classes. These are the classes that autogenerate tables if they do not already in the database with the necessary columns. They are marked by the annotation '@Entity' and instead of property annotations they have '@Column' annotations for the attribute types. Figure 4. 14 shows the UserMessages class; there are two other classes similar to this to represent user posts and user notifications.

```
@Entity
public class UserMessages {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    @Column(name = "From_user")
    private String From;
    @Column(name = "To_user")
    private String To;
    @Column(name = "Message")
    private String Message;
    @Column(name = "Time")
    private LocalDateTime Time;
```
Figure 4. 14 User Message Class

4.2.6 Issues

Within the Spring framework there was a lot of learning to be done when it came to how exactly classes need to be structured and the annotations needed for certain methods and attributes. This didn’t cause too many issues as there was a lot of documentation in the Spring website (22).

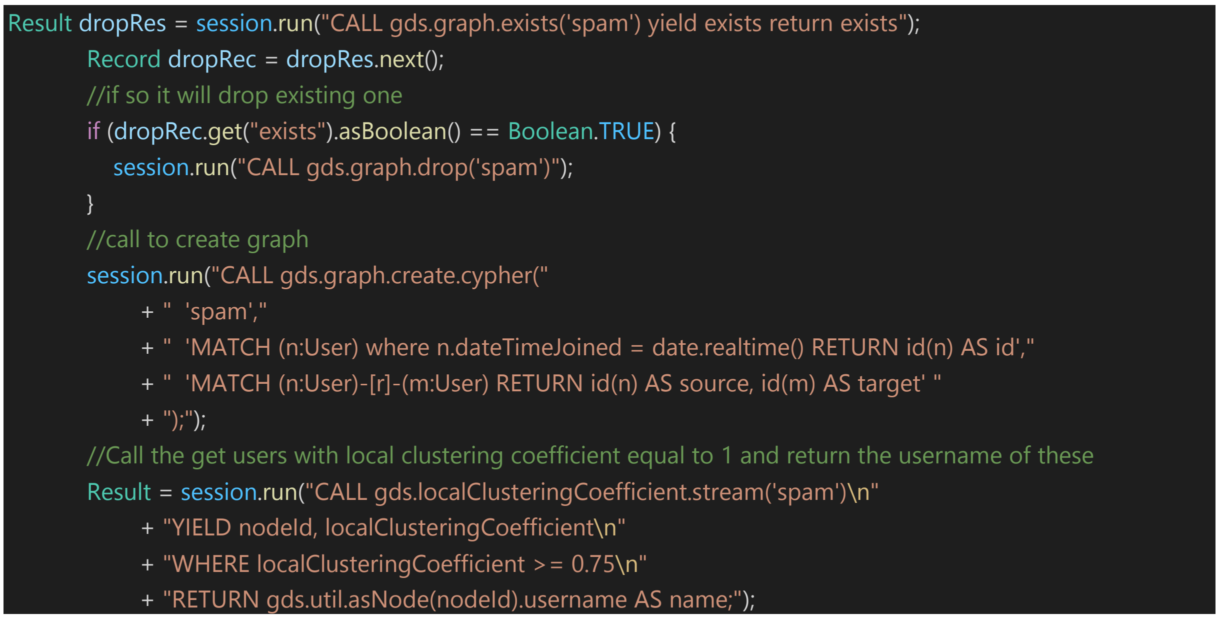
One of the main issues was problems with knowing how to differentiate different types of requests in the front controller and different types of queries. The issue came up a number of times but was easily solved through research and examples.

4.3 Logic Tier

The Logic Tier of the application consists of the methods and algorithms used within the backend. This tier works with Spring services to get data from the database and uses this data to implement other parts of the spam cluster algorithm as well as formatting this data for returning to the user. Since spring is implemented in Java all of this logic is written in Java with Cypher and SQL query languages to retrieve results.

4.3.1 Spam Detection

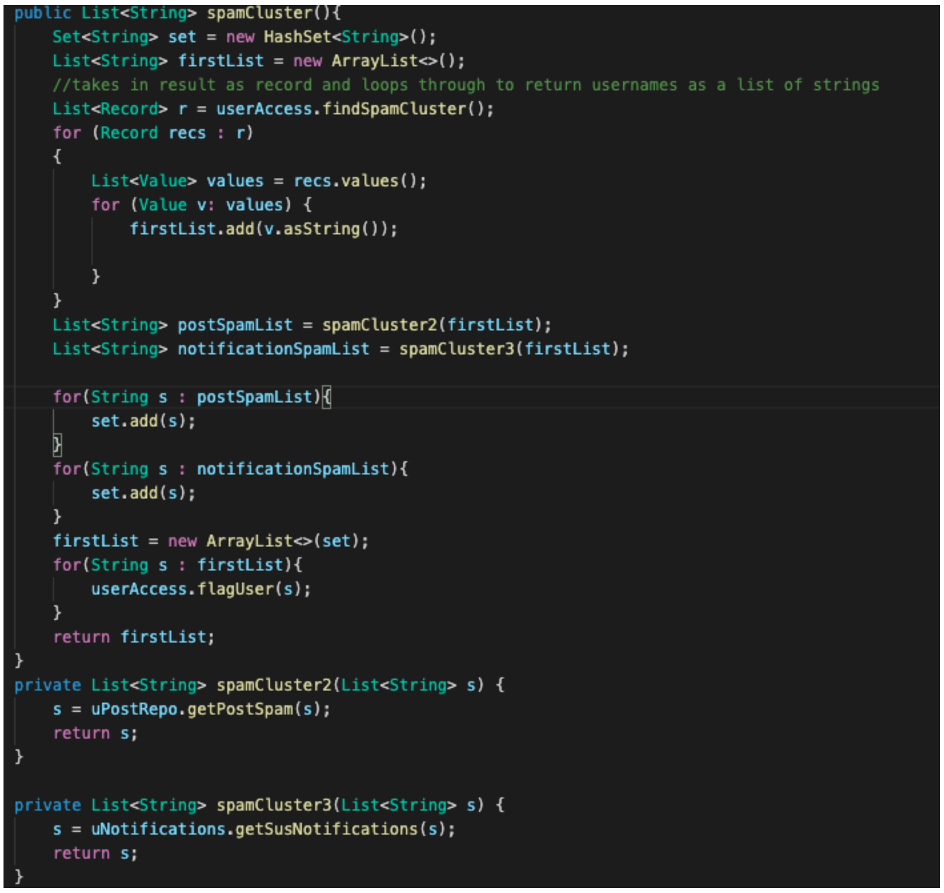
Spam Detection was one of the important parts of this application, as it is to be used in conjunction with the trust to allow users to feel safe on the application. The spam cluster algorithm consists of a number of steps in the backend of the application. It begins with the retrieval of data from the Graph database seen in the method in Figure 4. 15. Once this is successful it will first check if there is already a graph with the name spam in the database, if so it will drop it otherwise it will create a new graph within the database of all the users that have joined that day. Once this is created it will run the Graph Data Science(GDS) algorithm for getting the local clustering coefficient of the users in the graph. This essentially means that it will see how connected the users in the graph are. It is a number between 0 and 1, 1 meaning that the users connections are all connected to each other and 0 meaning that the user connections are not connected to each other at all. The reasoning behind this is to find spam clusters of users that are building connections in order to seem more trustworthy due to following size, which was one of the top indicators for trustworthiness in the survey, as discussed in Chapter 2. The application uses GDS to return a list of users with a local clustering coefficient of 0.75 or over. This number was chosen because spam users may have connected to genuine users already meaning that their connections will not all be in the cluster, the higher the clustering coefficient the more likely a user is part of a cluster but there is a cut-off point where it becomes too low to say that a user is within a cluster and this is why we choose 0.75 as the cut-off. This is the first step and some genuine users may be caught in this part but the next steps will remove some of the genuine users.



```
Result dropRes = session.run("CALL gds.graph.exists('spam') yield exists return exists");
        Record dropRec = dropRes.next();
        //if so it will drop existing one
        if (dropRec.get("exists").asBoolean() == Boolean.TRUE) {
          session.run("CALL gds.graph.drop('spam')");
        }
        //call to create graph
        session.run("CALL gds.graph.create.cypher("
                + "  'spam',"
                + "  'MATCH (n:User) where n.dateTimeJoined = date.realtime() RETURN id(n) AS id',"
                + "  'MATCH (n:User)-[r]-(m:User) RETURN id(n) AS source, id(m) AS target' "
                + ");");
        //Call the get users with local clustering coefficient equal to 1 and return the username of these
        Result = session.run("CALL gds.localClusteringCoefficient.stream('spam')\n"
                + "YIELD nodeId, localClusteringCoefficient\n"
                + "WHERE localClusteringCoefficient >= 0.75\n"
                + "RETURN gds.util.asNode(nodeId).username AS name;");
```

