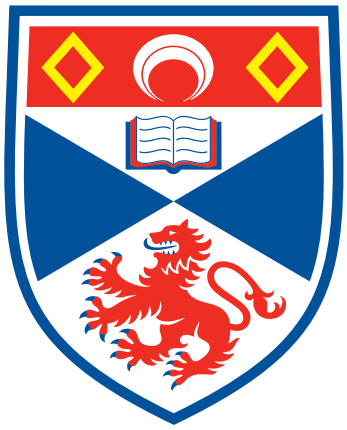
CS3099: Junior Honours Project

A Federated University Platform for Distributed Online Communities



University of
St Andrews

Kian Cross Robert Mardall Allan Mathew Chacko Lewis Mazzei

15 April 2021

Declaration

We declare that the material submitted for assessment is our own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 16117 words long, including project specification and plan.

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

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

Acknowledgements

We would like to extend thanks to our project supervisor, Professor Al Dearle, for providing support, guidance, and encouragement throughout the year. We would also like to thank Dr Edwin Brady for organising the module and ensuring its smooth running during difficult times.

Abstract

The hasty provisions induced by stay-at-home orders left a gaping hole in students' university experiences, including making friends, socialising, and networking. With little done by those who forced the move to address the issue, we set out to fill this gap, attempting to solve some of the perceived problems with contemporary social networking platforms in the process.

We have developed an online social-space targeting university students and staff, allowing access to content across a federated network of individually moderated communities. We hope that the decentralised nature may provide an alternative to many who feel that large technology companies can too easily suppress dissenting ideas and curtail vigorous debate, both of which are essential within academia.

Lead Authors

As per the requirements, each section[1] has a lead author. All members of the group have agreed that the table below is accurate.

Section	Name
1	Kian Cross
2.1	Lewis Mazzei
2.2	Allan Mathew Chacko
2.3	Allan Mathew Chacko
3.1	Robert Mardall
3.2	Kian Cross
3.3	Kian Cross
3.4	Allan Mathew Chacko
3.5	Robert Mardall
3.6	Lewis Mazzei
4.1	Kian Cross
4.2	Robert Mardall
4.3	Lewis Mazzei
5	Kian Cross
B	Lewis Mazzei
C	Allan Mathew Chacko
F	Robert Mardall
G	Kian Cross
I	Kian Cross
J	Lewis Mazzei
K	Robert Mardall
L	Lewis Mazzei
N	Kian Cross
O	Kian Cross
P	Kian Cross

Due to the iterative nature of the project, some sections of this report have been derived, or copied in whole from previous deliverables, which are included in Appendix A. In such cases, we have marked the section with a †.

[1] Appendixes referencing only external files have not been given a lead author.

Contents

1 Introduction **1**
1.1 Terminology . . . 2

2 Project Management **7**
2.1 Software Development Methodology . . . 7
2.1.1 Scrum Model . . . 11
2.1.2 Evaluation . . . 13
2.2 Development Tools . . . 14
2.2.1 Issue Tracking . . . 15
2.2.2 Project Boards . . . 15
2.2.3 Documentation . . . 15
2.2.4 Continuous Integration . . . 15
2.2.5 Code Reviews . . . 16
2.2.6 Testing Coverage . . . 17
2.3 Supergroup Interaction . . . 18
2.3.1 Evaluation . . . 21

3 Project Details **23**
3.1 System Functionality . . . 23
3.1.1 Core Requirements . . . 23
3.1.2 Extensions . . . 24
3.1.3 Supergroup Communication . . . 25
3.2 Software Architecture . . . 26
3.2.1 Overview . . . 26
3.2.2 Frontend . . . 27
3.2.3 Backend . . . 28
3.2.4 Infrastructure . . . 29
3.3 Machine Learning Spam Detection . . . 31
3.3.1 Input Data . . . 31
3.3.2 Training the Model . . . 33
3.3.3 Evaluating the Model . . . 35
3.3.4 Conclusion . . . 36
3.4 Group Video Chat . . . 37
3.5 Accessibility . . . 39
3.5.1 User Interface . . . 39
3.5.2 Mobile Users . . . 41
3.5.3 Colour-blind Users . . . 41
3.5.4 Screen Reader Users . . . 41

3.6 Security 41
3.6.1 Dependencies 42

4 Evaluation and Critical Appraisal **45**
4.1 Original Objectives 45
4.2 Specification 46
4.3 Similar Systems 55

5 Conclusion **59**

Bibliography **62**

Appendix A Interim Reports **63**

Appendix B Group Meeting Minutes **65**

Appendix C Project Plan **79**

Appendix D User Manual **81**

Appendix E Developer Documentation **83**

Appendix F Testing **85**

Appendix G Supergroup Meeting Minutes **89**

Appendix H Federation Protocol Specification **95**

Appendix I Protocol Improvements **97**

Appendix J User Stories & Requirements **99**

Appendix K Feature Descriptions **107**

Appendix L Privacy Notice **119**

Appendix M Source Code **123**

Appendix N Open Source Contributions **125**

Appendix O Deployment Plan **127**

Appendix P Technical Changes **129**

1. Introduction

On 15 March 2020, students at the University of St Andrews were informed that "there [would] be no classes or face-to-face teaching ... for the rest of [the] semester"[1]. This date marked an unforeseeable and unexpectedly long change to the lives of students at our university, and across the world. Decades of precedent that had set students' expectations were no longer applicable and new forms of socialising needed to be found.

For much longer, there have been concerns about the power individual social networking platforms hold and the lack of recourse individuals have if said companies make undesirable decisions. Naturally, social networking platforms believe the judgements they make to be justifiable and right. However, there have been high profile cases of politicians being deplatformed from notable social networks[2] and this has prompted a discussion regarding how much power and responsibility society is willing to give to these large technology corporations. There have also been concerns around prominent social networks making moderation decisions in collusion with each other. With more and more of our discourse transcending the traditional public sphere, where in many cases speech and assembly are protected by law, to spaces controlled by private enterprise, questions have been raised about the need for regulations to protect these freedoms. This was recognised by Shirky, who wrote (Shirky, 2011, p. 41):

> For good or ill, the platforms supporting the networked public sphere are privately held and run ... it is unlikely that without some legal framework, as exists for real-world speech and action, moral suasion will be enough to convince commercial actors to support freedom of speech and assembly.

We considered an alternative to regulatory enforcement: a federated network of individually moderated servers. With no central authority, many of the aforementioned issues are resolved. Nevertheless, limitations still remain. Powerful underlying infrastructure providers, such as internet service providers[3] and cloud platforms[4] can shutdown or prevent access to a service. This is somewhat of a separate issue and not one that we try to solve; we do not discuss it further.

[1] Said in an email from the Principal to students.

[2] We do not pass comment on whether these instances were justifiable or necessary, simply that they happened and that the precedent caused concerns.

[3] There have been high-profile campaigns to protect 'net neutrality', a principle that primarily benefits consumers, but also arguably helps to protect 'underdog' platforms.

[4] Parler — a controversial social media network — was recently dropped as a customer by Amazon Web Services and many of its other service providers, causing it to remain offline for a prolonged duration. We do not pass comment on whether this was justifiable or necessary, simply that it occurred.

Together, these issues form the core motivation for the project. Specifically, we were asked to implement a federated social network, such that no single organisation is responsible for the moderation of the platform. Content should be separated into 'communities' and available from all 'nodes' in the federated network. It was emphasised that our platform should be targeted towards a university environment and perhaps complement already existing educational platforms (rather than try to replicate them). The platform that we have developed satisfies all of the aforementioned requirements, as will be demonstrated throughout this report.

The language used so far is somewhat vague, so we start in Section 1.1 by clarifying the terminology used throughout. In the remainder of this report, we begin by discussing the management and organisation of the project (Section 2), such as the software development methodology and development tools employed. We then move onto details about the project (Section 3) and talk about the features that we have implemented, before carrying out a critical evaluation of our work (Section 4), where we compare our platform with other similar systems and the given requirements. Finally, we conclude with a summary of further possible avenues for development (Section 5).

1.1 Terminology[†]

Before continuing, we will use this section to clarify the language used throughout this report.

We use 'social network' to mean an "application which enables users to communicate with each other by posting information, comments, messages, images, etc." ("Definition of social network", 2020). By 'federated', we mean decentralised across separate and distinct organisations, where users of each organisation can access content from other organisations. We define some more terms in Table 1.1.

Table 1.1: Definitions of key terms.

Server	The highest level entity in the network, representing a logical unit that other servers may communicate with to form a federated network.
User	An entity representing a user (a student or a member of staff, for example), belonging to an individual server. Users can not sign-in to other servers but may access their content by utilising the federated features of the server.
Community	A sub-section of a server consisting of a collection of related content.
Post	A post is the basic unit of content. They can be nested, with non-root posts being considered comments associated with some root post. This structure allows for comments to form a tree structure.

[†]This section of our report is a derivative of a previously submitted appendix.

Local server	Relative to an entity, the local server refers to the server that such an entity belongs to.
Remote server	Relative to a local server, a remote server is any other server in the federated network.
Moderation	Moderation is the process of removing inappropriate content from a community and, if necessary, banning particular users from contributing to a community.
Administration	An administrator is a privileged user, belonging to one or more communities on either the local server or remote server. They are able to moderate content within the community.

Figure 1.1 intends to clarify some of this language. Server A is the local server for users $A_1, \ldots, A_n$ and communities $A_1, \ldots, A_m$. For the aforementioned users and communities, servers B and C would be remote servers. Servers, communicate with each other to form a federated network. Users from each server may access content from communities on other servers.

Figure 1.2 is the same diagram, but provides a concrete example of how servers and communities could be organised within a university.

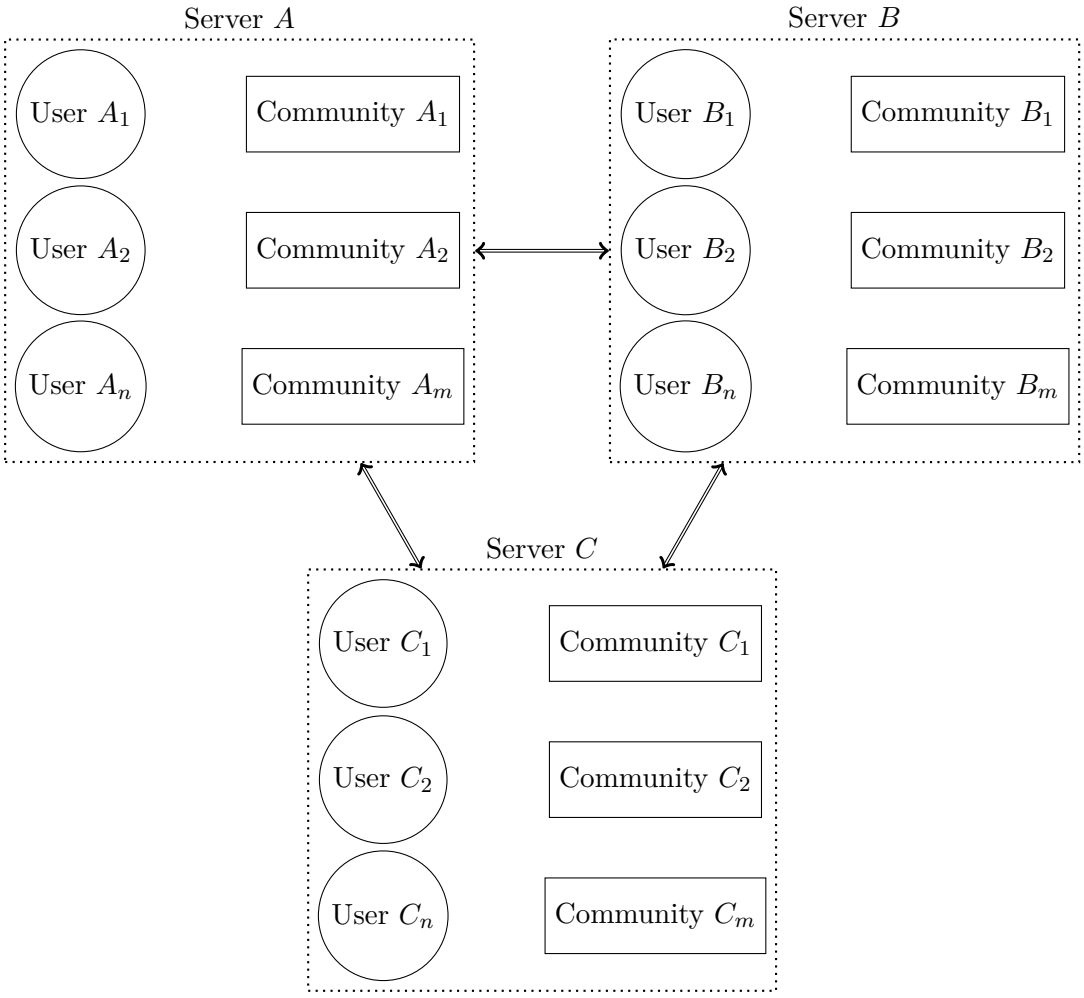


Figure 1.1: Relationships between servers, communities and users in the federated network.

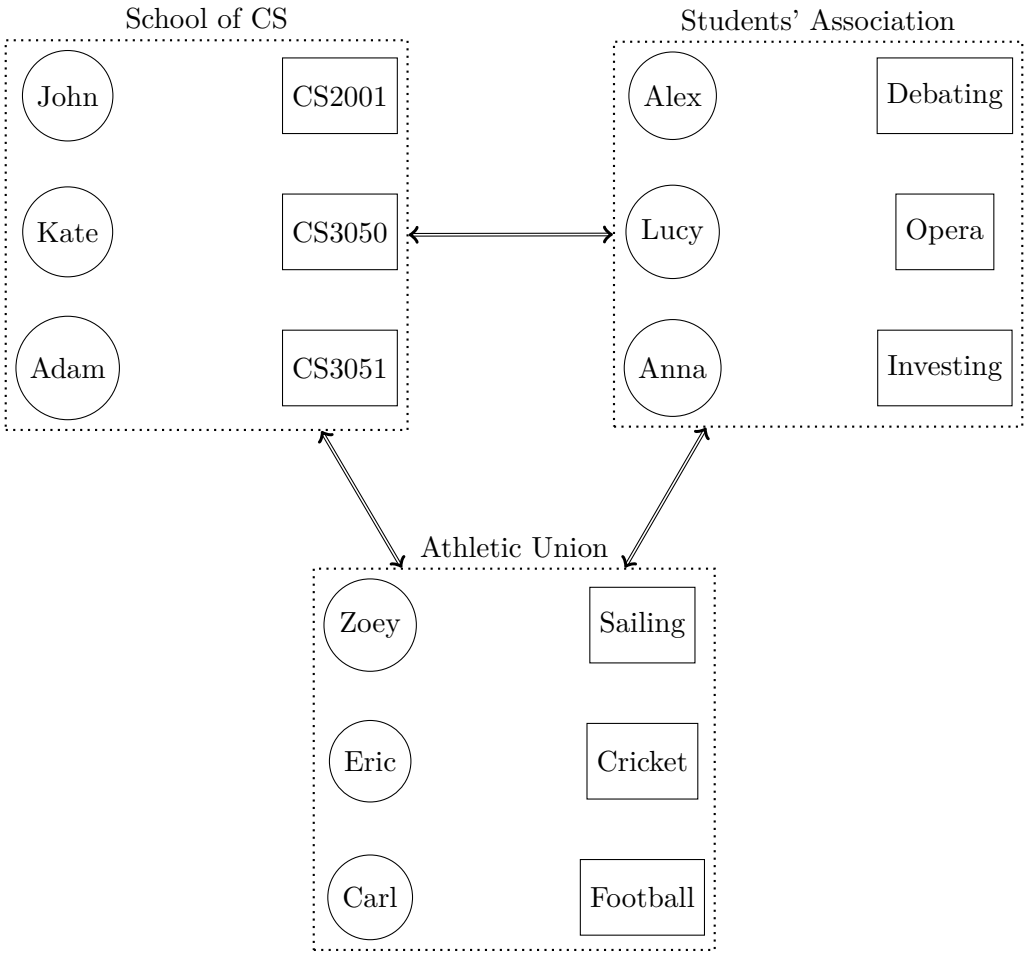


Figure 1.2: Relationships between servers, communities and users in the federated network (a concrete example).