Figure 4. 15 Spam Cluster Query

The first check is for posts made by users on that day, seen in Figure 4. 17. This will check if users within that last posted over a certain number of times that day in this case it is 40 but this can be adjusted depending on results from testing. The reasoning behind this is that spam users may post a lot of content on their profile in order to seem like they are an active user on the application. Once the results from this query are returned they will be added to the set of results. The other query executed by the function seen in Figure 4. 18, will check for notifications, specifically rejected notifications. If a user has already sent a number of requests to people on the application that have been rejected, they will be stored within the database as rejected and this query will check if they have already had over 6 rejections within their first day returning the user if so. A set is used to store the results which automatically handles duplicate results. There is also a query for checking if the user has messaged a lot of different users which is currently not being used because the system only allows you to send messages to connections at the moment but will be implemented in future when and if the application will allow messages to be sent to any user. The method in full is shown in Figure 4. 16.



```java
public List<String> spamCluster(){
    Set<String> set = new HashSet<String>();
    List<String> firstList = new ArrayList<>();
    //takes in result as record and loops through to return usernames as a list of strings
    List<Record> r = userAccess.findSpamCluster();
    for (Record recs : r)
    {
        List<Value> values = recs.values();
        for (Value v: values) {
            firstList.add(v.asString());

        }
    }
    List<String> postSpamList = spamCluster2(firstList);
    List<String> notificationSpamList = spamCluster3(firstList);

    for(String s : postSpamList){
        set.add(s);
    }
    for(String s : notificationSpamList){
        set.add(s);
    }
    firstList = new ArrayList<>(set);
    for(String s : firstList){
        userAccess.flagUser(s);
    }
    return firstList;
}
private List<String> spamCluster2(List<String> s) {
    s = uPostRepo.getPostSpam(s);
    return s;
}

private List<String> spamCluster3(List<String> s) {
    s = uNotifications.getSusNotifications(s);
    return s;
}
```

Figure 4. 16 Spam Cluster All Steps

```java
@Query( value = "Select Username FROM user_posts where Username in :s and CAST(Time AS DATE) =
CAST(CURRENT_DATE AS DATE) GROUP BY Username  having Count(*) >40", nativeQuery =  true)
   List<String> getPostSpam(List<String> s);
```

Figure 4. 17 getPostSpam

```java
@Query(value = "Select user2 from user_notifications where user2 in :s and status=\'rejected\' group by user2
having count(*) > 6", nativeQuery = true)
   List<String> getSusNotifications(List<String> s);
```

Figure 4. 18 getSusNotifications

The Spam cluster algorithm also works in part when users are reported. As seen in Figure 4. 19 when a user is reported a graph is created for all of their connections and it will do the first step for the Spam algorithm and find users with a clustering coefficient of greater than or equal to 0.75. This is useful because it will catch spam users in a cluster that might be not displaying usual spam behaviour and it will flag them and the user that was reported. This may give users more caution as to who they make connections with on the application as they do not want to be flagged as it might devalue their trust on the application.

```java
session.run("CALL gds.graph.create.cypher("
         + "  'spamFriend',"
```

```
          + "  'MATCH (n:User{username:\""+ username + "\"})[r:TRUSTS_EACHOTHER]-(m:User) RETURN id(m) AS 
id',"
          + "  'MATCH (n:User)-[r]-(m:User) RETURN id(n) AS source, id(m) AS target', "
          + " {validateRelationships : false} "
          + ");");
```
Figure 4. 19 Report User Spam Algorithm

4.3.2 Trust

The idea of trust is the most important aspect of the logic behind this application. Trust is often not easy to ensure in social media applications. The huge number of spam and fake accounts on social media makes it difficult for users to have trust in the accounts they are interacting with online. Even though trust is often difficult to establish in the real world as well it is easier than online. This is why the application implements trust that is defined by the user into the application and it does this in a number of ways.

The user can make trust connections with other users and these have two key attributes, the type and the level. These are very important as the level influences not only your trust for that person but also all of your connections' trust for that person. The type is also an important factor as it can show users what kind of relationships people have with each other in order to make decisions on if they should trust them.

The degree of separation is an important aspect of trust within the application as it goes one step further than the standard mutual connections seen on many social networks. It can show you how far away you are from a user through your own connections and users up to 5 degrees of separation away from you. This can be useful because it can show you which of your connections have connections with other users which may influence your decision to send connection requests or accept connection requests. Let's say you have a connection that you don't trust as much as your others and they are the user that connects you to a particular user you are deciding to connect with, this may change your mind on if you want to connect with that user. On the other hand if you have a connection you trust a lot and they are connected to a user you might be more inclined to connect with them. This also works if users just want to see how far away they are from certain users, which can help them decide as well. If we use an example for this it can be more clear as to how this might influence them. Let us say that you know of a person and you know them to be trustworthy but you have no relationship with them, if you see they are connected to a certain user you might trust that the user is genuine, likewise on the opposite side if you know of a user that you absolutely do not trust and see they are connected to someone, you might not want to connect.