2. Project Management

In this part of our report, we discuss the organisational aspects of the project. We start by outlining the software development methodology that we have followed, before moving onto an analysis of our development workflow, including the third-party tools that we have utilised. We finish with an overview of our work with the supergroup.

2.1 Software Development Methodology

Selecting an appropriate software development methodology is an important decision to make before starting a project. A methodology used for one project may not be right for another and so it is necessary to carefully consider the approach before adopting a particular process.

Throughout the year, we have been required to provide evidence of following the Scrum model. Naturally, therefore, this is the methodology that we have used. Nevertheless, we will briefly discuss other potential options and why they may or may not have been appropriate for our project. In Section 2.1.1, we will then explore the agile Scrum process that we have utilised in more detail.

A common alternative to Scrum is the waterfall model, with the well-known diagram first being attributed to Royce, 1987. There are a series of steps, including generating requirements, an analysis, coding, testing and documentation that are followed sequentially. The approach is easy to understand and places an emphasis on planning and documentation. However, it relies on being able to enumerate the requirements at the beginning of the project, which is not always possible. In some cases, the rigidity of the process is a feature, however it is often an unwanted restriction. In the foreward to Benington, 1987 — an adaptation of a presentation from 1956 laying out many of the steps used in the waterfall methodology — the importance of a prototype is stated:

> We undertook the programming only after we had assembled an experimental prototype ... that performed all of the bare-bone functions ... Twenty people understood in detail ... what each module would do, they understood the interfaces, and they understood the performance requirements. People should be very cautious about writing top-down specs without having this detailed knowledge ...

We discuss how we incorporated this emphasis on a prototype into our process in Section 2.1.2.

The spiral model, first described in B. W. Boehm, 1988, primarily focuses on handling risk. At each cycle, possible risks are evaluated and these determine the

development processes to use. This gives the spiral model the more accurate description of a "risk-driven process model generator". There is far more to be said about the spiral model, however, the main takeaway is that "different risk patterns can lead to choosing incremental, waterfall, evolutionary prototyping, or other subsets of the process elements in the spiral model diagram" (B. Boehm & Hansen, 2000, p. 3). This is a more complex procedure; processes can change significantly from one cycle to the next as certain risks increase and others decrease.

Whilst we did not formally follow the spiral model, we considered potential risks at different stages of the project and implemented procedures to mitigate these. Table 2.1 outlines these factors.

Table 2.1: Identification and mitigation of project risks.

Week	Risk	Mitigation(s)
4	**Learning and adopting new technologies.** We were unfamiliar with many of the technologies that we selected for the project. This was a potentially significant risk, as libraries and frameworks could contain bugs, be more complicated than expected, or not provide the solutions that we had thought they would.	We carefully researched all of the frameworks and libraries we intended to use so as to minimise the chances that we had misunderstood their features. As part of this research, we also looked at their open-source repositories, checking: how many open issues there were, how many open pull requests there were, how frequently the repository was committed to, when the last commit to the repository was, and if there were any unaddressed CVEs. By checking the open-source repositories, we could assess whether the project was actively being maintained, minimising the risk of encountering bugs or security issues.
4	**Working together as a new team.** We had not worked together as a team before. This introduced the risk of character incompatibilities or general personality disagreements.	Whilst this was only ever a small risk, we mitigated it by acting professionally, giving each other autonomy over parts of the project, and getting to know each other better throughout the year.

Continuation of Table 2.1.

Week	Risk	Mitigation(s)
4	**Adapting to unfamiliar working environments.** Access to the School's computer labs was restricted due to physical distancing rules. This also meant we could not work with each other in person. This introduced the risk of team members getting stuck whilst setting up software on personal computers.	We mitigated this risk by ensuring that our software stack was easy to set up and worked on all platforms. This was achieved using Docker. We also had regular meetings for solving technical issues, which replaced in-person coding sessions.
4	**Potential for requirements to change.** We had a fairly good idea of the requirements when starting the project. However, we were warned that a 'twist' might be thrown in after the inter-semester break. This introduced an unknown risk that depended on how significant the change would be. As it turns out, no such change was introduced, nevertheless, we were prepared for it.	We ensured that our code was written in a decoupled, modular fashion, which would have allowed for easy refactoring if we needed to make significant changes. We also ensured we had left enough time in our schedule for unexpected requirement changes.
4	**Running out of time.** We only had a few weeks to implement the minimum viable product. There was a risk that we would not meet the deadline if we planned too many features for completion.	We minimised this risk by prioritising essential features first, such that if we were to run out of time, we would still have a prototype meeting all of the requirements. We were also careful not to overestimate how quickly we could programme, which is a common pitfall in software development.

Continuation of Table 2.1.

Week	Risk	Mitigation(s)
7	**Potential for the federation protocol to change.** Having agreed on a basic protocol for communication between federated instances, there was the risk that when implementing it, we would find it to be inadequate. This could require large code changes and be a significant loss of time.	We implemented the least amount of the protocol possible until it was clear it was not going to change. We also wrote the federated part of our application as a completely separate module, meaning we could change the implementation without affecting any other parts of the application. This decoupling made potential major changes in the future less risky.
11	**Inappropriate proposals being added to the federation protocol.** Having completed the foundations of the protocol, many groups were proposing their own ideas and enhancements. There was a risk that poor quality proposals could make their way into the protocol. Additionally, it was possible that the protocol could become bloated with too many features to implement.	We minimised this risk by carefully reviewing each and every proposal made to the protocol. We also attended each supergroup meeting so that we knew what was under discussion.
13	**Forgetting things over the inter-semester break.** Between the minimal viable product and the start of Semester 2, there were approximately 8 weeks. Therefore, it was possible that at the beginning of Semester 2, we would be slow to gain momentum.	We minimised this risk by doing some work over the inter-semester break. We also formally restarted sprints two weeks before the start of the semester to ensure that we were back into the swing of things.
23	**Running out of time.** With only a short amount of time before the final deadline, there was a risk we would spend too much time on development and not enough on testing and documentation.	We minimised this risk by setting a hard deadline for implementing new features and then focused on testing and documentation beyond this date.

End of Table 2.1.

2.1.1 Scrum Model†

Having discussed two other software development processes, we now describe Scrum, an agile model, which we have followed throughout the project's development. This process is described in Schwaber and Sutherland, 2017, where they write that Scrum is a "framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value". First, we outline the formal process, then cover how (and why) we have adapted it slightly.

Roles

We start by describing one of three main components in the framework: roles. Table 2.2 outlines the responsibilities of each team member and the work they will be expected to perform.

Table 2.2: Role definitions.

Development team	The main responsibility of this team is to deliver an increment to the product every sprint. Work is delegated equally and accountability belongs to the team as a whole.
Product owner	The goal of the individual in this role is to optimise the product that results from the work of the development team. They manage the product backlog, ensuring that it is up to date and comprehensively understood by the development team.
Scrum master	The scrum master's duty is to promote and support the framework described in this section, making sure that the responsibilities of each role are adhered to, events are carried out in a suitable manner and that artifacts are correctly produced. This involves the planning and facilitation of events, ensuring that all members of the development team fully understand their tasks and helping to resolve blockers as and when they arise.

As we were only a small team of four working on the project part-time, we did not assign any individual permanently as product owner or scrum master. For the former, we all contributed to organising the backlog, selecting stories for each sprint, and ensuring that the short and long term direction for the project was well understood. For the latter, various responsibilities from the role were adopted organically by different members of the team within the first few weeks of us working together. This worked for us, however a more rigid structure may have been necessary for a larger group.

†This section of our report is a derivative of the software development methodology plan, submitted in the first deliverable.

Events

In Table 2.3 we define events, which take place in the form of meetings. They are attended by all members of the team and are facilitated by the scrum master.

Table 2.3: Event definitions.

Sprint	A sprint is a unit of time in which an increment to the product is created. This amount of time remains constant throughout the project. A new sprint commences immediately after the previous one concludes.
Sprint planning	This event is the first phase of a sprint and is performed on the first day. The team must agree on a well-defined sprint goal, which is carried out by pulling a set of user stories from the product backlog, splitting each into several tasks, and then delegating these tasks to members of the development team (usually based on their individual skill-set). These tasks are stored in a sprint backlog that the development team updates throughout the sprint.
Daily scrum	This event usually occurs on every day of a sprint and involves each member of the development team describing what they did the previous day and what they will do that day to make progress on their assigned tasks.
Sprint review	This event occurs on the last day of a sprint and involves making changes to the product backlog based on the experience and outcomes of the previous sprint.
Sprint retrospective	This event also occurs on the last day and involves reflecting on the sprint that has just occurred. Each member of the development team should contribute at least one positive and one negative aspect of the process so that improvements can be made in the next sprint.

As we were only working on the project part-time, we compressed some of these events together. Each sprint was a week, starting on Monday and ending on Sunday. The 'daily' scrum took place every Thursday; we scheduled optional meetings for the remaining days, which were used to resolve blockers as and when they arose. Sprint review, planning and retrospective all took place on Monday, at the start of each sprint. Appendix B contains minutes from all of our group meetings.

During the planning meetings, we assigned story points to each task, which represented the comparative difficulty to complete a particular story. This was carried out by asking each member of the group to assign points to the story, with those who had assigned the largest and smallest number asked to justify their reasoning. We repeated this until we all reached an agreement. Each member of the group was then assigned a roughly equal number of story points in order to help distribute the workload evenly.

Artifacts

Finally, in Table 2.4, we describe artifacts. These are documents designed to make development decisions transparent and visible to all members of the team.

Table 2.4: Artifact definitions.

Product backlog	This is the single source of requirements for any changes to the product. It is a dynamic, evolving document consisting of an up-to-date set of user stories, listed in terms of priority.
Sprint backlog	This document contains a subset of the product backlog and is made up during the sprint planning event. Each story is split into smaller tasks that are then assigned to members of the development team for the duration of that sprint.

As a group, we kept the product backlog updated and decided together what to include in each sprint backlog.

2.1.2 Evaluation

Having described the Scrum process, we now evaluate the parts that went well and aspects that could be improved.

We believe that the focus on a minimum viable product (or prototype) helped to provide solid foundations that we built on throughout the year. Having a working product allowed us to iteratively make improvements, which was beneficial for multiple reasons. First, it allowed us to understand the problem to a much greater extent, so we could better estimate how much time was required for further features. Second, it provided an opportunity for us to validate that the product was on track to meet the requirements.

Generally, our time management was good; for example, we did not miss any deadlines. Figure 2.1 shows the weekly code contributions throughout the year. Demonstrations in Week 10 and Week 24 correspond to the noticeable activity spikes in Week 9 and Week 23. This suggests (and we can confirm) that there was a disproportionate amount of work done shortly before these demonstrations. Whilst this is not necessarily unexpected, it does indicate that better planning could have reduced stress and pressure at these points. Ideally, Figure 2.1 would be even throughout the year.

We have two explanations for this. First, we underestimated both how much was required and how long said requirements would take to implement. Second, we left testing too late and were fixing bugs at the last minute. Ideally, we would have identified this pattern during the project and discussed it in our sprint retrospectives; we could then have worked to mitigate it. The fact that we did not realise suggests that in the future we should put more of an emphasis on evaluating ourselves throughout the project (rather than simply at the end). This way we can fix issues as we go along and benefit from the improved efficiency.

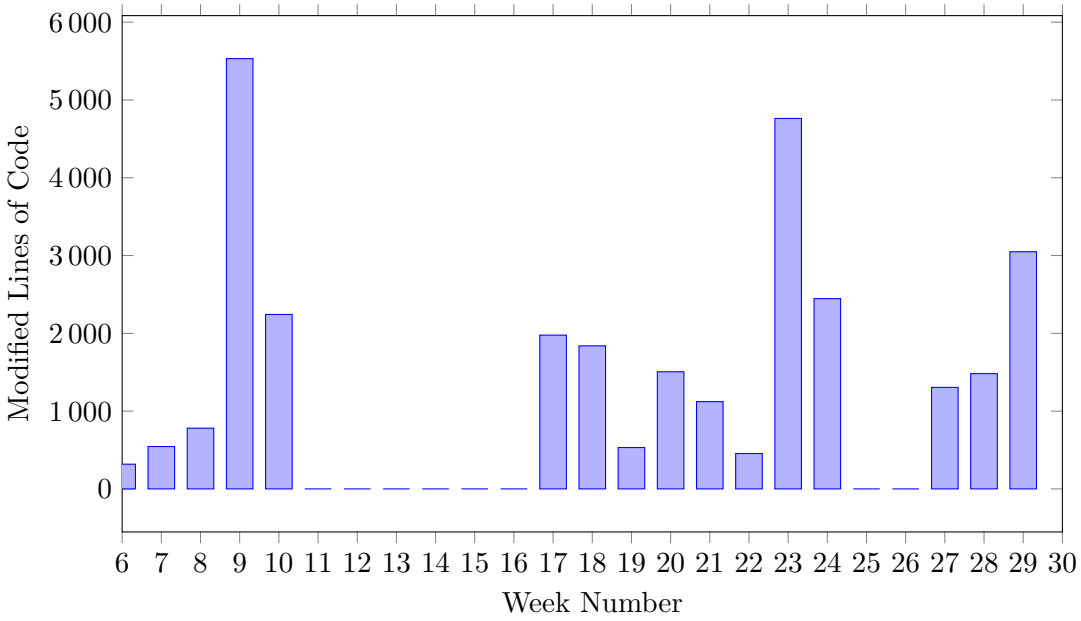


Figure 2.1: Graph showing the project activity throughout the year. For reference, Week 20 is the first week of Semester 2. *This part of the report was written in Week 30 and so data from the final two weeks of the project is unavailable at the time of writing. Nevertheless, this is sufficient for the point we make.*

In future projects, it would probably be beneficial to have an additional meeting, bringing the total to three per week. These meetings are not usually long, however touching base slightly more often may have helped with some of the aforementioned time management aspects. Particularly, being able to see any delays that were occurring sooner rather than later would have allowed us to act more proactively rather than reactively.

As can be seen from our project plan in Appendix C, we divided responsibility for the frontend and backend between us. We think this was a good idea, as it allowed each of us to specialise in a particular area of the project and avoided us all having to learn multiple new technologies. However, this plan became somewhat strained when one member of our group took a leave of absence. As this happened towards the start of the project, the impact was small and we were able to re-evaluate our approach. It highlighted the importance of keeping all members of the team in the loop so as to allow smooth transitioning between roles, if required.

Despite these areas of improvement, we believe we worked well as a team and consider our implementation of the Scrum methodology to have been successful. Throughout the project, all members of the group understood the short term goals and long term direction, suggesting that we were on the same page and communicating well. We all actively participated in both project management and development; this individual investment meant we all cared about the project's success.

2.2 Development Tools

The previous section described the development methodology that we have followed. We have also used a variety of tools and processes to supplement this. We now outline

these aspects and discuss how they have assisted us.

GitHub has been an integral part of our development workflow and has done far more than just provide source control. We have also used it for issue tracking, project management, documentation, continuous integration, and code reviews. Our use of Dependabot, which is also builtin to GitHub, is outlined in Section 3.6.

We initially selected GitHub over the School's GitLab[1] instance as the GitLab GUI was not available outside of the School's network without a SOCKS proxy. Even if we were not programming every day, our team wanted to encourage habits such as reviewing code, responding to issues and checking the status of the sprints regularly. We felt that this additional barrier to accessing the GitLab GUI may discourage this type of habitual engagement.

2.2.1 Issue Tracking

The issue tracker integrated into GitHub is where we tracked all bugs and features. These could easily be searched and filtered, which made it convenient for us to see who was working on what. Tags were used to categorise issues, such as 'backend' and 'frontend'.

When we encountered bugs in the application, we used issues to store details, including aspects such as: steps to reproduce, warnings, and errors. This ensured that we did not lose or forget useful information. Overall, our issue tracking was effective and contributed to the success of the project.

2.2.2 Project Boards

For organising our product and sprint backlogs, we used GitHub's project boards, an example of which can be seen in Figure 2.2.

From the product backlog, stories were converted to issues and then added to a sprint backlog. These boards were a convenient way of tracking the tasks that were completed in each sprint.

Having the product and sprint backlogs easily accessible and displayed in a clean and clear way ensured that all members of the group understood both the current focus of the project and also the long-term direction.

2.2.3 Documentation

We used both the repository's wiki and GitHub Pages[2] for documentation. The wiki was used throughout the year for storing meeting minutes, report plans, and other miscellaneous documents. The wiki is version controlled and was easily available to the whole team, which made it particularly convenient.

GitHub Pages was used for hosting our user manual and developer documentation (which can be found in Appendix D and Appendix E respectively).

2.2.4 Continuous Integration

The GitHub Actions[3] feature was used to automatically test our application. We set up a continuous integration flow, which was executed on every pull request. This

[1] We regularly synchronised our GitHub repository with the School's GitLab.
[2] https://pages.github.com/
[3] https://github.com/features/actions

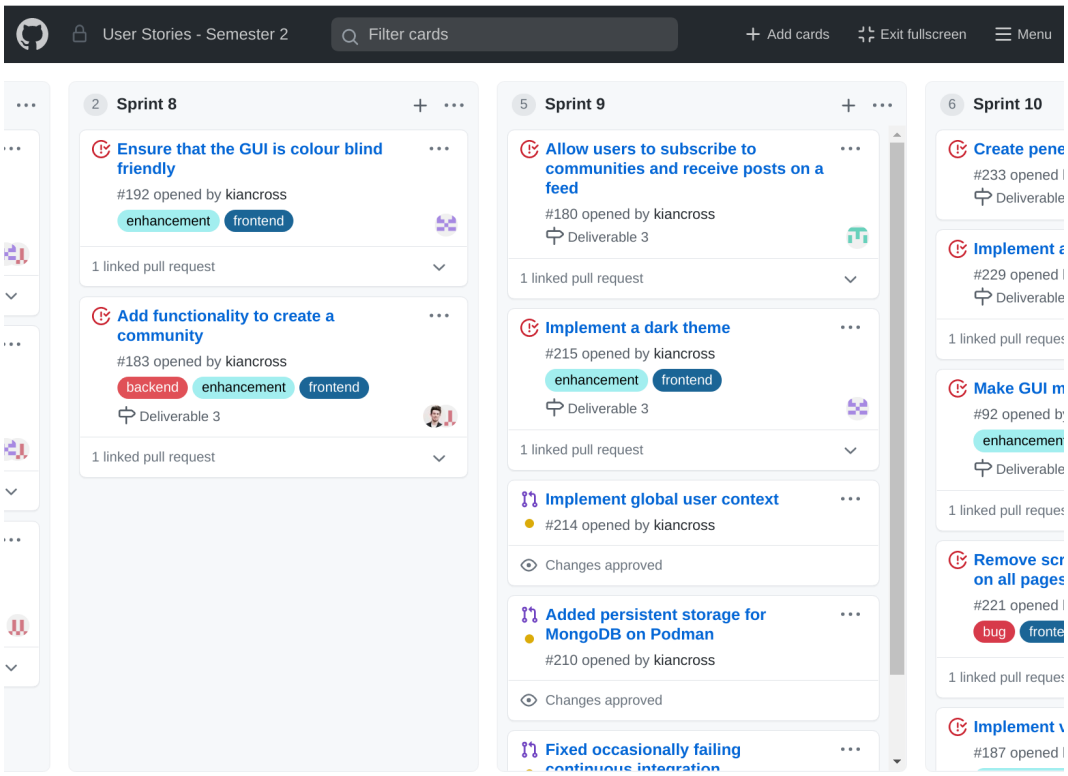


Figure 2.2: Example of the GitHub project board we used to organise our user stories and sprints.

ensured that our code was well tested and regressions were spotted immediately. Figure 2.3 shows an example of a continuous integration run.

Our continuous integration flow did more than just run automated tests. We also used a tool to keep our code style consistent. This is particularly helpful when multiple people are working on a project, as otherwise code style across the codebase can diverge. Preventing this improves code readability and helps when refactoring. Another advantage to a consistent style is that it reduces diffs to only the functional changes, rather than superfluous code style differences.

Our use of GraphQL allowed us to verify that the interface between the frontend and backend code was correct. As part of our continuous integration flow, we generated the GraphQL schema implemented by our backend, then checked that the frontend queries conformed to this. Critically, this covers a weak point in many applications' testing: the API communication between components.

During the continuous integration flow, the code is also statically analysed to preemptively find any problems, the documentation is generated, and a build is carried out to ensure that there are no compilation errors.

2.2.5 Code Reviews

Each pull request was reviewed by at least one other member of the team. GitHub provides useful features for doing this, allowing detailed discussions about aspects of code. Part of a review can be seen in Figure 2.4.

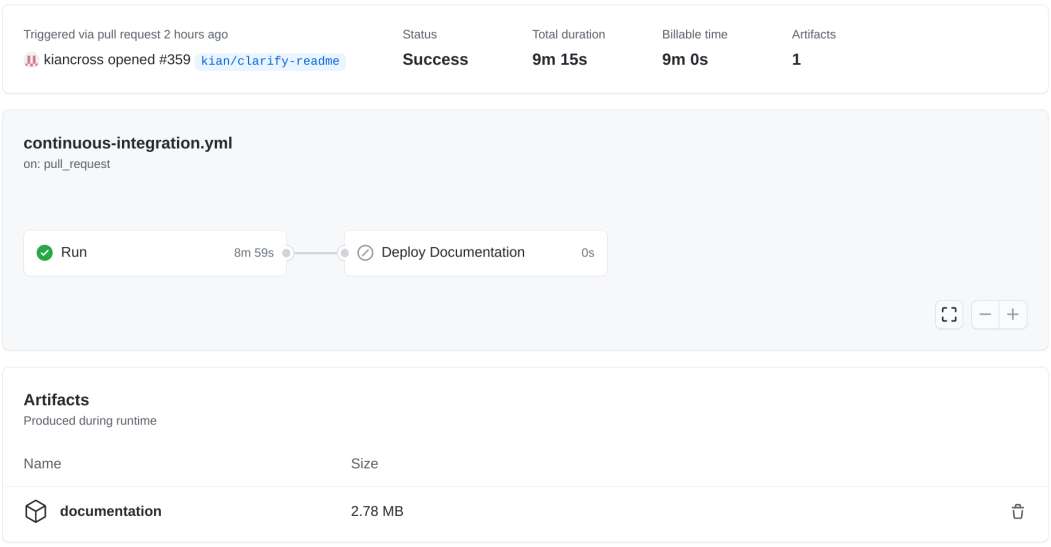


Figure 2.3: Summary of a continuous integration run.

The reviewer can double-check that the test coverage is suitable[4] and that the code is well designed before marking it as ready to be merged. The review was also a good opportunity to check that the commit messages were clear before being merged into the master branch. We endeavoured to write detailed commit messages to ensure that we were properly tracking changes to our code, with appropriate reasons and explanations for each change.

We wanted to encourage engagement from all members of our group in different aspects of the project, even if they were not directly involved in the development of said aspects. We think code reviews were a good way to achieve this, as it gave exposure to all parts of the system.

We were initially concerned that code reviews might slow us down too much. However, in the long term, having a second pair of eyes look over the code probably caused fewer bugs and produced a higher quality codebase. There is certainly a balance to strike between being overly picky in a code review and letting poor quality code slip through. Overall, we think we achieved this balance.

2.2.6 Testing Coverage

On each pull request, our automated test suite was run and a coverage report was generated. This allowed us to easily review how well tests covered a particular feature and ensured that there were no regressions. We used a service called Codecov[5] for this, which automatically adds a comment to each pull request with a summary of the test coverage. This can be seen in Figure 2.5. In addition to these summaries, Codecov provides a browsable coverage report, that allows lines of code not covered by tests to be easily identified. An example of this can be seen in Figure 2.6.

At the start of the second semester, we overhauled our testing strategy. We ceased using our end-to-end testing framework, for reasons explained in our third

[4] 'Suitable' was defined as at least a coverage percentage of 85%, unless there was an exceptional reason that made this impossible or impractical.

[5] https://codecov.io/

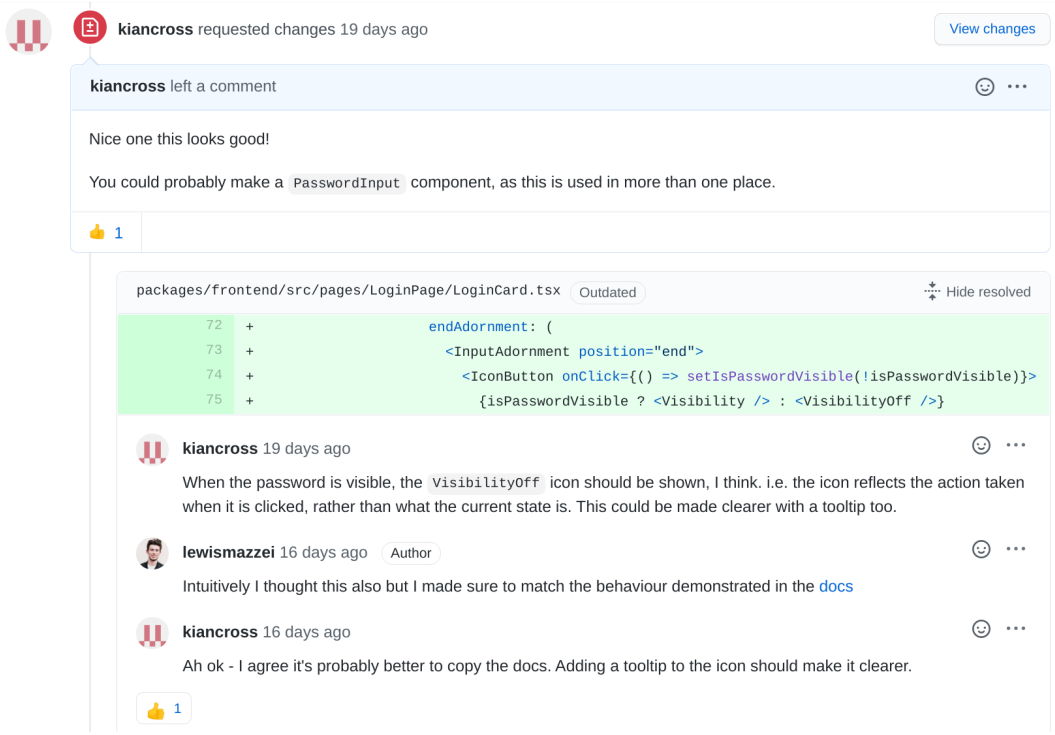


Figure 2.4: Example section of a code review on a pull request.

deliverable, and focused on unit and integration tests. Reimplementing the testing was not ideal, as it took substantial time. In the future, it would be beneficial to spend more time considering our testing strategy at the start, in order to get it correct from the offset. However, it was better to correct it late, rather than never. We provide more details about testing in Appendix F.

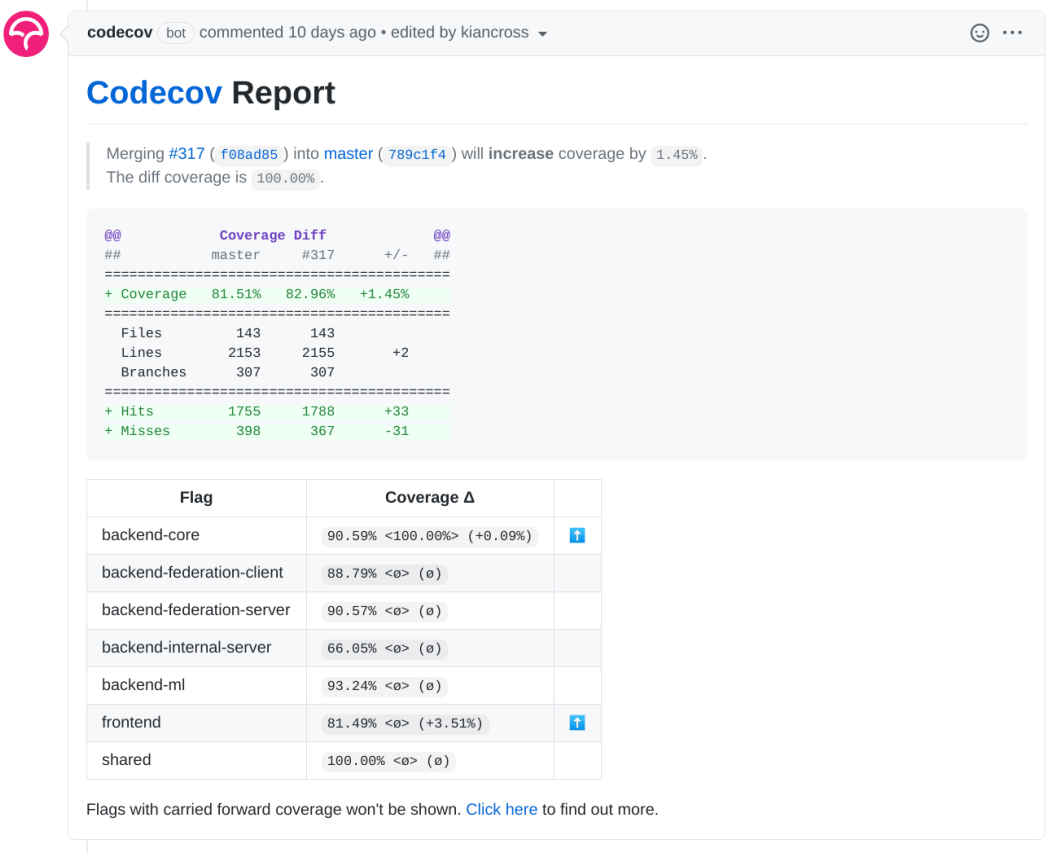
Testing coverage was one of the most valuable tools we used, as it allowed us to quantify to some degree how good our testing was, using the coverage percentage[6]. However, as with any sort of statistical measure, it is important not to fall victim to Goodhart's law: "any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes" (Goodhart, 1984, p. 116). The goal should not be to increase the test coverage percentage at all costs, but instead to have a well-tested application with few regressions. Figure 2.7 shows how the test coverage percentage changed across the second semester. After removing our end-to-end tests and restarting with unit tests, we can see it was rather low. However, throughout the semester it grew substantially, to over 83%. Using this statistical measure aided us in accessing ourselves regularly throughout the project: in this case, our testing strategy in particular.

2.3 Supergroup Interaction†

The protocol used for servers to communicate within the federated network was a key component of this project. A supergroup (consisting of 12 groups) was formed

[6]This was the percentage of code branches covered with tests, out of the total number of code branches.

†This section of our report contains aspects from previously submitted deliverables.



codecov bot commented 10 days ago • edited by kiancross

Codecov Report

Merging #317 (f08ad85) into master (789c1f4) will **increase** coverage by 1.45% .
The diff coverage is 100.00% .

```
@@            Coverage Diff             @@
##           master     #317      +/-   ##
==========================================
+ Coverage   81.51%   82.96%   +1.45%
==========================================
  Files         143      143
  Lines        2153     2155       +2
  Branches      307      307
==========================================
+ Hits         1755     1788      +33
+ Misses        398      367      -31
```

Flag	Coverage Δ	
backend-core	90.59% <100.00%> (+0.09%)	⬆️
backend-federation-client	88.79% <ø> (ø)	
backend-federation-server	90.57% <ø> (ø)	
backend-internal-server	66.05% <ø> (ø)	
backend-ml	93.24% <ø> (ø)	
frontend	81.49% <ø> (+3.51%)	⬆️
shared	100.00% <ø> (ø)	

Flags with carried forward coverage won't be shown. Click here to find out more.

Figure 2.5: Example of a code coverage summary on a pull request.

to develop a specification to achieve this. Here, we describe the process of developing this protocol and our interactions with the supergroup over the past year.

Communications have taken place entirely remotely using Microsoft Teams. Meetings have occurred weekly between nominated representatives from each group, for which minutes can be found in Appendix G. Representatives take their group's views to the supergroup meetings and act as advocates for issues that they consider important. Key decisions are then communicated back at regular group meetings.

As might be expected, the internal workings of the supergroup changed substantially throughout the year, becoming significantly more efficient by the end. Our first meetings were chaotic and had little direction. We decided we needed to formalise



```
57
58  1        return await processRawPost(rawPost, host);
59         } catch (error) {
60  1        if (error.response && error.response.statusCode === 400) {
61  1          throw new RemoteResponseError("Invalid permissions");
62           } else {
63             throw new RemoteResponseError("Unknown error");
64           }
65         }
66       }
```

Figure 2.6: Code coverage report demonstrating a line of code not covered by tests.