This works as seen in Figure 4. 20. The query will find the shortest path between you and another user with up to 5 degrees of separation, this is indicated where you see '..5' in the query. The reason it is capped at 5 is because it becomes more expensive as you reach further out and this would be used a lot on the application, this is due to the exponential nature of the query, for example with 3 degrees of separation you would be checking all of your friends and all of the friends of your friends, which on the scale of a social media application can be very expensive, therefore this will need to be monitored. It will then return the degree of separation as an integer and will also calculate a personalised trust level for that users by multiplying the trust at each degree of separation by 0.75, meaning as you get further away the trust will reduce. It also works in a way that you might have users along the path with a low trust for a user which will decrease the personalised trust returned. This is then stored in a record and formatted and returned to the user when looking at different user profiles.

```
Result = session.run("match path=shortestPath((u1:User{username:$username1})-[rel*..5]-
(u2:User{username:$username2})) "
                    + "return length(path) as Degrees_of_Seperation, REDUCE(s=100.0,r IN range(0,size(rel)-2) | s *
0.75) as Your_Trust;"
              , parameters("username1", username1, "username2", username2));
```

Figure 4. 20 DOS query

4.3.3 Issues

In the logic layer there were some issues that will be discussed in this section. Some of them include formatting of the queries correctly using java, deciding on what sort of algorithm to use for the spam cluster algorithm and finding ways to access the data returned in more complicated queries.

Formatting the queries initially gave me some issues because of the way they have to be formatted to work correctly including where to place in variables needed for queries etc. This caused issues because the log of the application didn't show me a lot of information on why the queries weren't working. The way that this was solved was by testing the queries in Neo4j and in PostgreSQL for the SQL queries and adjusting them using the feedback from the errors there.

The algorithm for spam detection caused some issues as well this was due to the ideas not working as well as originally thought at the start. Initially the system set the query to only return users with a clustering coefficient of 1 this would prove to not work when spam users made other connections outside the cluster so to solve this the coefficient needed to return users was reduced it greater than or equal to 0.75 to allow for some spam users to be caught even with some outside connections. This brought about a new issue where it would find users that weren't in the cluster and return them and to solve this there was additional checks discussed previously with the message, posts and notification parts of the algorithm. For the most part this worked but the messages query was not useful at this stage of the project as spam users would only be able to message their connections. This was solved by removing the messages query from the algorithm but was kept in the DAO for future use.

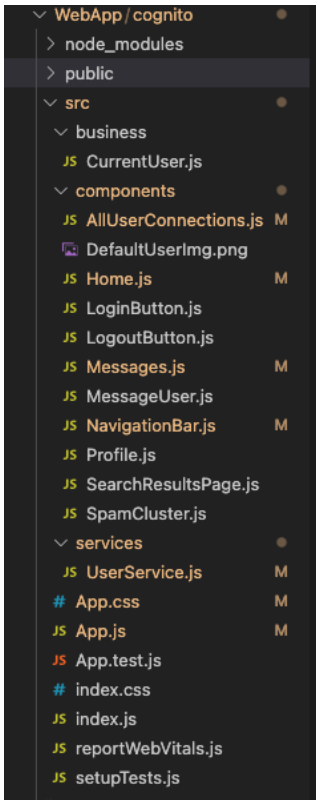
The last issues was figuring out how to access the data within more complex queries, this is because the queries returned records with more than one column and this was solved by research in how to loop through each record and get the keys and values for each of the columns.

4.4. Presentation Tier Development

This section will detail the development process of the front-end of the application. As discussed in Chapter 2 and Chapter 3, the technologies used to develop this tier were React Js, Node Js, React Bootstrap and JavaScript. This section will go over the code that was written in order to achieve the final result for the application.

4.4.1 React

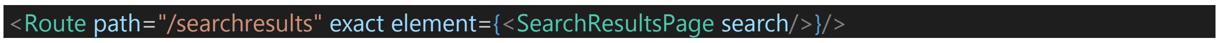
I researched various different important react documentation including the use of hooks which allows using states and effects without creating a class component which proved useful considering I initially started with using class components and realised the limitations it had in relation to what I wanted to achieve. My final file structure was as seen in Figure 4. 21 with a components folder which holds all my custom components, Business folder which holds the Current User Class and services folder which handles User requests to the API. App.js is the page all these Components are accessed through, through the use of a router.



```
∨ WebApp / cognito
  > node_modules
  > public
  ∨ src
    ∨ business
      JS CurrentUser.js
    ∨ components
      JS AllUserConnections.js   M
      DefaultUserImg.png
      JS Home.js                 M
      JS LoginButton.js
      JS LogoutButton.js
      JS Messages.js             M
      JS MessageUser.js
      JS NavigationBar.js        M
      JS Profile.js
      JS SearchResultsPage.js
      JS SpamCluster.js
    ∨ services
      JS UserService.js          M
    # App.css                    M
    JS App.js                    M
    JS App.test.js
    # index.css
    JS index.js
    JS reportWebVitals.js
    JS setupTests.js
```

Figure 4. 21 File Structure

Components are essential when developing a react application, they are reusable bits of code that serve the same purpose as JavaScript methods, but will work in isolation and return html. There are two types of components in react and these are Functional and class components. The class component will extend React Component and will need a 'render()' method to return the HTML, whereas for functional components you can have the same functionality with less code. In this application functional components were used because they allow the use of hooks like states and effects unlike class components This was essential for this application as these hooks provided important functionality for what the application needed to achieve, which is discussed more in this section below. The components in the application can range from whole pages to logout and login button components as seen in file directory in Figure 4. 21. Components can also take in props which are attributes that are defined and sent to the component when rendering. This proved to be particularly important in this application when using the search results page seen in Figure 4. 22. Where we pass in a search value to the 'SearchResultsPage' component in order to search a particular query.



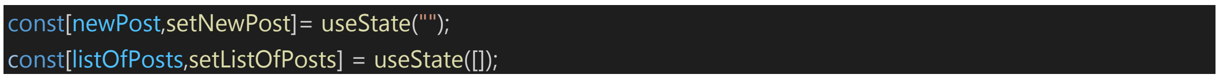
```
<Route path="/searchresults" exact element={<SearchResultsPage search/>}/>
```

Figure 4. 22 Props Example

Moving on from this it can be seen in Figure 4. 22, the use of ‘Route’, which is from the Routing process in react that allows the user to be directed to different pages within the application. These routes are defined like above and given a directory and these directories in the website are the only ones users can access. The Route tag will take a component that will be rendered when a user is on that directory in the website. In this particular application these were useful in navigating through the pages especially when not allowing the normal user access the admin components like the spam cluster component. The root component will check if the user is an admin or not and if they are they have a route defined for admin components and if not they will not be able to access them.

Hooks are an important part of the react framework that allow you to use stateful logic and other react features in a component without class and allows for the reuse of these features. They essentially allow you to use React state and lifecycle features (discussed more in the paragraph below), while using a functional component. Hooks are used at the top level of components before any returns and include Effects and States which were used a lot in the development of this project. There are 3 main rules with hooks, they can only be used in a functional component, they can only be called at the top level of a component and they cannot be conditional.