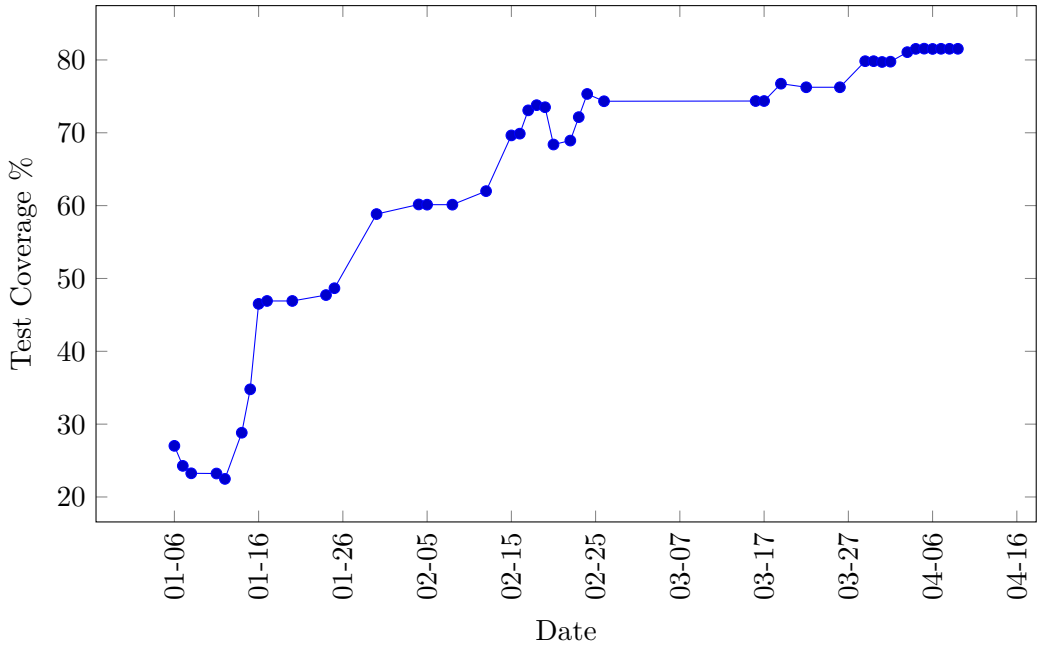


Figure 2.7: Graph showing the test coverage percentage throughout Semester 2. *This part of the report was written in the penultimate week of the project and so data from the final two weeks is unavailable at the time of writing. Nevertheless, this is sufficient for the point we make. Updated figures will be provided in Appendix F.*

meetings and came up with the following process:

1. Before each meeting, via messages, we agreed upon an agenda.

2. Each meeting had a 'chair' who invited people to speak (indicated by 'raising their hand' on Teams) and kept the conversation on topic.

3. If there was significant disagreement, those with strong views either way posted their arguments in a particular Teams channel, where discussion could continue via message.

4. If we ran out of time and there were important aspects left to discuss, we scheduled a second meeting later on in the week.

Meetings had far more direction after implementing this process.

By the end of November, we had agreed on a protocol and implemented a basic version of it. Hereon, the focus of the supergroup changed to iteratively improving the specification. As before, we found inefficiencies in our process and so adopted the steps in Figure 2.8. The completed protocol can be found in Appendix H.

Our group has been particularly active within the supergroup, with 18 out of the 35 approved changes made by us. Additionally, we have reviewed all of the other proposals. By doing this, if something inappropriate was added to the protocol (and we had to waste time implementing it), we would only have had ourselves to blame.

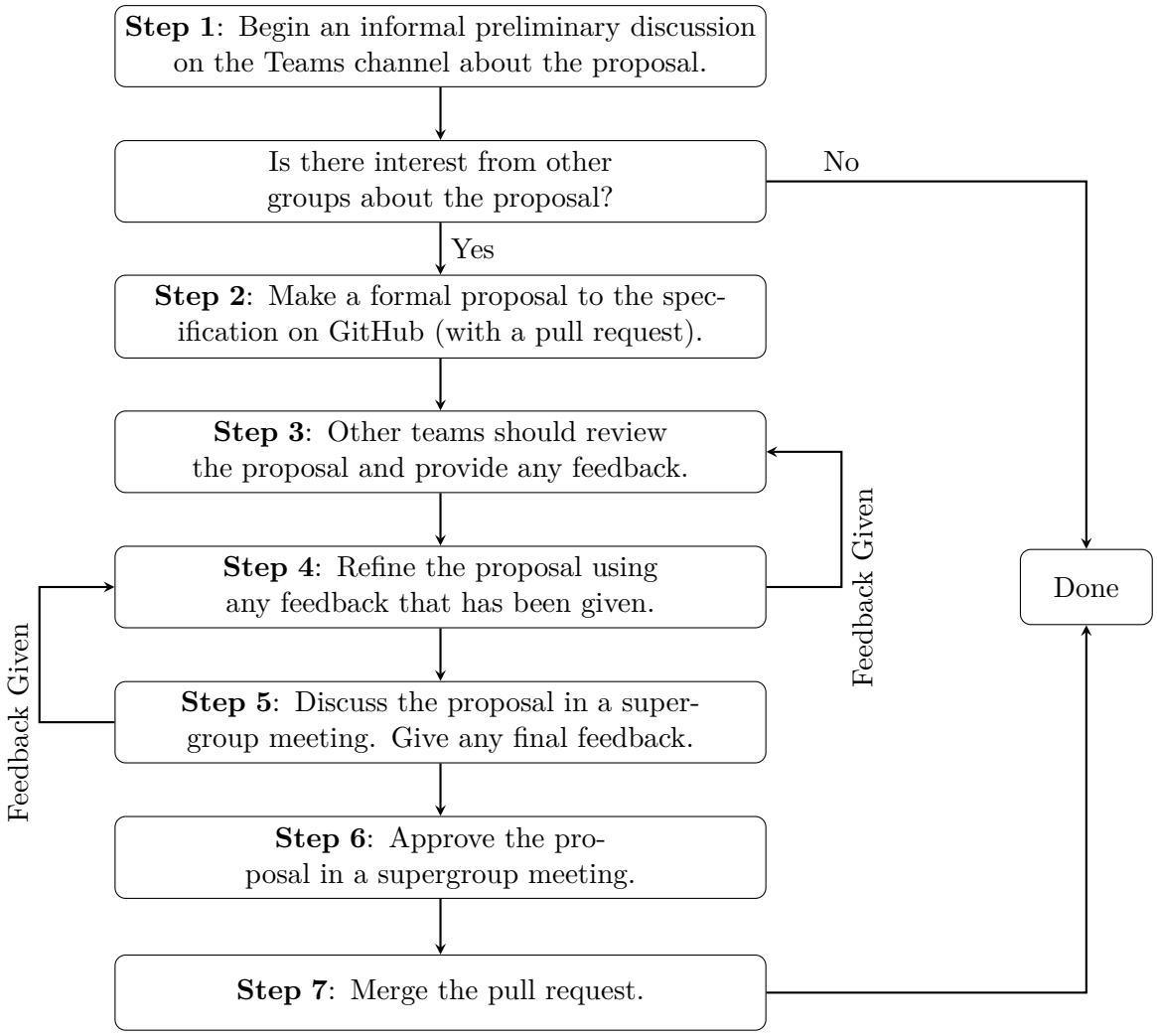


Figure 2.8: Iterative supergroup workflow for building upon the protocol foundations.

2.3.1 Evaluation

We were rather introspective throughout the year and made improvements as we went along; this can be seen above. Whilst this helped us throughout the year, it means that there is less to evaluate now.

In hindsight, a set of automated tests that all groups could run their implementation against may have improved efficiency. These would have been the definitive source for how the public federation interface behaved. We spent a significant amount of time testing our implementations together, which automated testing could have reduced. We did discuss this at the start of the project, however, the upfront investment was off-putting. Were we to do a similar project again, we would use some of the time we spent on internal automated tests on ones that could be used by all groups.

Generally, we think we worked together well as a supergroup and are happy with the protocol we produced. We have evaluated the technical aspects separately in Appendix I.

3. Project Details

Having described the organisational aspects of our project, we now move onto discussing some of the technical details. We begin with an outline of the features we have implemented and give an overview of our software stack and architecture. Finally, we explore two of our more technically complex features, before concluding with an outline of our accessibility and security considerations.

3.1 System Functionality[†]

This section of our report outlines the implemented functionality of our system, including features that make use of supergroup communication.

The full set of requirements can be found in Appendix J. Appendix F describes our testing procedures and how we have validated that these features work as intended. Finally, in Appendix K we give far more detailed descriptions of the features below, with test cases.

3.1.1 Core Requirements

We were provided with a set of core requirements to implement for the minimum viable product:

- ability to register users;
- creating new posts;
- creating comments on posts;
- a simple web-based user interface and
- an implementation of a protocol for propagating content between instances.

Based on these core requirements, we have implemented the features in Table 3.1.

Table 3.1: Core features.

Registration	Users can register an account, which gives them access to the features of our application, such as reading and creating posts.

[†]This section of our report is a derivative of the feature summary submitted in the third deliverable.

Sign-in	After registration and email verification, users can sign-in to our application to access the remaining features.
Creating a Post	Users can create posts using an extended markdown format (which includes TEX maths support).
Creating a Comment	Similarly to posts, users can create comments using an extended markdown format. Additionally, comments can be nested, allowing extensive discussion between users on a post.

Our application is accessible through a web-based interface.

3.1.2 Extensions

In addition to the above core requirements, we have implemented multiple additional features, found in Table 3.2. Those that we consider significant are marked with a ♦.

Table 3.2: Extended features.

Editing Posts and Comments	Posts and comments that users have made across the federated network can be edited by the user who created them, or an administrator of the community.
Deleting Posts and Comments	Posts and comments that users have made across the federated network can be deleted by the user created them, or an administrator of the community.
Password Reset	If a user forgets their password, they can reset it by requesting a password reset link.
View Profile	Users can visit the profile pages of other users, which displays information about the user, including the posts that they have made to different communities.
Account Settings	Users can change their account information, such as name and password, by using the 'account settings' page.
Creating a Community	Users can create a new community for a topic which interests them.
Searching for Communities	Communities across the federated network can be discovered by searching for a particular host in the network.
Privacy Notice	We have written a privacy notice as part of our commitment to upholding the seven GDPR principles. This can be found in both the application and Appendix L.

♦ **Group Video Chat**	Each community has a group video chat, the idea being to emulate a 'common room' or 'coffee room', where people can come and go as they wish. Further information is available in Section 3.4.
♦ **Machine Learning Spam Filter**	We have implemented a machine learning spam detection filter, using a model we have implemented and trained ourselves. Posts detected as spam are marked as requiring administrator approval. Further information is available in Section 3.3.
♦ **Text Toxicity Detection**	We are utilising a text toxicity classifier, which can detect posts that fall into the following categories: identity attack, insult, obscene, toxic, and sexually explicit. We use this to mark posts as requiring administrator approval before displaying them to users.
♦ **Reporting and Moderation Queue**	Users can report posts and comments, flagging them for the attention of the community's administrators. Flagged posts appear in a queue for administrators to view.
♦ **Nested Comments**	Users can reply to comments, which generates a nested comment structure. These are displayed as separate indented levels to the user.
♦ **Dark Mode**	We have developed a dark mode theme, which can be toggled on and off by the user.
♦ **Subscriptions and Feed**	Users can subscribe to particular communities, which puts recent posts on a feed for the user to browse.
♦ **Colour Blind Friendly GUI**	We have ensured that our user interface is colour-blind friendly. More details can be found in Section 3.5.
♦ **Screen Reader Friendly GUI**	We have ensured that our user interface is screen-reader friendly, to a certain extent. More details can be found in Section 3.5.
♦ **Mobile Friendly GUI**	We have made our user interface responsive and mobile friendly. More details can be found in Section 3.5.

3.1.3 Supergroup Communication

Many of our features are interoperable with other instances, using an implementation of the protocol in Appendix H. These features are listed below:

- searching and viewing communities;
- creating posts and comments;
- deleting posts and comments;

- editing posts and comments;
- viewing profiles.

All required aspects of the protocol have been implemented.

3.2 Software Architecture

We have just summarised the functionality within the application. We now discuss the software architecture used to implement this, outlining some of the technologies that we have used and how the different components of our application interact.

3.2.1 Overview

The application has three distinct components: the frontend, GraphQL server, and federation server. These components are split into multiple separate packages. In this overview, we describe the purpose of each of these components and summarise how they work together. In subsequent sections, we explain this in more detail.

The federation server receives requests from both the GraphQL server and other instances. The frontend communicates only with the GraphQL server. This relationship is outlined in Figure 3.3. The interaction between the frontend and GraphQL server is self-explanatory, however communication between the backend components is more interesting; we discuss this in more detail below.

Many aspects of the system are federated, for example creating and accessing posts. All of these features are provided by the federation server, which implements the protocol described in Appendix H. Servers from other groups can communicate with our federation server. When the GraphQL server receives a federated action from the frontend, it communicates with either the local federation server or a remote federation server belonging to another group. To reiterate, for federated aspects of the system, there is no differential between the local server and remote servers. This allows for a vast amount of code reuse and a generally simpler architecture. Some features, such as creating a community, are not federated. In these cases, the GraphQL server interacts directly with the database to make the necessary changes.

The application is comprised of eight different packages. The dependencies between these packages are shown in Figure 3.2. Complete details regarding individual packages can be found in Appendix E, however Table 3.3 provides a brief description of each.

Table 3.3: Package descriptions.

`@unifed/shared`	This package contains shared code used across all packages. This is primarily validation functions.
`@unifed/frontend`	This package contains the frontend code for the application. It is written with React, using GraphQL for communication with the backend.

`@unifed/backend-core`	This package contains code used across the other backend packages. Mostly, this is the database models, however there are also some helper methods for logging and accessing configurations.
`@unifed/backend-testing`	This package contains code used for testing the backend; it is not used in production. This includes helper functions for generating fake posts, users and communities, as well as database setup and teardown routines.
`@unifed/backend-ml`	This package contains machine learning related code. In particular, it implements the models used for the spam detection filter. It also provides a wrapper around the pre-trained model that we use for classifying the toxicity of messages.
`@unifed/backend-federation-client`	This package is a HTTP client for the federation protocol. It provides a programmatic interface for interacting with federation servers, abstracting away any of the underlying implementation details.
`@unifed/backend-internal-server`	This package implements the GraphQL server, which communicates with the frontend of the application. This is where user authentication is handled.
`@unifed/backend-federation-server`	This package implements the federation server, which conforms to the protocol defined in Appendix H. The GraphQL server and other external servers communicate with the federation server to perform federated actions.

The whole of the application is written using TypeScript. Using the same language across the frontend and backend allowed us to context switch more easily. We were also able to reuse code across the whole application. Using a statically typed language helped us to pre-emptively catch bugs that would otherwise slip through if we were to use a language such as JavaScript. We think this was the correct decision; it allowed us to start very quickly, as we only needed to learn one language for the whole application.

We now provide some further details about the main components of the application.

3.2.2 Frontend

The frontend of the application is written using React. React allows user interfaces and their components to be written in an intuitive, reusable way. Data moves down into components, whilst events bubble up. This leads to declarative and clear code, which is easier to understand and debug.

Our frontend is a fairly standard React application, with some libraries used to communicate with the GraphQL server and manage sessions. Material UI, which is discussed further in Section 3.5, was used to provide most of the components on the interface.