States in react are used to track the state of objects within a component. It is an essential feature in react especially within this application because of one important functionality and that is that whenever a state changes within a component, the component will re-render, this allows for the application to update as objects change without having to refresh the page for updates. Within the application this was used in the development of multiple components, in particular the message and post components. A state is declared within a component as seen in Figure 4. 23. The state declaration creates an attribute and a method. The attribute holds the data within a sate and the method will update it. In the application anytime a message is sent or a post is made, the state of ‘newPost’ will be updated which allows the app to re-render and display the new message or post after it is sent.



```
const[newPost,setNewPost]= useState("");
const[listOfPosts,setListOfPosts] = useState([]);
```

Figure 4. 23 State Declaration

The other feature used in the development of the front-end of the application was Effects. Effects are hooks that allow the user to run functionality that might be needed at the first render of a page only once or if any functionality needs to be executed if a certain value changes. In this application effects were used with states and also by themselves in order to execute important functionality. This is used with the example discussed in Figure 4. 23. We see in Figure 4. 24 that when the component is rendered in home it will call the getCurrentUser() function. This will be triggered every time the user makes a post as state is being updated, causing a re-render. The function in this effect hook will update the list of posts on the home page so the user will be able to see the post they made.

```
useEffect(() =>{
      getCurrentUser()
  },[]);
```

Figure 4. 24 Effect Example

Finally within the web application there was a service class created. This was used to call the API if there when any data needed to be posted or returned from the database in the backend. These functions within the service class all took similar form as seen in Figure 4. 25. They used Axios to call the API using different directories in order to access different functions within the front controllers of the API.

```
isFriend(user1,user2){
      return axios.get(USERS_SERVICE_API_URL + 'isFriend/' + user1 + '/' + user2);
  }
```

Figure 4. 25 Service Function Example

4.4.2 Auth0

Auth0 is what was used for the authentication and access management part of this project, i.e. the login and registration. This is a solution that is added to your project with setup specifications within React and within the Auth0 website in order to connect to the online Auth0 server. This was used to create login and registration functionality without having to encrypt and decrypt and store passwords myself.

Auth0 allowed for three key important functionalities within the front-end, logging in, registering and pre-user registration methods. The login functionality is handled mainly through Auth0 authentication and can be customised to allow for login with google *etc.* The registration is also done through Auth0 but with this application there was a need for extra functionality upon registration. Pre-user registration allowed for this functionality. When the user registers there are flows within auth0 that allow you to write functions before registration is complete, and part of this was to execute functionality to set user metadata. This was for the use of checking if users are admin or not as seen in Figure 4. 26. Also seen is an API call that will add new users to the database upon registration with their default details set which they can change when logged in.

```
exports.onExecutePreUserRegistration = async (event, api) => {
	await axios.post("http://localhost:8080/api/newuser/" + event.user.email)
	api.user.setAppMetadata("app_role", "normal");
};
```
Figure 4. 26 Pre-User Registration

4.4.3 Issues

There were a number of issues encountered and overcame throughout the development of the presentation layer of this project and this largely came down the inexperience with the React system as a whole before this project. This section will briefly discuss some of these issues.

Neovis Js was used in order to help with graph visualization to the user and while implementing there was issues regarding security of the database. Since this visualization is used in the front end, the information required to connect to the database server had to be declared there, this means that users are able to access the credentials of the database if they were to use developers' tools to see source code. This is a potential security problem and is largely due to the limitations of Neovis.

In the pre-production and testing phase this wouldn't be an issue per say but when the application would be put into production this issue would need to be resolved in order to secure the database and make sure users do not have access to credentials that would allow them to connect to it. This is work that I have set for the future and my reasoning behind this is the prioritization of the essential and parts of the project that shows the potential of what this application could do. The spam detection algorithm and social media functionality was prioritized.

There are some ways in which I could go about doing this fix in the future such as storing the credentials securely within Auth0 and calling it and this will be investigated further in future along with possible other solutions to this security issue.

States and Effects was another issue that caused some time in the development of this project, again this was largely down to the inexperience with react. There were a number of cases where Effects were being used incorrectly causing infinite loop calls to the API which crashed the system and this was overcome by research on how to properly implement Effects and also how to use states with them correctly. States gave some issues throughout the development due to their asynchronistic nature when setting the state. This caused issues when re-rendering while the states were not set yet which was solved through further research on how to properly implement them.

Asynchronous API calls were the last issue faced and still caused some issues to the final system. This is due to idea of fetching data from the backend and attempting to map and use the data before it has been returned to the front-end. In a lot of cases this was solved by waiting for the data to return but there is still some instances where the problem remains and this will be investigated in the future work of this application

4.5. Conclusions

In this chapter we looked at the development of the system, firstly by discussing the front-end development, then discussing an overview of the middle API layer , moving on from that the backend that was implemented and then finally the database layer.

The API Layer was at first tricky to navigate due to the specific ways in which classes and methods needed to be formatted but this was quickly learned and the API layer was successful in implementing the necessary requirements for the application. There could be much more work to do on this as the application grows and new features are added but this will be discussed in future work.

The logic layer successfully did what was required of it but will need some adjustments as the system continues to grow. The vital next part to the logic is testing it on real online databases of spam clusters and how they behave to be able to adjust accordingly. The logic layer was a vital part in implementing key aspects of the application and there is much scope for improvements in future.

Overall the outcome of the front-end was a huge success in the project. It has the functionality and look for a social media application, there is a lot more improvement to be done but given the scope of the project and the timeframe, it achieved most of what was set out from the designs.

For the development part of this project, there was a lot of work to be done and still is a lot of work to do in future to bring this to web scale, given the task at hand, the system mostly works efficiently and smoothly however with growth of the application some changes to accommodate would be necessary.

There were a lot of learning outcomes from this part of the project, including new technologies and methodologies that allowed for a lot of experimentation and learning along the way. The development was very successful and proved to be a welcome challenge.

5. Testing and Evaluation

In this chapter I will discuss the testing and evaluation of the application. The system was tested throughout the development process through a test plan, which we will be discussed in more detail below. As well as this there will be discussion on usability testing done using 6 end-users of different ages and technical backgrounds. Finally we will evaluate these results and the system as a whole.

5.1. System Testing

For the System testing a test plan was created, as seen in Figure 5. 1. This plan consisted of a list of tests for the system that should be working at the final stage in the project. The description contains a pre-condition for the particular test and the test itself, the expected outcome is a statement of what should occur and the pass column contains pass, fail or partial. The partial and fail results will be discussed. The test plan was also split into admin and normal user to test different functionalities between them. The admin section will only contain admin functionality tests, for the sake of avoiding repetition.

Test No.	Description	Expected Outcome	Pass?
	Normal User		
1.	Pre-condition: User has browser open and has the address. Test: Does the application load successfully when address is entered?	The application will load on the start-up screen.	Pass
2.	Pre-condition: User is on the opening page and has login information. Test: Can users successfully login from the opening page?	Users can easily login from the opening page.	Pass
3.	Pre-Condition: User is on the opening page. Test: Will user be able to register?	User will be able to register a new account.	Pass
4.	Pre-condition: User is on the opening page and doesn't have login information Test: Can users proceed given the wrong login info?	Users should not be able to proceed given the wrong login info	Pass
5.	Pre-condition: User is on home.	User will be able to make a post on home and post should	Pass

	Test: User can make post from home.	be seen immediately after sent	
6.	Pre-condition: User has connections Test: Will user will be able to see friends posts?	User will be able to see friends posts	Partial
7.	Pre-condition: User has a connection. Test: Will user be able to remove trust.	User will be able to remove trust	Pass
8.	Pre-condition: User has connections. Test: Will user be able to send message to connection?	User will be able to send message to connection and message should be seen after sending.	Partial
9.	Pre-condition: User has connections Test: Will user be able to see connections messages?	User will be able to see messages from connections in real time.	Partial
10.	Pre-condition: User knows someone on the application to search for. Test: Will users be able to search successfully for other users on the application?	Users should be able to successfully search for other users on the application.	Pass
11.	Pre-condition: User is given two users to test both outcomes, I.E a user with a path and with no path. Test: Can users see their separation from other users(if any), the path to that user and the trust level they have for that user?	Users should be able to see the path, degree of separation and trust level for another user.	Pass
12.	Pre-condition: Users have connections. Test: Can users see their trust connections visualised?	Users will be able to see their trusted connections visualised	Pass
13.	Pre-condition: User is has a spam user to report. Test: Can users successfully report spam or fake accounts?	Users will be able to report other users.	Pass
14.	Pre-condition: User has been reported Test: Can user see if another user if flagged?	User will see warning if user is flagged.	Pass
15.	Pre-condition: User has a notification	User can see notifications when logged in.	Pass

	Test: Can user see notifications when they login?		
16.	Pre-condition: User has a trust request. Test: will user be able to accept request?	User will be able to accept request and define relationship and trust level.	Pass
17.	Pre-condition: user is on a profile they do not have a connection with. Test: Will user be able to send trust request?	User will be able to send trust Request.	Pass
18.	Pre-condition: user is registered and has a node in the database. Can users edit their details?	Users will be able to edit details.	Pass
19.	Pre-condition: User has connections and/or is looking at a user who they have a degree of separation with. Test: Will user be able to interact with the map?	User will be able to interact with the map.	Pass
20.	Pre-condition: user is logged in. Test: Will user be able to logout?	User can log out.	
Admin			
21.	Pre-condition: user has been flagged Test: Will admin be able to unflag user.	Admin will be able to unflag user	Pass
22.	Pre-condition: user is logged in as admin Test: Will admin be able to see extra menu option for spam cluster?	Admin will see extra menu option for spam cluster.	Pass
23.	Pre-condition: Users have been reported. Test: Will actions be taken on users who have numerous reports?	Users who have a number of successful reports will have actions taken against their account	Fail
24.	Pre-condition: Spam users with spam behaviour are in the database. Test: Will Spam Clusters be successfully found and flagged with limited to no outliers?	Spam clusters will be found with limited to no outliers.	Pass

Figure 5. 1 Test Plan

Most of the tests in the test plan were successful with the exception of partial passes and one fail which we will discuss further. The first partial is Test 6, ‘Will users be able to see their friends posts?’. The reason this was a partial pass is because on the home screen you will be able to see your friends posts, however if a friend makes a post after the home page has loaded you will not be able to see the post until you refresh.

The other two partials, Test 8 and 9, are similar to the test 6 though applicable to sending and viewing messages. Users will be able to send messages, as well as receive and view them but this is not in real time as of yet, so users will have to reload the page or click on the user they are messaging again to get new messages received.

The fail was on 'will actions be taken on users who have numerous reports'. Although there is a warning displayed after a user is reported there is no action as of yet for these reported users. This will be discussed in more detail in the further work Chapter 6 Section 6.3 of this report as it is something that will need to be added to the application in order for the reporting and flagging system to operate to its best ability

5.2 Usability testing

The next tests that was implemented on the system was usability testing. Usability is a very important testing phase that needs to be done especially given the nature of the application. Social media networks are used by all types of people in today's world, from technically challenged individuals to the more technically proficient. This test was carried out where 6 end-users were given a list of tasks and they reported whether or not they were able to do those tasks and gave feedback on if they weren't, as well as general feedback that was used for the evaluation section of this chapter. This test was carried out on 6 different users 3 women and 3 men, of ages ranging from 21-65, all with different levels of technological experience. They were all given a username and login, in which the user they were given had a notification, connections with posts and connections with whom they had already messaged previously. The list of tasks are seen below in Figure 5. 2 follows:

Task No.	Task
1.	Try Login with wrong credentials
2.	Login in with given credentials
3.	Search for users
4.	Go to a profile from search
5.	Interact with the map on the profile and try understand what all the information means

6.	Go to home and view posts
7.	Make Post
8.	Go to messages and view messages
9.	Send message
10.	View your own profile
11.	Search for the user 'acole' and see if you can see warning message on their profile
12.	Report an account
13.	See notifications
14.	Accept notification and define trust level and relationship.
15.	Send Trust Request
16.	Logout

Figure 5. 2 List of Tasks

The overwhelming feedback from this test was that the users were successful in most, but not all of these tasks. There were 2 users who attempted to send multiple messages because they weren't sure that the message had sent, this is an issue with the asynchronous method to call back and retrieve updated messages. The issue being that the messages state is not updating before the page is re-rendered. This is an issue that will be fixed in future work.

There were 2 users that didn't understand what degree of separation meant, which is understandable as not everyone will know what this means, from this feedback it is apparent that is it important for explanations and examples to be included in the documentation for the users.

There was also some difficulty with accepting a trust request as it did not go through for all of the users due to users trying to add a space in the relationship type which gave an error in the API call, there are many ways that this could be fixed, one of them being that spaces could be removed from the string or another option could be to replace the spaces with an underscore or dash.

The final difficulty was that some users hit enter for their search and this did not return any results because the system will only search for a person when the search but is pressed, not the enter key. This particular issue can be resolved with a key listener for when the user enters that key. The functionality when the user presses that key would be the same as the search button.

All of the other tasks were completed successfully with no extra assistance needed. It was interesting to see the feedback from this because when developing a system like a social media website there are a lot of standard familiar ways in which certain functions will work. This can help with users knowing automatically what to do when using it, but it must be said that not all users will have this familiarity and automatic response when seeing these common features. This is one of the main learning outcomes taken from this test.

5.3. System Evaluation

Evaluation is very important along with testing, especially for a project like this where the whole idea is building a safe environment for users to feel like they not only trust the users but the system as well. It's very important that users can use the system and benefit from the features without having it be too complicated for them even though underneath the front-end the code might be quite complicated. I evaluated my code based on usability and I split this in to understandability, documentation, buildability and learnability (23). The users who undertook the usability test were also asked to give feedback on these four categories.

5.3.1 Understandability

This criteria is based on how easy the system is to understand. This includes the users' understanding of the purpose of the application, they understand what the software does and there are descriptions about how it works to assist them in both basic and advanced functions. (23) Most of the users, given their feedback, understood what the software does and how to use it.

They were given a brief description of the application but nothing of much detail and there were some users who didn't quite understand the idea behind the degree of separation because they didn't exactly know what it was. The users were then given an oral explanation of this and they understood and also understood the use that it might have in a social media application. A lot of users expressed what they thought the application's purpose was and a few said that the idea of the relationship type and trust level we're definitely interesting because in the real world people have different relationships with different people and on a lot of social media applications there is only the option to be a friend or not a friend. This was important feedback as this was the reasoning behind those additions to the application.