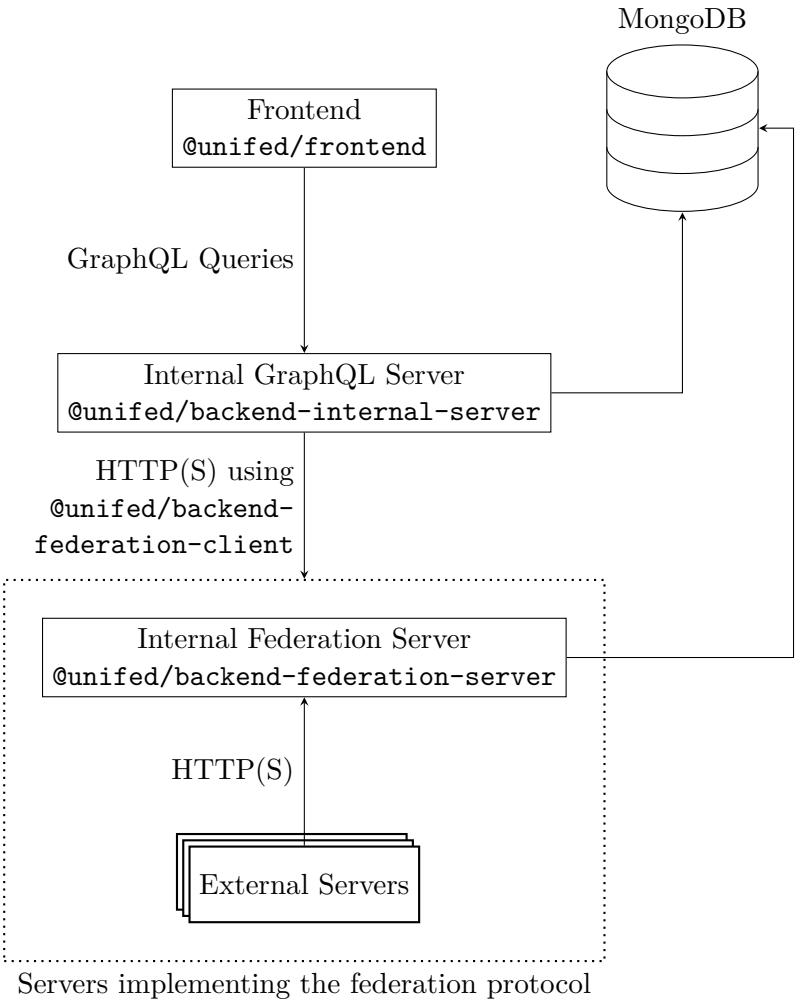


Figure 3.1: Communication between distinct components of the application. Arrows originate from the initiator.

3.2.3 Backend

As mentioned previously, the backend is comprised of two components: the federation server and GraphQL server. In this section, we will broadly discuss the backend design and third-party packages that we have utilised.

We will start by discussing the entity models, which are part of the `@unifed/backend-core` package. Typegoose[1] and TypeGraphQL[2] were used to annotate TypeScript classes. The aforementioned packages then use reflection to generate a MongoDB and GraphQL schema. Having this single source of truth was particularly useful.

[1] https://typegoose.github.io/typegoose/
[2] https://typegraphql.com/

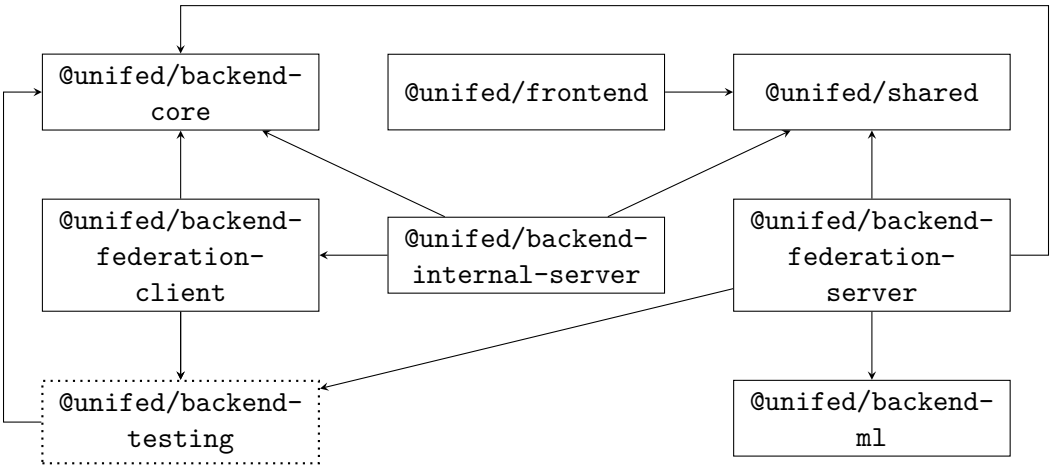


Figure 3.2: Relationships between internal packages in the application. An arrow indicates a relationship from the dependent to the dependency.

GraphQL Server

The GraphQL server handles all requests from the frontend, including user authentication. To do this, we use a library called AccountsJS[3], which provides both frontend and backend libraries to assist with integration. A GraphQL transport layer, written by the maintainers of AccountsJS, was used to integrate AccountsJS with our GraphQL server. After merging the GraphQL schema generated by us and the schema generated by AccountsJS, both can then be accessed through the same GraphQL server.

TypeDI[4] was used for dependency injection. This allowed us to easily replace some dependencies during unit testing, if necessary. Adapting to this coding pattern took some time investment, however it forced a separation of concerns between different modules, which resulted in a cleaner codebase. As mentioned before, this decoupling also made testing simpler.

All federated functionality, such as creating posts, was forwarded to the federation server using the `@unifed/backend-federation-client` package. There were some features, such as creating communities, that were not federated. In these cases, the federation server modified the database directly.

Federation Server

The federation server provided a REST API conforming to the protocol defined in Appendix H. Express[5] was used to provide this web service.

This component utilised the least number of third-party packages and was probably the simplest; it very closely followed the protocol specification.

3.2.4 Infrastructure

Our continuous integration workflow and development tools have already been described in Section 2.2. In this section we discuss the infrastructure and services

[3]https://www.accountsjs.com/
[4]https://github.com/typestack/typedi
[5]https://expressjs.com/

used.

Our application was hosted within a cluster of containers. Specific installation and setup details are provided in Appendix E; here we outline the motivation for containers and how they interact with each other. First, it is important to note that the configuration of these containers was optimised for a development environment. Whilst they would work in production, it would be nonoptimal. We discuss a production deployment setup in Appendix O.

During development, two members of the group used Windows, one member used MacOS and another used Linux. When the School's labs were open, some of us also used the non-root lab machines. So we needed a solution that supported these many platforms. The natural choice was to use containers. To achieve this, we utilised Docker Compose on our personal machines and the Podman equivalent — `play kube` — on the lab machines. We wrote a script to abstract over these different underlying tools — again, this is outlined in Appendix E. Following these instructions, it should be 'plug and play' for the application to run on a machine with Docker/Podman installed. Hereon, we will discuss only Docker, as the Podman equivalent is mostly identical.

Six containers were orchestrated in a single cluster using Docker Compose, shown in Figure 3.3. The MongoDB container hosts the database for the application. For development purposes, a container running Mongo Express[6] allows developers access to the database. An NGINX container proxies requests to the federation server, GraphQL server or frontend container, depending on the URL path. URLs beginning with `/fed` are sent to the federation server, those starting with `/internal` are sent to the GraphQL server, and all others are sent to the frontend container. Whilst the frontend simply consists of static files, serving these through a container allows us to automatically rebuild these assets when the source code changes, which is particularly convenient for development. Finally, it is worth noting that when making internal federation requests, the GraphQL server sends these requests directly to the NGINX proxy, which then re-routes them to the federation server; this is also shown in Figure 3.3.

As mentioned previously, MongoDB was used as our database. The database schema is defined using TypeScript classes, and can therefore be found in the developer documentation: Appendix E. It was convenient to include MongoDB inside the container cluster for development, however in production, this would likely be orchestrated separately in order to capitalise on its scalability advantages.

For further technical details, it is recommended to look at the developer documentation in Appendix E and source code in Appendix M. In particular, we recommend looking at the `@unifed/backend-ml` package's code. A technical evaluation is included in Appendix P.

Finally, it is worth noting that we have made some small open-source contributions throughout the year, which are listed in Appendix N. Primarily, this was fixing dependency declarations in `package.json` files, however there were a couple of more involved contributions too. It is positive to have given back — albeit minimally — to the open-source community, which has made this project possible.

[6] https://github.com/mongo-express/mongo-express

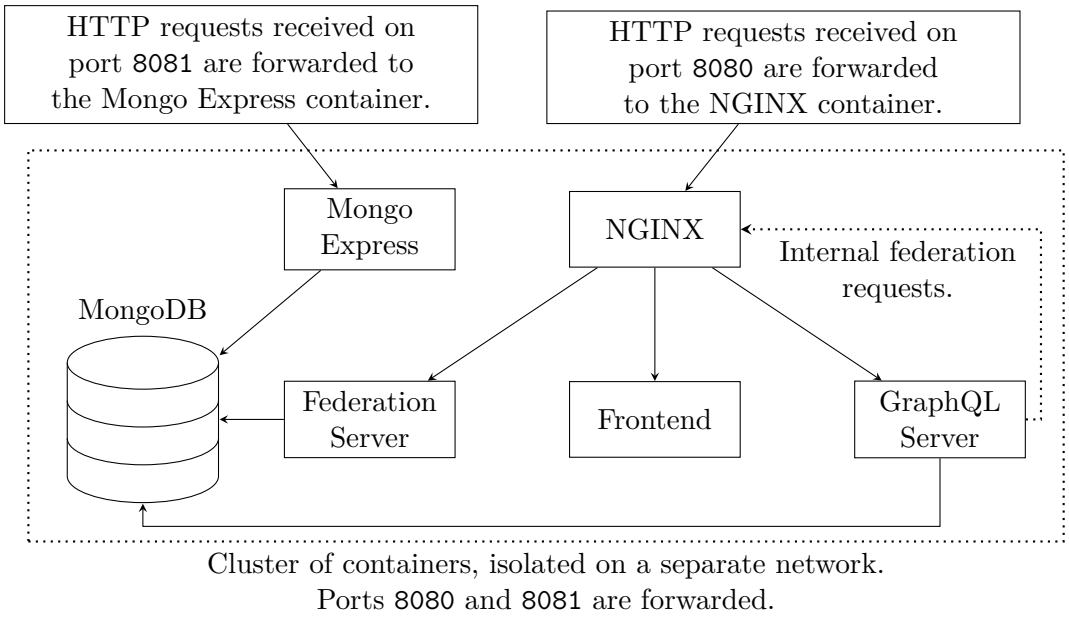


Figure 3.3: Diagram representing the cluster of containers used to host the application.

3.3 Machine Learning Spam Detection**

We have discussed our software architecture and outlined the implementation of our other features. Now we go into some further detail about one of the more complex aspects: a machine learning spam filter.

In this section, we layout the design of our machine learning model, justify the parameter choices and demonstrate that the that it works effectively.

The general process to train our model can be seen below:
There are a couple of important questions to note here:

1. What should we choose n to be?

2. What length should we normalise the messages to?

The first section of this document will focus on these two questions, the second section will look at how we trained our model and the final section will demonstrate that the model is effective.

3.3.1 Input Data

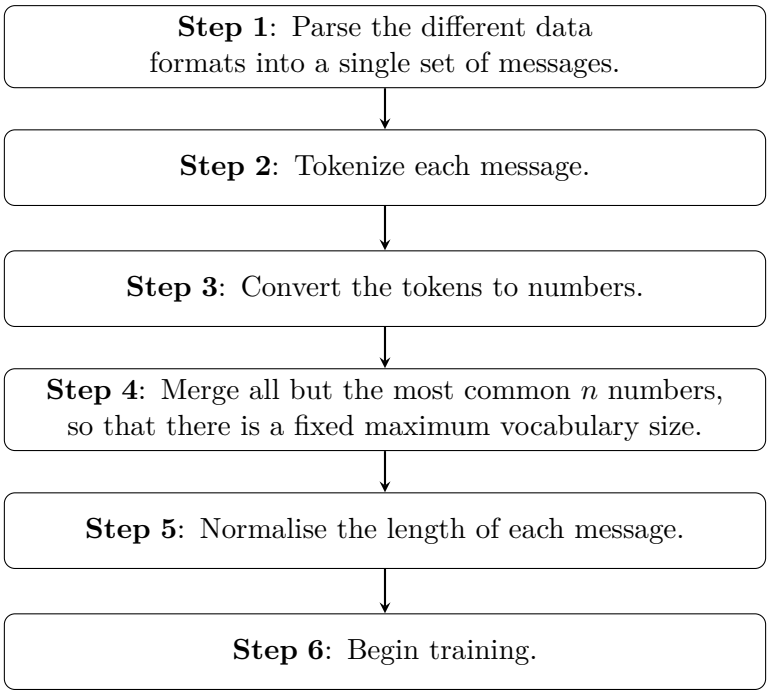
Input data was obtained from the following sources:

1. https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

2. https://www.kaggle.com/veleon/ham-and-spam-dataset

3. http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html

**This section of our report was submitted as an appendix to our third deliverable.



The first is a collection of SMS messages, the second is a set of emails from Spam Assassin and the third is a subset of the Enron Corpus[7]. All three datasets have had their messages categorised as spam and non-spam. In total, there are 42312 messages.

Tokenization

Tokenizing a message turns a string of text into a list of separate smaller strings. When we are splitting up English sentences, the most appropriate place to split tends to be at white spaces. There are some tokens that we deal with differently. For example, '07665443227', '07443448882' and '07665443776' would usually be stored as separate tokens. However, it is more useful to merge numerical values into a single 'number' token. We do the same to URLs.

We then need to turn our list of tokens into a list of numbers. This can be done in numerous ways, however we create a frequency table of all tokens, sort these tokens into a list from most frequent to least frequent and use the index of the token in the list as its numerical equivalent.

Choosing the Vocabulary Size

The first of our questions was how to select the vocabulary size, n. We want to choose n so that the most common tokens are included, but those only occurring a small number of times are not. A larger vocabulary size increases the training time, which is why we must restrict it. To choose the value, it is helpful to see how often each token occurs, which is shown in Figure 3.4a. However, because so many tokens only appear a handful of times, the graph is not very useful. Figure 3.4b solves this by restricting $x \geq 8$ (selected through trial and improvement).

[7] https://en.wikipedia.org/wiki/Enron_Corpus

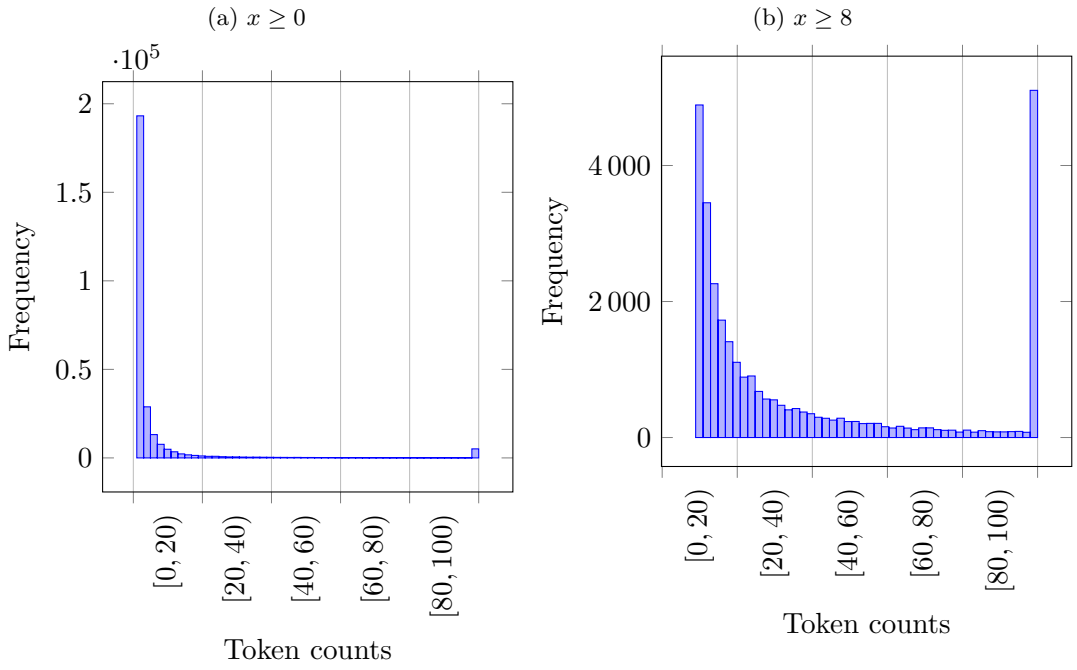


Figure 3.4: Graphs showing how often particular token counts occur.

We can see that in our set of 42312 messages, about 5000 tokens occur 100 or more times. Restricting our vocabulary to words that appear more than 100 times in a dataset of this size is appropriate and therefore we select $n = 5000$.