5.3.2. Documentation

The Documentation criteria refers to the quality of the documentation provided to users. The users documentation is considered sufficient if the user is able to navigate the system and get any questions they might have in the documentation. If there are a lot of questions the users can't solve, then the documentation needs to improve.

There is documentation for the users in the help component, this allows the user to see the basic functions of the system and although this documentation has potential for improvement, it certainly would have helped some users who were involved in the usability testing and this was an outcome that was taken from that test. Other documentation that was completed as part of this project was the README file which was not given to users but rather to developers for assistance on how to run the application.

5.3.3 Buildability

This refers to the ability to build the software on the user's devices, for example with my project I am hoping to build a react application which should allow the application to be built on most devices.

In the case of this project the application was not deployed, meaning that it was not tested on multiple different devices as it is currently running locally. However upon deployment the application will be easy to build as it will mostly be deployed on the server-side rather than on users devices.

5.3.4. Installability

Finally the criteria of installability is in reference to how easy software is to install on various devices. Which can include if its available on all devices, all device versions etc.

Since this is a web application the users are able to access it on any device or version through the browser on that device this was one of the main reason React Js was used as it allows for ease of access across a lot of different devices.

5.4. Conclusions

In this chapter we discussed the testing and evaluation of this project. Firstly the system testing via the test plan table was discussed, then the usability testing was discussed where users were given a list of tasks to complete and asked to give feedback. Finally the evaluation of the system as a whole was discussed.

The testing phase of this project proved to be successful in part due to the success of the tests themselves but also some of the bugs found in testing which demonstrated the need allowed for some necessary changes to system that were needed to complete this project. The user feedback from testing was also vital in the future development of this application. There were a lot of ideas given and concerns given back which will prove to be extremely valuable when continuing to work on this application in future.

6. Conclusions and Future Work

In this chapter the key lessons learned from this project will be discussed. This will begin with discussion on the initial project plan and how the plan changed throughout the initial proposal. Then moving onto the conclusions and insights gained through the research, development, design and testing of this application and finally the future work of this project.

6.1. Project Plan

Throughout the development of the project there were some changes to the initial plan but the overall objective of the project stayed the same throughout. The objective of this project was to explore a new possibility in creating a social media application that uses trust to establish a sense of security. This trust would help users in their decisions with connecting to other users. In doing this it would allow users to see personal trust levels for different users and this, combined with a spam cluster algorithm, users would allow users to feel more secure using the application. The objective came from the idea of how connections are made in the real world as there seemed to be a disconnect between how social media applications work to create relationships between users and how this happens in the real world.

Although the initial concept stayed the same throughout, there was some adaptations as to how it would be implemented. Initially the thought was that the personalised trust could be calculated just by degrees of separation but later this was changed to combine this idea with the idea of varying trust levels between user nodes to give a more accurate representation of real personal trust between users. Another adaptation was the relationship type which was adapted to allow users to define their relationship(if they wanted) with a user, which would again add an extra layer to the trust process to try and portray real world scenarios.

The spam algorithm initially started as finding clusters of interconnected users, but was later adapted to be able to distinguish more between normal users and spam users by trying to catch spam behaviour in these clusters to eliminate genuine users who were being caught in the algorithm. This algorithm is and will always be a work in progress as spam users will have behaviour changes and new methods of spam detection are investigated.

The initial project consisted of three stages as seen in the Gantt chart in Figure 6. 1, all with different sections and time frames, for the most part these timelines were an accurate representation of the project timeline, with only the development stage being a bit longer than expected due to the amount of development needed for this project.

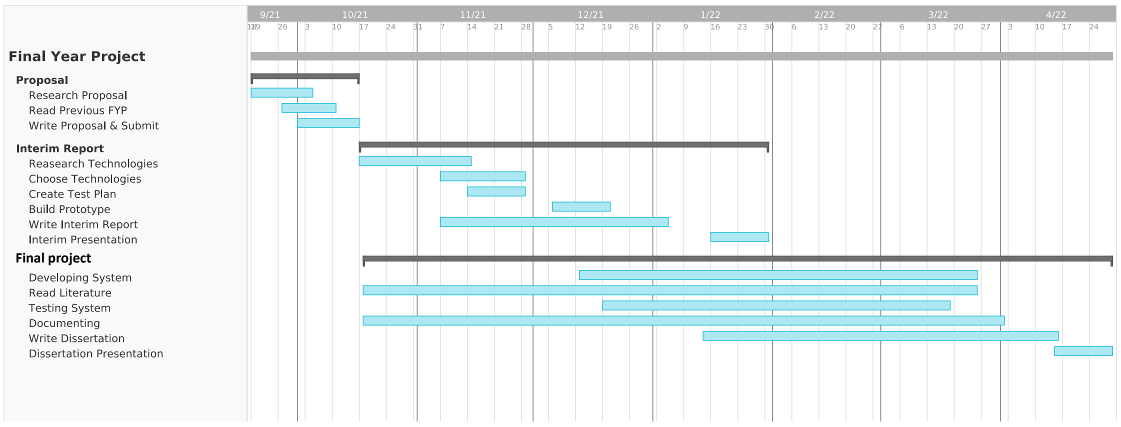


Figure 6. 1 Initial Project Plan

As a result of the development part of this application taking longer than expected, the application was not deployed as planned in the initial proposal. This was due in large part to the fact that it was more important to continue to work on the application and to implement the functionality properly before deployment. With that in mind, the important deliverables were achieved in the development of the application.

6.2. Conclusions

The idea of the project as a whole is an extremely interesting and exciting one. It is a complex problem that is becoming more and more important to solve as the world evolves becomes more and more reliant on social media and its powers. You can see this happening everywhere from the way we communicate with each other on a daily basis, to connecting with colleagues, promoting businesses etc and given the extreme circumstances due to Covid-19 even more so. That is why this area is so fascinating and ever changing.

The project itself took this problem and started to look at ways to solve it using real world concepts and applying them. The success of an application like this however relies a lot on how it would work once released for users to test and use, but the overall concept and ideas were met in the finished product of the application. The idea of trust online is such a tricky concept because a lot of people don’t have trust in the online world and are justified in thinking this way. That is why this project brings in the idea of users defining their trust because it gives some power back to the users.

As for my own personal conclusions, I think this implementation has the potential for bringing a new concept to social media that has not particularly been seen and I am excited to continue to research and test this concept and adapt it to reach its full potential. I think that this is only the beginning of the development around this idea and there is much to be considered, but as a start to creating such an application this project reflects a great starting point to this potential.

6.3. Future Work

As a social media application is a complicated system with many different functionalities there are a lot of different changes and additions which could be made to the application. This section will discuss what kind of possible changes in future this application will implement.