Choosing the Message Normalisation Length

Our second question was how to choose the length to normalise messages to. If we select a value too large, our matrices will be very big and training will take too long. If the value is too small, important parts of our messages will be truncated. Figure 3.5 shows the frequency of message lengths, which we can use to determine this value.
We can see that most messages contain between 0 and 1000 tokens. Therefore, setting our normalisation length to 1000 is appropriate.

3.3.2 Training the Model

This part of the document discusses how we have trained the model, which can be used to predict whether a message is spam or non-spam.

We have implemented five models: dense[8], dense pooling[9], Twilio dense[10] (name made up by us), long short term memory[11] and bi-directional long short term mem-

[8] https://towardsdatascience.com/nlp-spam-detection-in-sms-text-data-using-deep-learning-b8632db85cc8

[9] https://towardsdatascience.com/nlp-detecting-spam-messages-with-tensorflow-part-ii-77826c8f1abf

[10] https://www.twilio.com/blog/spam-deep-learning-detection-sms-keras-python-twilio

[11] https://towardsdatascience.com/nlp-spam-detection-in-sms-text-data-using-deep-learning-b8632db85cc8

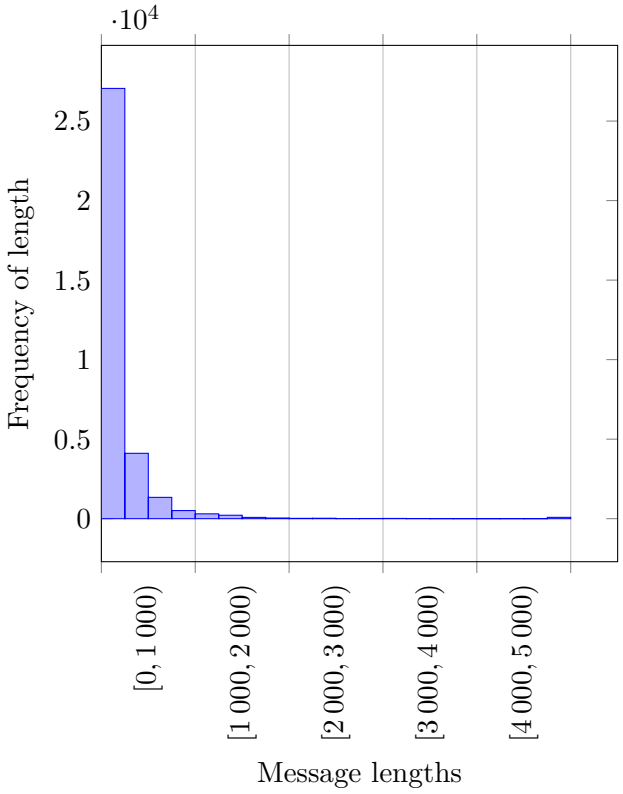


Figure 3.5: Graph showing the frequency of message lengths.

ory[12]. The final two can be eliminated immediately, as after over a day of training we had made almost no progress — they were taking too long. This leaves us with the first three. The derivation of these models and the parameters used within them will be omitted, as we essentially just copied them from the sources.

The input data was split into training data and validation data (80% and 20% respectively). This was then used to fit the model. Training ceased either after the validation loss did not improve for 3 epochs, or after 20 total epochs — whichever occurred sooner. Stopping early prevents the model from becoming over-fitted. Figures 3.6 – 3.8 show the training progress for each model.

The first and last models stopped very early, whereas the second continued for significantly longer. For all three, we can see that both the training accuracy (99.4%, 97.0%, 98.8% respectively) and validation accuracy (96.9%, 95.7%, 97.2% respectively) are very high. We can additionally see that the training accuracy and validation accuracy do not diverge (particularly, the validation accuracy is not significantly higher than the testing accuracy), which indicates that the models have not been over-fitted. For all models, the validation loss eventually becomes larger than the training loss.

All of the models look acceptable at this point, so we will now evaluate each of them with test data and see how they perform.

[12]https://towardsdatascience.com/nlp-spam-detection-in-sms-text-data-using-deep-learning-b8632db85cc8

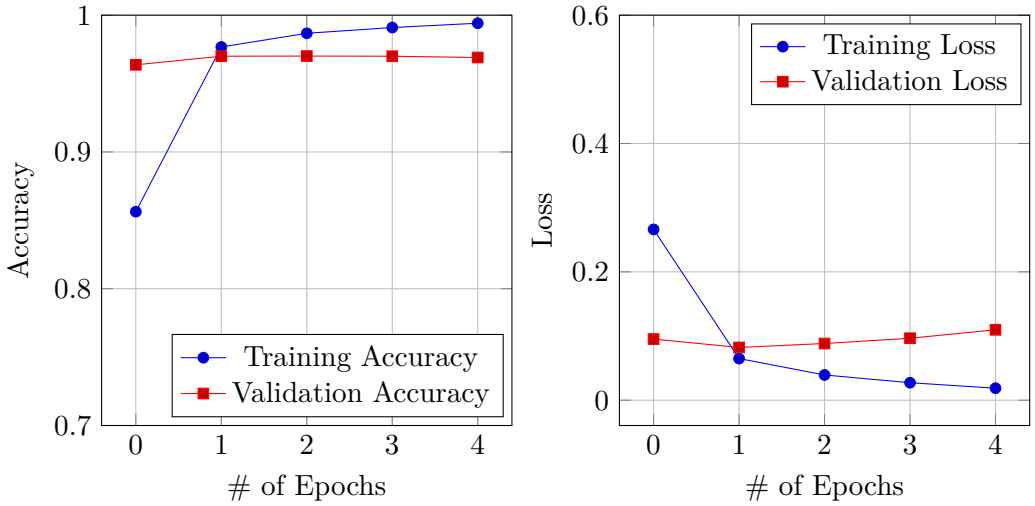


Figure 3.6: Graphs showing training accuracy, training loss, validation accuracy and validation loss for the 'dense' model.

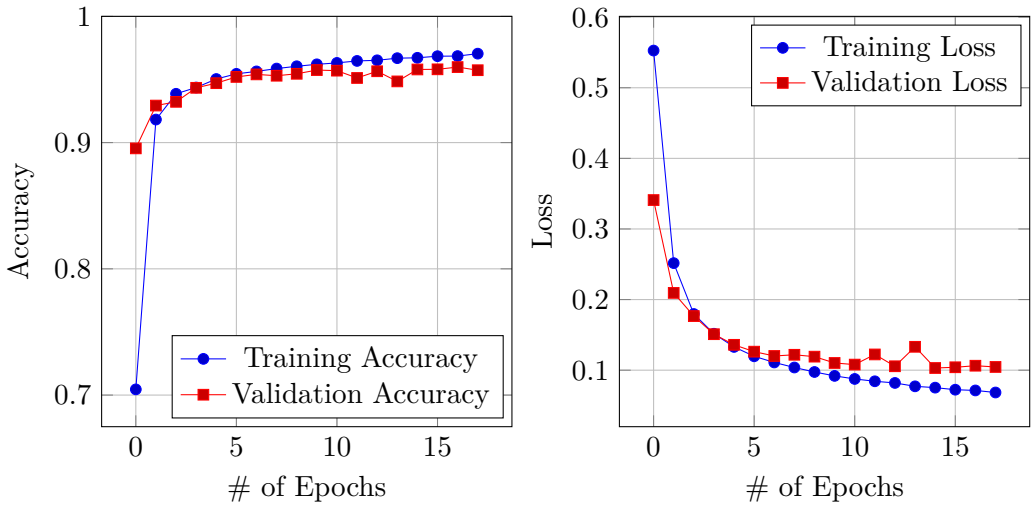


Figure 3.7: Graphs showing training accuracy, training loss, validation accuracy and validation loss for the 'dense pooling' model.

3.3.3 Evaluating the Model

The testing data used is a collection[13] of spam and non-spam posts from a website called the Linguist List[14]. These posts are about "linguistic interests around job postings, research opportunities and software discussion" and have not been used to train the model. These posts are rather subject-specific and therefore differ from the very general messages that have been used for training. This is comparable to how the spam filter will be used within our application: filtering posts made to communities that are domain-specific (e.g. a physics community).

The model predicts how likely a post is to be spam by returning a number in between 0 and 1 (non-spam and spam respectively). Figure 3.9 shows how predic-

[13]https://www.kaggle.com/mandygu/lingspam-dataset
[14]https://linguistlist.org/

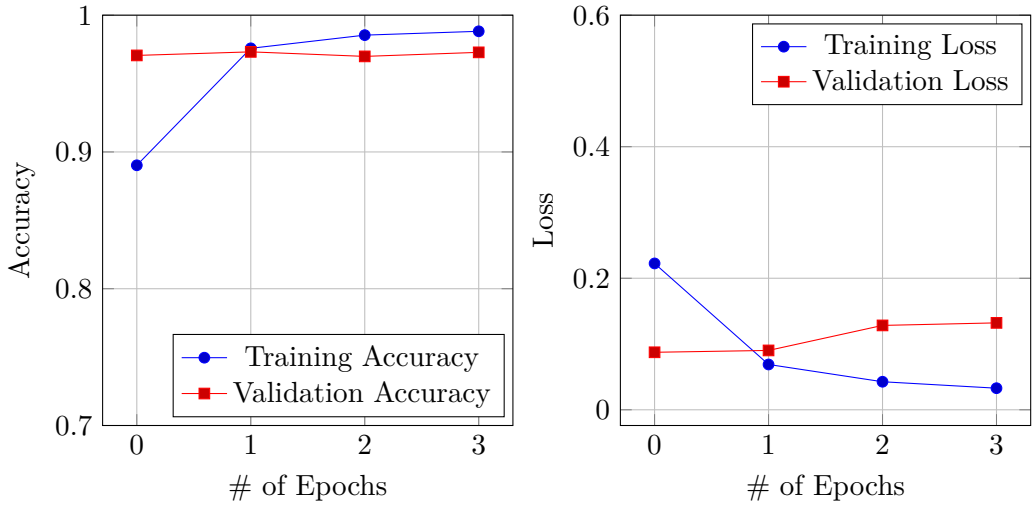


Figure 3.8: Graphs showing training accuracy, training loss, validation accuracy and validation loss for the 'Twilio dense' model.

tions deviated from their actual values. Smaller deviations are better. The largest possible deviation is 1, which indicates that the prediction was completely wrong.

We can see that almost all spam posts are detected correctly, with only a very small number detected as non-spam. A slightly larger number of non-spam posts are incorrectly detected as spam. If we consider that a deviation of ≤ 0.2 indicates certainly that a post is spam/non-spam, then we get the results found in Table 3.4.

Model	Spam / %	Non-spam / %
Dense	83.2	72.6
Dense pooling	94.0	71.3
Twilio dense	93.6	74.2

Table 3.4: Percentage of predictions with a deviation ≤ 0.2.

Considering both spam and non-spam posts, the Twilio dense model is generally the most effective. We will therefore use this in our application to detect spam content.

3.3.4 Conclusion

Developing a spam detection model could be a project in itself — we could certainly take this further. There are possibly more effective models (such as those mentioned earlier, which we did not have the resources to train), which could improve our results. Using a larger, possibly more diverse set of data could increase the accuracy of the models. Particularly, the types of spam posts used for training tended to be the very obvious "congratulations, you've won £1000", rather than more subtle forms of spam advertising. However, the extent to which we have taken this will be

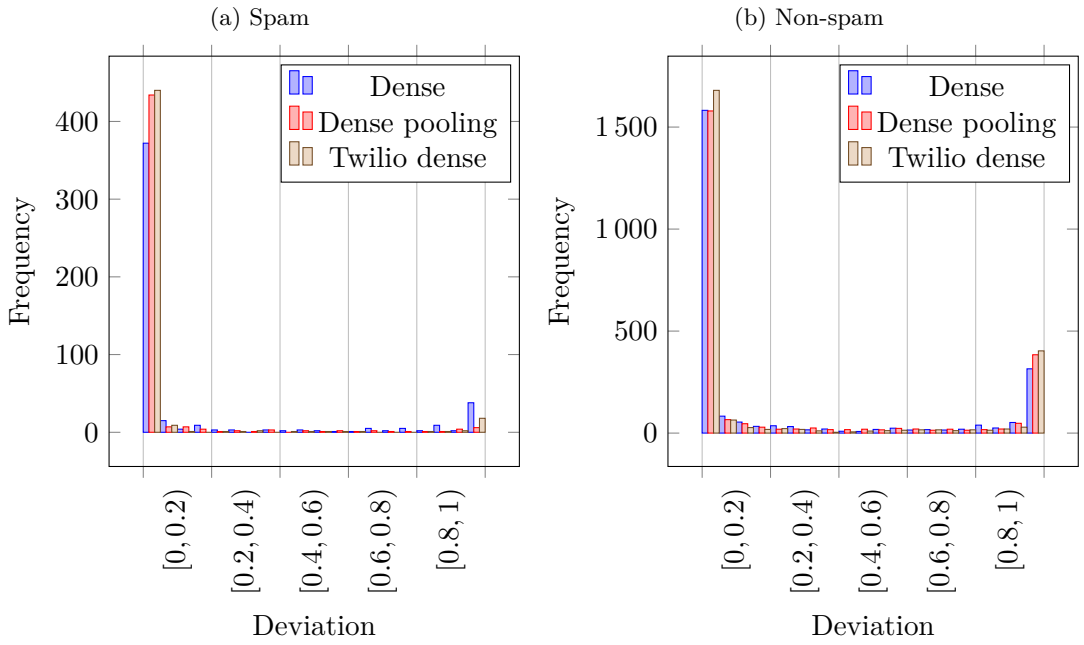


Figure 3.9: Graphs showing deviation of predictions.

sufficient for our project and we are confident from our evaluation that it will work effectively.

3.4 Group Video Chat

We now discuss another of our other more technically complex features: group video chat. To facilitate this feature, we have utilised WebRTC[15], which significantly simplified the implementation. This is a peer-to-peer technology, with the negotiation performed through a central server (known as the signalling server). No more detail regarding the workings of WebRTC are necessary to understand how our group video calling was implemented.

Media capture devices are concepts within the browser that allow access to components such as cameras and microphones, as well as virtual devices, for example screen recorders. We request access to the user's camera and microphone — they will receive a prompt asking them to allow this by their browser. We can then send these streams over peer-to-peer connections, which is explained below.

Upon entering the video chat room, each user subscribes to a WebSocket, which allows messages to be received from the server. It also sends a `request` message to the server, which is broadcast to everybody subscribed to the WebSocket. At this point, all users who received the `request` message initiate a peer-to-peer connection with the new user by sending an `offer` message. The joining user responds to the `offer` with an `answer`. At this point, some negotiation occurs through the server between the two users. After successful negotiation, a peer-to-peer connection has been established and the exchange of video and voice data commences. This process is shown in Figure 3.10. The video and voice data does not travel through our server,

[15]https://webrtc.org/

which makes implementation less complicated and also reduces server load.

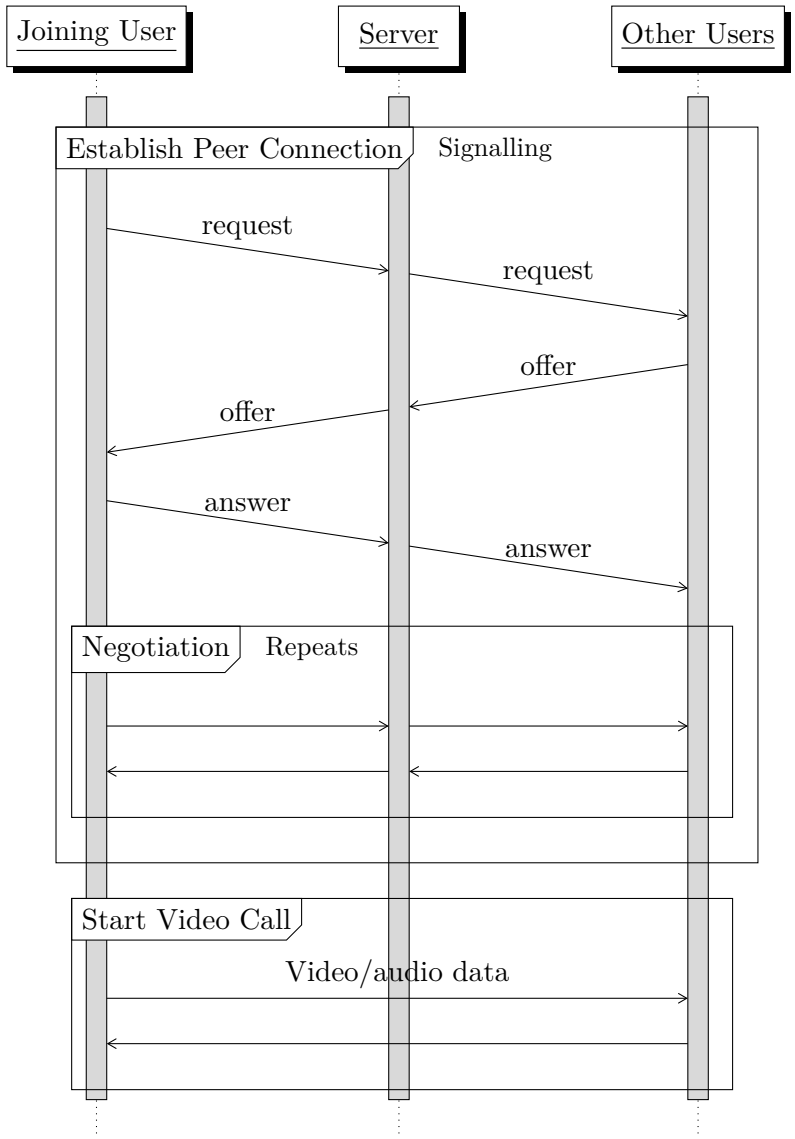


Figure 3.10: Sequence diagram showing messages sent between the joining user, the server, and other users already in the call.

Part of the negotiation process involves using a STUN server to obtain information such as the user's external IP address. These servers are hosted for free by multiple companies; we are using one from Google. In some rare instances, obtaining information with the STUN server may fail, making a peer-to-peer connection impossible. In these cases, a TURN relay server can be set up as a fallback. TURN servers are expensive to host and so are not available for free; we have not set up our own, as it was not essential. A TURN server would be a useful addition to make in the future.

3.5 Accessibility[†]

Having discussed the technical aspects of the project, we now outline some of our accessibility considerations. We begin by discussing some general user interface principles, then outline adaptations we have made for three types of user: mobile users, colour-blind users, and screen reader users.

3.5.1 User Interface

Creating a high-quality user interface maintains the user's engagement and enhances their experience. We have followed established user interface design principles in order to achieve this.

Shneiderman's Golden Rules

Shneiderman's "eight golden rules of interface design" (Shneiderman & Plaisant, 2010, p. 66) are intended to improve usability and establish a well-designed user interface; we go through each of these next.

Strive for consistency We utilised Material UI, a library of user interface components conforming to the Material Design[16] specification. This provided a consistent look and feel across the whole of our application. We also used this library to compose our own more complex components, which were reused across the application.

Enable frequent users to use shortcuts Our user interface can be navigated fully using the keyboard (discussed further in Section 3.5.4), however the implementation of shortcuts is limited.

The text editors in use throughout the application support standard keyboard shortcuts, such as those used for highlighting text with the arrow keys, copying, pasting, undoing, and redoing. Our markdown editor also contains shortcuts for multiple formatting options, including italics, emboldening, underline, strike-through, and bullet points. Improvements could be made here with keyboard shortcuts for some of the other features on the user interface.

Offer informative feedback Throughout the application, users are shown informative warnings and error messages. These are human-readable and understandable.

Titles are displayed on many pages to indicate to the user where in the application they are.

Design dialogue to yield closure When users perform actions, they are notified of the outcome. For example, when creating a post, the user is taken to the created post's page on success. Other actions within the application use similar methods to give closure to the user.

[†]A small amount of this section was included in our first and third deliverable.

[16]https://material.io/

Offer simple error handling When they occur, user-friendly errors are displayed. Within forms, the particular field causing the error is highlighted in red, with the error displayed underneath it. This makes it particularly easy for users to rectify problems.

Permit easy reversal of actions Most actions on the interface can be reversed if the user wishes. For example, posts can be deleted or edited after being created. This principal in particular would benefit from more work; for example, it is currently not possible to delete a community.

Support internal locus of control This somewhat cryptic rule essentially means giving users a sense of command over the application; they should be the initiators of actions and have overall control.

Users of our application are certainly in control; there are no surprises and nothing is carried out without the user's acknowledgement.

Reduce short-term memory load This rule can be rephrased as 'recognition over recall'; it is essentially an attempt to reduce the amount the user must remember. An obvious example of this is having buttons to navigate the user interface, rather than needing to access URLs directly.

Galitz's Principles

The general principles presented in Galitz, 2007 also aim to produce high quality, user-friendly interfaces. These principles are extensive and cover a wide range of considerations. One of the most important concepts is that of 'trade-offs' (Galitz, 2007, p. 58):

> Design guidelines often cover a great deal of territory and often conflict with one another or with technical requirements. In such conflicts the designer must weigh the alternatives and reach a decision based on trade-offs concerning accuracy, time, cost, and ease of use. Making these trade-offs intelligently requires a thorough understanding of the user and all design considerations. The ultimate solution will be a blend of experimental data, good judgement, and the important user needs.

Recognising that it may not always be possible to conform to these principles allows flexibility in the user interface's design. Whilst it is desirable to follow the principles as closely as possible, there are instances, such as the examples given, where trade-offs must be made.

The remaining 24 principles are comprehensive and so we do not go through each individually, however we discuss *accessibility* more in the next sections. Other principles — such as *clarity*, *familiarity*, *simplicity*, *consistency* and being *aesthetically pleasing* — are achieved in part due to the well designed Material UI component library that we have utilised; the Material Design look and feel is familiar to users.

Configurability is achieved in some areas, such as allowing users to change the interface theme; those who prefer a darker theme can use one. The *flexibility* principle is demonstrated in our markdown editor, which allows novices to write markdown

using a simple editor, whilst simultaneously allowing advanced users to write TEX formulas.

Other principles, such as *forgiveness*, *control*, *responsiveness* and *recovery* overlap with Shneiderman's rules and so have previously been discussed.

3.5.2 Mobile Users

We designed our user interface to work seamlessly on mobile devices; it is fully responsive, with all features accessible on all screen sizes. A mobile-friendly user interface allows the application to be used conveniently from places where larger devices may not be available, such as when travelling. This portability will be expected by our users, who will be familiar with other social networking sites that support this. Standard methods using CSS media queries were used to conditionally change the layout on various screen sizes. Testing on mobiles is discussed in Appendix F.

3.5.3 Colour-blind Users

Our user interface is fully accessible for colour-blind users. By law, public sector websites must meet the WCAG 2.1[17] standard. As we were targeting a university community, it was important we took steps to meet these guidelines.

We used a Google Chrome extension called WAVE[18] to evaluate how colour-blind accessible our application was. Part of this tool highlights areas that have poor colour contrast, which would cause difficulties for a colour-blind user. Where this was the case, we modified the colours to resolve any issues. We discuss accessibility testing further in Appendix F.

3.5.4 Screen Reader Users

For the same reasons described in the previous section, it was important that we took steps to make our user interface accessible to those who use screen readers.

This was a multipart process; full testing details are described in Appendix F. ARIA[19] labels were added to forms and elements, allowing screen readers to accurately tell users what is being displayed. The interface was also tested to ensure it was fully operable via the keyboard. As before, we used the WAVE Chrome Extension to detect any accessibility issues during development and testing.

Our use of Material UI helped significantly with making our user interface accessible, as most components already conformed to accessibility guidelines. Further work could be done here, in particular, testing with an individual who regularly uses a screen reader.

3.6 Security

Security has been an important consideration in our project. Our first deliverable contained the following user story: "as a student/member of staff, I want my information to be kept secure". The following considerations have been made to ensure that our application is secure:

[17] https://www.w3.org/TR/WCAG21/

[18] https://chrome.google.com/webstore/detail/wave-evaluation-tool/jbbplnpkjmmeebjpijfedlgcdilocofh

[19] https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

- Using HTTPs to communicate with clients (encryption in transit).
- Using HTTPs to communicate with other federated instances where possible (encryption in transit).
- We designed and proposed the security aspect of the federation protocol, to ensure that instances can communicate securely.
- Hashing passwords in our database using a modern hashing algorithm (bcrypt).
- Sanitising and validating user input.
- Protection against NoSQL attacks[20].

There are further details about testing the security of our application in Appendix F.

3.6.1 Dependencies

Dependencies are often a weak point in applications. With such deep dependency trees, it is impossible to manually track updates and security advisories. We used GitHub's Dependabot, which gives notifications about security advisories for dependencies. During the project, we received only one advisory; the notification can be seen in Figure 3.11. This was actually a dependency for Jekyll, the software we have used for our user manual, so it was not within the actual application. Figure 3.12 shows the information available after viewing the security advisory. Clearly this is an invaluable tool for keeping the application secure.

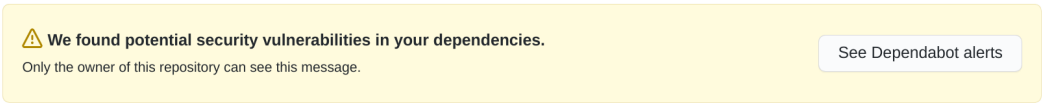


Figure 3.11: Security advisory notification from Dependabot.

In addition to the above, we updated our dependencies once a week[21] in case repository owners had not issued a security advisory for bugs.

[20] https://www.netsparker.com/blog/web-security/what-is-nosql-injection/

[21] Dependabot does have an option to do this automatically, however it does not support the version of Yarn we were using.

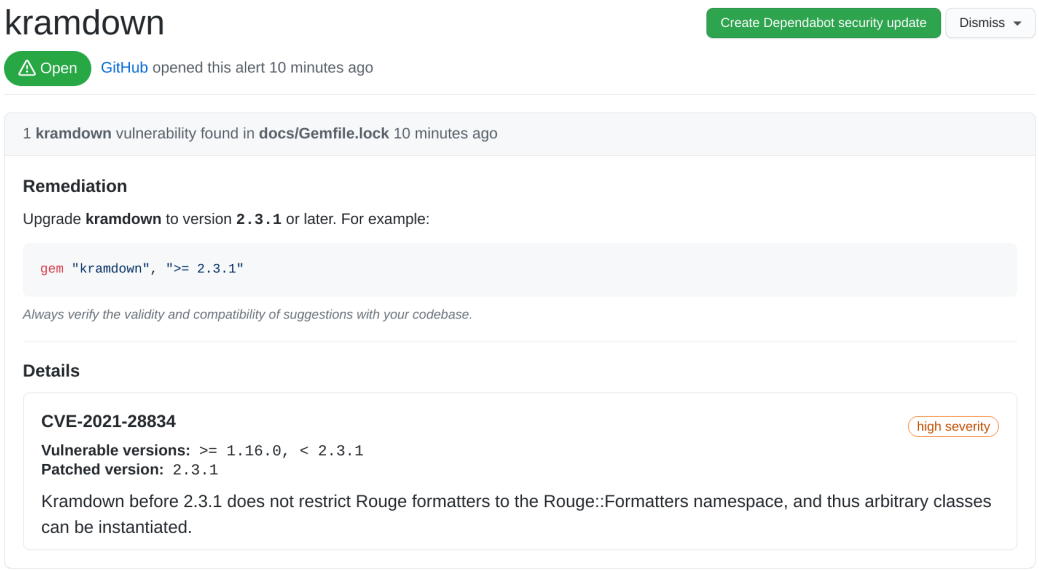


Figure 3.12: GitHub security advisory for a dependency.

4. Evaluation and Critical Appraisal

We have now described the project as a whole, detailing both its organisational and technical aspects. In this part of our report, we evaluate our application, comparing it to both the original objectives, specification, and other similar systems.

This evaluation focuses on appraising the application and discussing whether the requirements were met. Analysis and improvements of our project management and supergroup interactions have been discussed in Section 2.1.2 and Section 2.3.1 respectively. Technical improvements are omitted from here and instead outlined in Appendix I and Appendix P.

4.1 Original Objectives

In Section 1, we outlined the motivations for the project. Broadly, these were to (i) provide a federated application in an attempt to solve some of the problems with centralised moderation and (ii) overcome the social gap in universities left by the move to online learning.

For the first, we have certainly implemented a federated application. However, it is difficult to predict whether within a university this would prevent the issues that come with centralised moderation. It is easy to envisage the existing power hierarchy[1] simply being reflected in the moderation structure. Of course, this is not the same as large technology companies making moderation decisions that potentially affect billions of users. However, it emphasises an important point: much of the success in a federated network comes from giving users enough diverse choice. If users can not — without prejudice — simply leave a server to join another, a centralised system is likely to form around whichever server becomes established soonest. Mastodon — another federated social network — has over one hundred servers advertised[2]; with no known major moderation issues, this emphasises the important role that user choice plays in success for an application of this nature. Within a university, it is difficult to foresee whether moderation issues would be resolved with federation; it needs to be tested and analysed on a large scale within such an environment. However, evidence from these other federated social networks suggest that it is possible in certain circumstances.

The second objective can be appraised with more certainty. It is hard, probably impossible, to completely overcome the void left by in-person socialising. Organic, by chance, interactions are frequent in real life: bumping into people. The closure of

[1] By power hierarchy, we have in mind the central university, schools, the students' union etc.
[2] https://joinmastodon.org/communities

common spaces has minimised most of this and made remaining social interactions far more arranged. In particular, we aimed to address this. We wished to develop a very open platform, reflecting the sorts of public spaces that are currently unavailable: common rooms, computer laboratories, classrooms, pubs, cafes etc. For this reason, we did not focus on features such as *private* communities or *private* messaging, but instead collaborative aspects such as *group* video calling. Allowing users to drop into a 'room' to see, hear and chat with whoever else may be in there is comparable to the spontaneous interactions we are currently missing. With all this in mind, we believe that we have gone some way to overcoming the social gap currently experienced by students.

4.2 Specification

Having discussed the project objectives, we now take a more detailed look at the original specification. We were provided with the following set of core requirements:

- ability to register users;
- creating new posts;
- creating comments on posts;
- a simple web-based user interface and
- an implementation of a protocol for propagating content between instances.

These were all prioritised and completed by the minimum viable product; it was important to focus on these first, as they were the only mandatory requirements that we were given. Using a web-based user interface, users were able to register an account, create posts and create comments. Registration was enhanced with email verification, as part of our duty to keep accurate contact details for our users. Having completed these features early on in the project, they have now been fully tested and we are confident that they work as intended. Therefore we consider this part of the specification successfully met.

In our first deliverable, we produced a set of over one hundred user stories — the product backlog — which constitutes the specification in a project using the Scrum model. Some of these have evidently been completed — such as creating or editing a post — and so there is little of interest to discuss; it would be impractical to go through and appraise each and every one of them. In Table 4.1 we focus on user stories for which there is something of significance to be presented. The full set of requirements can be found in Appendix J, separated by the sprint in which they were completed.

Table 4.1: Appraisal of implemented user stories.

User Story	Appraisal
As a community owner, I want users to have verified emails so that I know they are less likely to be bots or spam accounts.	On registration, users were sent a verification email to ensure that the email address that they had provided was correct. This is important because it ensures we have accurate contact information for the user. Based on this, the requirement has been met. However, an aspect missing from this is the ability for the user to request the email confirmation to be re-sent, in unlikely event that they do not receive it. As this is a minor issue, which most users will not need, we do not think it impacts the success of this particular user story. However, it should be noted for future work.
As a student/member of staff, I want to be able to change my name and password while logged in.	Users are able to change their display name when logged in, however they can only change their password through the 'reset password' functionality. Therefore we consider this requirement partially met. Whilst the functionality for both exists, the flow for a user to change their password is convoluted and is not acceptable from a user experience perspective.
As a student/member of staff, I want to be able to report offensive content to the community administrator(s).	This requirement has been successfully met for communities hosted on our own server. However, the federation protocol does not have these capabilities and so it does not work for communities hosted elsewhere. Therefore we consider this requirement partially met.

Continuation of Table 4.1.