Firstly the spam cluster algorithm will need further development and testing. There is a lot of potential with the spam algorithm to become more complex and factor in a lot of different ways of catching spam accounts on the application. Content based behaviours is a concept which has a lot of potential in the future development to be included, such as posts similarity, which is an idea of the way in which spam users will post and the content of these posts. This is definitely an interesting concept because spam users tend to act in similar ways when sharing content and finding ways of defining these similarities and identifying users with these similarities and this can be implemented using methods such as the Jaccard coefficient (24), such that every post pair of a user, the number of shared words is divided by the total number of words in the post pair. Mention Ratios is another content-based feature that could be implemented in future, provided the application implements a type of mentioning functionality. This is essentially where the number of posts containing mentions of users who are not connected to that node is divided by the number of posts that user has (24). This is a behaviour that has been seen in spam users where they will tag people in their posts, so it could be very useful in detecting spam accounts. These two concepts are just to name a few of the many different ones that are used and tested that have potential in the future development of this application.

There are also a lot of social media basic features that need to implemented within this application to bring it to the standard expected from social media applications today including blocking users; exchanging more advanced content such as voice messages, photos or gifs; group chats for users etc. These have become a standard in the area of social media and would be necessary to implement in order to be successful.

The part of the algorithm, previously discussed about user message spam detection is currently not being implemented within the application because it does not work as accurately if users are only allowed to message their trusted connections. This is something that needs to be thought about in future and this comes down to if users would like to be able to message people they are not connected to. It is uncertain what the response for this question will be, because on the one hand, users might not want any account to be able to message them, but on the other hand this feature could enhance the algorithm to catch more spam accounts making it safer overall for users.

At the moment the application shows the level of trust between each user and although this can be useful for users making decisions on who to trust, it could also be problematic as people might not want to share the level of trust they have for other people. This means that there is also the potential for hiding the level while still calculating personal trust levels in the backend.

Based on some of the feedback given by the users who tested the system there was an idea brought to my attention that could use trust levels to hide content from certain users. We see some elements of this in social media applications such as close friends stories on Instagram (25), where users pick a number of users that can see certain stories they post on their account. This would work in that users would be able to set different posts to be only accessed by people at certain trust levels and this idea would be a great addition that I think users would love, because in the real world users will only tell things to certain people and this could be reflected in the application.

Bibliography

1. AIT News Desk. aithority.com. [Online].; 2021 [cited 2021 November 13th. Available from: https://aithority.com/technology/analytics/hypeauditor-reveals-key-findings-of-frauds-impact-on-the-us-influencer-market/.

2. Basicthinking.com. [Online]. [cited 2021 12 12. Available from: https://www.basicthinking.com/bots-instagram-accounts-fake/.

3. Solon O. Nbcnews.com. [Online].; 2021 [cited 2021 October 18th. Available from: https://www.nbcnews.com/tech/social-media/british-accounts-sent-bulk-racist-abuse-during-euro-2020-final-n1276432.

4. Çıtlak O,DM&DİA. Springer Link. [Online].; 2019 [cited 2021 October 18th. Available from: https://link.springer.com/article/10.1007/s13278-019-0582-x.

5. Sheikhi S. An Efficient Method for Detection of Fake Accounts on the Instagram Platform. Academic. Gorgan: Azad University, Gorgan, Iran, Department of Computer; 2020.

6. LinkedIn. [Online]. [cited 2021 12 22. Available from: https://www.linkedin.com/.

7. Almabdy S. IEEEXplore. [Online].; 2018 [cited 2021 October 18th. Available from: https://ieeexplore.ieee.org/abstract/document/8441982/authors#authors.

8. Ontotext. Ontotext. [Online]. [cited 2021 12 22. Available from: https://graphdb.ontotext.com/documentation/free/free/graphdb-free.html.

9. Orientdb. orientdb. [Online]. [cited 2021 12 18. Available from: https://orientdb.org/.

10. Neo4j. Neo4j. [Online]. [cited 2021 12 19. Available from: https://neo4j.com/developer/graph-platform/.

11. Cybertec. cybertec.com. [Online]. [cited 2022 March 20. Available from: https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/#:~:text=PostgreSQL%20is%20the%20most%20professional,more%20than%20two%20decades%20now.

12. talend. talend.com. [Online]. [cited 2022 March 22. Available from: https://www.talend.com/resources/what-is-mysql/.

13. Conceptatech. Conceptatech. [Online].; 2018 [cited 2021 12 22. Available from: https://www.conceptatech.com/blog/php-server-side-scripting-designed-web-development-review.

14. Kean K. Makeuseof. [Online].; 2021 [cited 2021 12 19. Available from: https://www.makeuseof.com/node-js-server-side-javascript/.

15. Yank K. Sitepoint. [Online].; 2001 [cited 2021 December 10. Available from: https://www.sitepoint.com/server-side-language-right/.

16. Macquarrie A. The Manifest. [Online].; 2018 [cited 2021 December 4. Available from: https://themanifest.com/app-development/blog/hybrid-apps-vs-native-apps.

17. Synopsys Editorial Team. Synopsys. [Online].; 2017 [cited 2021 November 17. Available from: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/.

18. Justin Jones SW. Medium.com. [Online].; 2019 [cited 2022 April 06. Available from: https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec.

19. Martin E. Google. [Online].; 2021 [cited 2021 11 4. Available from: https://docs.google.com/forms/d/e/1FAIpQLSd3B8XHl-IEbtA4gaXEd2C5M8_I7lXBfjphLKavEpd7HaJz7A/viewform?usp=sf_link.

20. Zat V. GitHub. [Online].; 2018 [cited 2021 12 22. Available from: https://github.com/vzat/mongo-stratus-connection-manager.

21. Hanratty E. Detecting Bot Twitter Accounts using machine learning. Dissertation. Dublin: Technology University Dublin; 2017.

22. Spring. spring.io. [Online]. [cited 2022 March 30. Available from: https://spring.io/projects/spring-boot.

23. Mike Jackson SCaRB. Software Evaluation: Criteria-based Assessment. Edinburgh: Software Sustainability Institute; 2011. Report No.: 1.

24. Girdzijauskas ASaS. Adaptive Graph-based algorithms for Spam Detection in Social Networks. Stockholm: Royal Institute of Technology (KTH), Sweden; 2016.

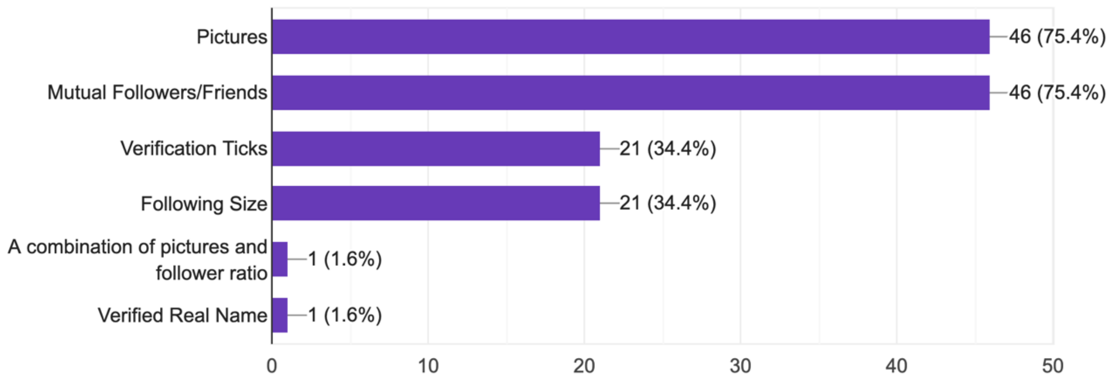
25. Instagram. Instagram.com. [Online]. [cited 2022 April 03. Available from: https://help.instagram.com/2183694401643300.

26. React. React-Bootstrap. [Online]. [cited 2022 March 28. Available from: https://react-bootstrap.github.io/#:~:text=React%2DBootstrap%20replaces%20the%20Bootstrap,choice%20as%20your%20UI%20foundation.

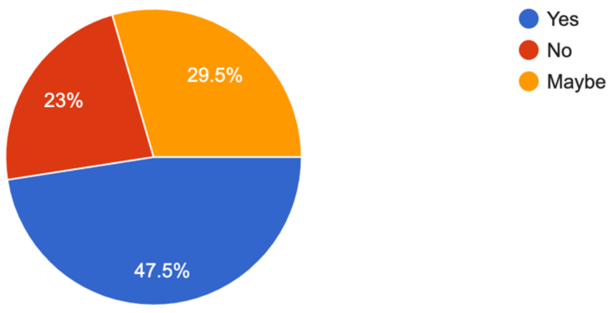
Appendix

Appendix A

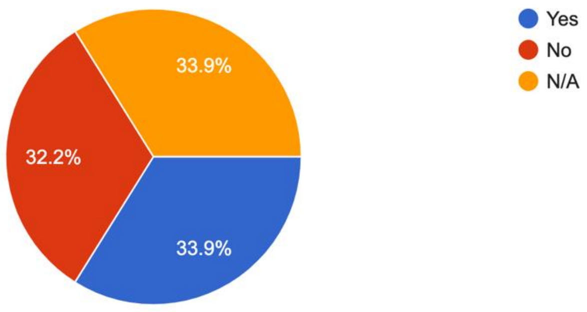
Tick below what you look for to verify a user is real on social media:
61 responses



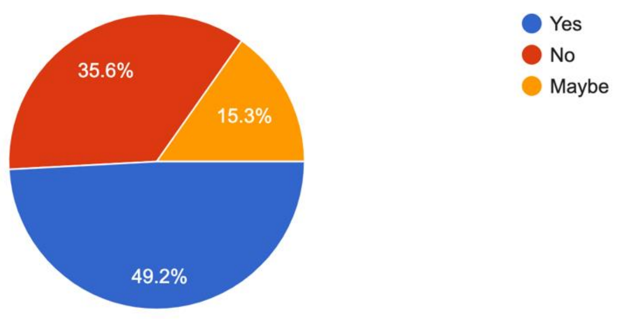
Given an algorithm that removes spam/fake users with a 99% accuracy, would you be willing to let 1% of legitimate users be removed as well?
61 responses



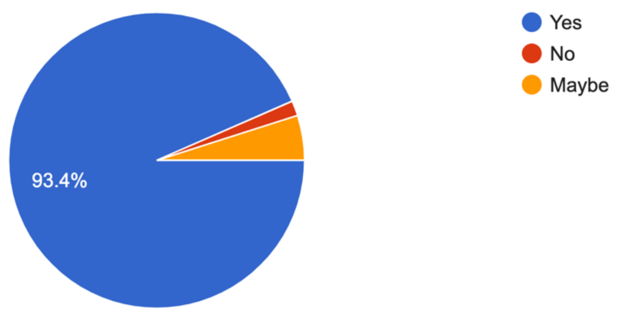
If yes, would you be willing to let more than 1% be removed?
59 responses



If this algorithm found you in the 1% would you still have the same response?
59 responses



Following the rise in discrimination online, such as the racist abuse after the EURO 2020 Final, do you think it's important that a social media needs to hold users accountable for what they say?
61 responses

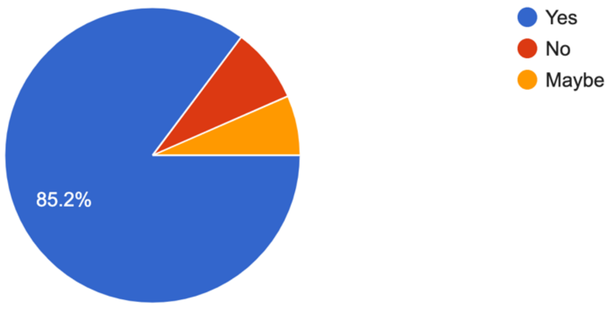


If so, what would be an appropriate action to be taken on these accounts (If you answered no type N/A)
54 responses

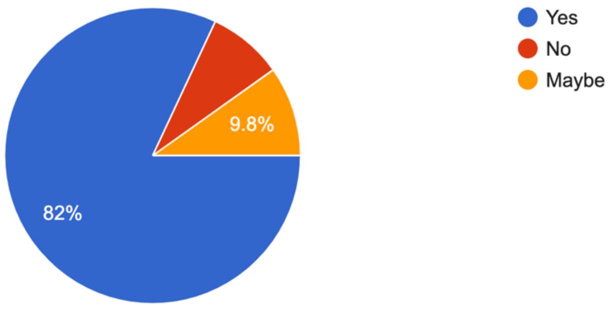
- Banned
- Permanent ban
- Temporary ban
- Banned from platforms for hate speech etc but actually properly banned not just soft bans or people getting away with it
- Banning them
- Deleted account or no ability to post (can only View other people's content)
- Removal from platform or details given to local authority for reprimanding
- A ban system
- Bans
- Banned account
- Account suspended
- Removal of their account
- Inability to post for x amount of time

- Account removed
- Ban them
- Freedom of speech (obviously do not agree with any sort of hate speech but a freedom of speech a legitimate argument)
- Banning the user for a set period of time or deleting an account
- Take as much action as would be taken in real life
- Ban their accounts
- Banning accounts made attached to a phone number/IP address
- Banned for a period of time
- Delete their accounts and not allow them to make a new account in the future (ban IP address)
- They should be prosecuted under laws
- Accounts deleted
- Delete them
- Having their account temporarily banned until they learn their lesson
- 3 strike system. Give clear warnings show user their strike history
- Blocked from posting or commenting
- Banned from said website
- Ban for a period of time, if repeated banned completely
- Ban account
- Deactivate accounts
- Warning then account removed
- Warnings, then deletion of account. I'd think there should be an appeal option in case it was an in joke between people of the group
- Banned from platforms.
- Bans
- Suspension/termination of account
- They should be banned and reported to the police
- They're accounts should be taken from them. Or they should be locked out of them forever and not be able to make a new one under their name or different name but same details.. i.e.. their email or phone number
- Account blocked or users name and info reported to respective authorities
- ban from social media
- Block their account and email
- Long term bans using a devices IMEI
- N/A
- Banning them, having the account removed etc
- Account banning
- Banned
- Fines or imprisonment
- Law enforcement
- Soft ban. Permanent ban for repeat offenders

Would you be willing to scan your face with your camera in order to make sure you are not a bot account? (No data would be kept from this, only for verification you are human)
61 responses



Would you be interested in seeing the degrees to which you are separated from a user? I.E. If you go onto a users profile you might see "2nd" where...trust on the network and so on? (See image below)
61 responses


● Yes
● No
● Maybe