User Story	Appraisal
As a student/member of staff, I want to be able to access the application from my mobile phone, so that I can use the application when travelling. As a student/member of staff with colour blindness, I want the GUI to be colour-blind friendly. As a student/member of staff with a visual impairment, I want the GUI to be screen-reader friendly.	These three user stories relate to accessibility requirements for our user interface. The application is fully mobile accessible; the user interface is responsive on all screen sizes. As such this requirement has been successfully met. The same can be said for the implementation of a colour-blind friendly user interface. We also implemented features to make the user interface accessible to a user with a visual impairment. The interface is fully navigable using a keyboard and we have implemented ARIA attributes to assist screen readers. However, much more testing — preferably with somebody who often uses a screen reader — is required before we can claim with confidence that our application is accessible for those who are visually impaired. Based on this, we consider this requirement partially met. These aspects have been discussed further in Section 3.5.

End of Table 4.1.

In Table 4.2 we appraise the non-functional requirements that were submitted as part of our first deliverable.

Table 4.2: Appraisal of non-functional requirements.

Requirment	Appraisal
The GUI should be quick to load, preferably take no longer than 100 milliseconds to perform any action.	Many actions — such as creating posts on other servers — require external communication; response times are in part determined by the performance of these other servers. As a result, analysing this requirement is slightly more difficult. We restrict the scope to only internal actions. The initial load of the user interface takes over a second, which clearly fails the requirement. However, the user interface does not reload during navigation and so subsequent actions are fast, taking less than 100 milliseconds. Therefore we consider this requirement partially met.
The GUI should be consistent such that the application feels homogeneous to the user. The GUI should communicate that a user's action has been received, for example, darken a button when it has been pressed.	These requirements are considered together as they are both satisfied by our use of the Material UI library. Material UI provides a consistent look and feel across the entire application. This ensures that the application feels homogeneous to the user. When buttons are pressed, they darken. When actions are performed, loaders are displayed. This feedback indicates to the user that their action has been received. This requirement has therefore been met.
The GUI should be visually appealing. For example suitable colours and logos can be used to make it interesting to look at.	The user interface uses visually appealing colours, conforming to a colour scheme that we have chosen. In addition to this, we have designed a logo[3] to give our application a professional, branded look. This requirement has therefore been met.
The GUI should be simple so that the user is not overloaded with information and options.	The user interface has a small number of options and requires no configuration by the user. A user manual is available to explain all of the features. Tooltips appear over buttons to provide additional guidance regarding the action they perform. Therefore we consider this requirement met.

[3]Derived from a logo found on https://hatchful.shopify.com/.

Continuation of Table 4.2.

Requirment	Appraisal
The font used in the GUI should be easy to read.	The user interface uses Roboto, a font used across many Google and Android products. This font is clear and will be familiar to users. Font colours have been carefully selected such that light font colours are on dark backgrounds and dark font colours are on light backgrounds. This requirement has therefore been met.
Text/images should be of a size so that the user does not have to strain their eyes to read it. Text/images should not be so large that it dominates the screen.	The default font sizes of Material UI are used; headers, sub-headers, paragraphs etc. all have pre-configured sizes, which are appropriate for our application. The only image used on the application is the logo, which is appropriately sized in the top left of the user's screen. We consider this requirement met.
Error messages should be clear and non-technical.	Error messages and warnings have been written with clarity and simplicity in mind. They are non-technical and give the user a clear explanation about what has gone wrong. We consider this requirement a success.
Actions, particularly destructive actions, should have an undo button if at all possible.	Actions such as creating posts can be undone. However, some actions, such as deleting posts cannot. This is a limitation of the federation protocol. We consider this requirement partially met.
As a student/member of staff, I want my information to be kept secure.	We have outlined security considerations in Section 3.6. Therefore we consider that this requirement has been met.
As a student/member of staff, I want my privacy to respected, for example not selling my information.	We have written a privacy notice, available in Appendix L. This outlines how we use our users information and demonstrates our commitment to upholding the seven GDPR principles. Therefore we consider that this requirement has been met.

End of Table 4.2.

The plan presented in our third deliverable outlined what we could focus on for the remainder of the project: a high-quality user interface, security, moderation, direct messaging, up-votes and down-votes, community suggestions, and federation compatibility. However, in the feedback for that deliverable we were advised to

be "dissuaded [from] increasing the amount of functionality in [the] system" and to instead "focus on documentation and testing". As such, our plans changed. In Table 4.3 we discuss a broad selection of the user stories that we did not implement. The primary reason for most of these not being implemented was time; we started off with a list of over 100 user stories and had to prioritise the most important ones. As such, if the only reason for a user story not being implemented was a time limitation — with no particularly interesting reason for why it was not prioritised — we do not include it below.

Table 4.3: Discussion of non-implemented user stories.

User Story	Comments
As a student/member of staff, I want to be able to like and dislike posts. As a student/member of staff, I want to be able to undo a like or dislike on a post. As a student/member of staff, I want to be able to like and dislike comments. As a student/member of staff, I want to be able to undo a like or dislike on a comment. As a student/member of staff, I want to be able to see the number of likes and dislikes a post has. As a student/member of staff, I want to be able to see the number of likes and dislikes a comment has. As a student, I want to be able to react to content using emojis, so that I can indicate a further range of feelings rather than just 'like' and 'dislike'.	All of these user stories relate to up-votes, down-votes and 'reacts'. These probably moved up and down our product backlog the most; we changed our minds on whether we wanted to implement it quite a few times. We all agreed that it would be a positive feature, however we were not convinced it would be worth it with the limited time we had. The federated aspect of this was quite complicated and we were concerned that it could end up being a time sink. With the up-vote/down-vote proposal only ratified into the protocol within the last month of the project, we decided that there was not enough time for us to be able to implement, test, and document the feature to the standards we would like. There was a risk that we would reach the project deadline — having spent a significant amount of time on the feature — not in a position to confidently merge it into our master branch. With this in mind, we decided our time would be better spent on improving already implemented features.

Continuation of Table 4.3.

User Story	Comments
As a student/member of staff, I want to make posts consisting of file attachments, such as images or PDFs. As a student/member of staff, I want to post polls that anybody can see the responses to. As a student/member of staff, I want to post polls that other people can add response options to. As a member of staff, I want to be able to post polls to which only I can see the responses.	Whilst these features might have been desirable to have, we had to consider how they would help us achieve the core objectives discussed earlier. There are numerous websites for hosting files — these can just be shared on our platform as links. The same argument applies to polls; there are a vast number of high quality services that already provide this. Based on these considerations, these user stories were not prioritised and therefore not implemented.
As a community administrator, I want to be able to delete my community.	We could not decide upon how to implement this user story. Deleting a whole community is an incredibly destructive action; users may have spent lots of time contributing. We needed to balance the wishes of an administrator wanting to delete their community with the desires of the users. We considered a 'close down' period, where users were notified that the community would be deleted in 30 days and have time to migrate. Alternatively, we considered archiving communities rather than deleting them, such that the content is always available. In the end, we decided that it was not important enough to spend a long-time deciding over and so deprioritised this user story.

Continuation of Table 4.3.

User Story	Comments
As a community administrator, I want the option for my community to be 'closed', such that only people who are invited can join. As a community administrator, I want to review and accept/decline requests from students/members of staff that want to join my community. As a community administrator, I want to requests to join my community to be able to include custom fields, such as answering a question or agreeing to some rules. As a student/member of staff, I want to be able to have private conversations with a group of users. As a student/member of staff, I want to be able to send private messages to other users. As a student/member of staff, I want to be able to receive private messages from other users. As a student/member of staff, I want to be able to report offensive private messages to the community administrator(s). As a community administrator, I want to be able to read private messages that have been reported by a user.	We were not keen on implementing private spaces, such as closed communities or direct messages. All of these user stories went against one of our fundamental objectives, which was to create an open platform, reflecting the aspects of life currently missing: classrooms, cafes, pubs etc. These were the few features where we actively decided not to implement them, even give sufficient time. When considering features, it is important to determine whether they align with the core philosophy of the application; this was one of the few examples where they did not. This demonstrates how the project objectives changed somewhat since we wrote the initial product backlog.

Continuation of Table 4.3.

User Story	Comments
As a student/member of staff, I want to be able to see which communities a particular user is subscribed to. As a student/member of staff, I want to be able to see the position of a particular user, such as their year of study or position within a department. As a community administrator, I want to be able to add a 'flare' to a user, which is displayed next to their name so that it easy to identify special users in the community. As a student, I want to be able to post anonymously to some communities so that communities such as 'St Fessdrews'[4] and 'St Crushdrews'[5] on Facebook can be recreated. As a community administrator, I want to be able to pin important posts so that they remain at the top of community's posts page.	Due to the federation protocol not supporting the features required, these user stories were not implemented. It was the case that sometimes we needed agreement from the whole supergroup before implementing a feature. In some instances, like the above, these features were not the priority of other groups and so did not make it into the protocol. This was to be expected in some instances.

[4]https://www.facebook.com/stFessdrews
[5]https://www.facebook.com/StACrushes

Continuation of Table 4.3.

User Story	Comments
As a community administrator, I want to set rules such that users with fewer than n posts to my community must be approved by me, to try and reduce spam on the community.	This particular feature was not implemented due to a lack of time. However, we had some interesting ideas for this. We were keen on using a rule-based language to allow administrators total control over the users that were allowed to post. Rules could be based on account age, posting frequency, the number of spam posts detected from the user etc. In our research, we came across the idea of using a language such as Prolog to achieve this; we encountered some examples of this being used in development tools for constraining dependency versions between multiple packages. Whilst not being used for the same purpose, we saw how this could be adapted for constraining the users allowed to post to a community. User interface considerations would need to be made; we would require some preset rules for administrators uncomfortable with the total freedom provided by a feature such as this.

End of Table 4.3.

4.3 Similar Systems

In the previous section, we looked at how our original specification compared with the application that we developed. We now look at existing systems that offer similar services and see how our application contrasts. We start with a feature comparison in Table 4.4, then analyse the similarities and gaps. These platforms were selected due to being frequently mentioned in our group and supergroup meetings.

Table 4.4: Comparison of features with other similar systems.

Description	**Ours**	Facebook	Reddit	Diaspora	Mastodon
Federated	Yes	No	No	Yes	Yes
Account registration	Yes	Yes	Yes	Yes	Yes
The platform is centred around communities (or the equivalent)	Yes	No	Yes	No	No
The platform is centred around individual users	No	Yes	No	Yes	Yes
Creation of posts	Yes	Yes	Yes	Yes	Yes
Creation of comments	Yes	Yes	Yes	Yes	Yes
Editing of posts and comments	Yes	Yes	Yes	No	No
Deletion of posts and comments	Yes	Yes	Yes	Yes	Yes
Rich text posts	Yes	Yes	Yes	No	No
Typeset mathematics	Yes	No	No	No	No
Syntax highlighting	Yes	No	No	No	No
Deeply nested comment replies	Yes	No	Yes	Yes	Yes
Up-votes and down-votes	No	No	Yes	No	No
Emoji reacts	No	Yes	No	No	No
Content recommendations	No	Yes	Yes	Yes	Yes
Profile pages	Yes	Yes	Yes	Yes	Yes
Content search	No	Yes	Yes	No	No
Community search	Yes	Yes	Yes	N/A	N/A
User search	No	Yes	Yes	?	Partial
Spam filter	Yes	Yes	Yes	?	?
Video calling	Yes	Yes	No	No	No

Continuation of Table 4.4.

Description	**Ours**	Facebook	Reddit	Diaspora	Mastodon
Follow communities	Yes	Yes	Yes	N/A	N/A
Follow users	No	Yes	Yes	Yes	Yes
Private/closed communities	No	Yes	Yes	N/A	N/A
Accessible user interface	Yes	Yes	Yes	?	Yes
Dark mode theme	Yes	Yes	Yes	No	Yes
Direct messaging	No	Yes	Yes	Yes	Yes
Account deletion	No	Yes	Yes	Yes	Yes
Username change	No	Yes	Yes	Yes	No
Polls	No	Yes	Yes	Yes	Yes
Blocking other users	No	Yes	Yes	Yes	Yes
Data download/migration	No	Yes	No	?	Yes
Moderation tools	Yes	Yes	Yes	Yes	Yes
Mobile compatible	Yes	Yes	Yes	Yes	Yes
Content reporting	Yes	Yes	Yes	Yes	Yes

End of Table 4.4.

We can see that up-votes and down-votes (or 'reacts') are a key feature that our application is missing; these were available on all other platforms that we compared with. This leads to another key feature that we are missing: content recommendation. Some form of voting is an essential precursor to content recommendations, as we need to be able to rank the popularity or sentiment of a particular piece of content within the application.

We are also lacking an account deletion function, which other platforms have. Whilst this is not essential for GDPR compliance (we can delete accounts manually), it gives users a sense of control over their data. Closely related to this is allowing users to download their data. Facebook and Mastodon allowed this, whilst Reddit does not. It is not essential for GDPR compliance, as data can be provided to users on request. However, as before, a feature such as this provides users a sense of control over their data.

All other platforms had the ability for users to block other users. This is a useful feature, however we would need to consider how it would work with features such as the drop-in group video calling.

Following on from this, group video calling was a feature only available with Facebook and us. However, there are some functional differences: our group video calling is 'drop-in', whereas Facebook's is pre-arranged between particular friends. Both are useful for different purposes; ours would not be useful for holding a private conversation, for example.

In summary, it is clear that our application is not as feature-rich as the alternatives; we are missing some important aspects that users may expect. However, we stand out in particular areas, such as typesetting mathematics and video calling, which most other platforms did not have. This is encouraging but shows that some future work may be needed to make the application production-ready.

5. Conclusion

We have discussed our project management approach, explored technical details and evaluated the application itself. To conclude, we summarise the project and its aims, discussing some future possible directions of work.

In November 2011, Diaspora — a federated social media platform — had 216 000 users (Bielenberg et al., 2012, p. 1). By April 2021, this had grown to over 654 300 (Federation, 2021). The threefold increase over the past decade provides encouragement for the adoption of federated social media networks, although this is clearly dwarfed by the number of users on conventional platforms. The obvious explanation is the lack of mainstream attention given to these applications; they are often open-source and therefore lack the large advertising budgets other sites may enjoy. Dismissing their relative unpopularity as simply a marketing failure, however, is probably simplistic.

Large social networking sites — such as Facebook and Twitter — invest significant amounts in keeping users engaged, primarily through deeply personalised recommendation of content (Guy et al., 2010, p. 1). Trienes et al. note: "as with traditional social media, one key success factor of [a federated] network is an active and engaged community" following with "[recommender systems] are needed to make distributed social media attractive to large user groups as well as competitive to centralized networks" (Trienes et al., 2018, p. 1). We agree with this analysis and believe this should be a central focus of any future work. This is covered further in Dhekane and Vibber, 2011.

From a pedagogical perspective, the purpose of this project was to obtain "a broad overview of software engineering", in particular the "background and practical experience ... to enter professional careers where [we] will be working on large-scale software projects in teams"[1]. The process of self-organising ourselves, planning, and executing the project has given us all invaluable experience, both technically and interpersonally. The skills we have gained will, in many cases, be directly transferable to the workplace.

Appendix J contains user stories for extensions that are yet to be implemented; this is an obvious source of future work on the application. In particular, a focus on administration and moderation tools should be made, as these are currently sparse.

We have implemented an extensive social networking application, complete with a wide range of features. This includes significant extensions, such as group video calling, machine learning spam detection, and accessibility considerations. At the core of the application is the concept of federation, allowing communication within a network of other servers conforming to the same protocol. We have specified,

[1] https://www.st-andrews.ac.uk/subjects/modules/catalogue/?code=CS3099&academic_year=2020/1

planned, designed, implemented, tested and documented the application, presenting all of these aspects within this report. We take away much from this collaborative project and value the experience. Whilst there is further work to do, we are pleased with the application that we have developed.

Bibliography

Benington, H. D. (1987). Production of large computer programs, In *Proceedings of the 9th international conference on software engineering*.

Bielenberg, A., Helm, L., Gentilucci, A., Stefanescu, D., & Zhang, H. (2012). The growth of diaspora-a decentralized online social network in the wild, In *2012 proceedings ieee infocom workshops*. IEEE.

Boehm, B., & Hansen, W. J. (2000). *Spiral development: Experience, principles, and refinements* (tech. rep.). Carnegie Mellon University Software Engineering Institute.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61–72.

Definition of social network. (2020). In *Lexico.com*. Oxford University Press. Retrieved March 21, 2021, from https://www.lexico.com/definition/social_network

Dhekane, R., & Vibber, B. (2011). Talash: Friend finding in federated social networks, In *Ldow*.

Federation, T. (2021). *Diaspora**. Retrieved April 12, 2021, from https://the-federation.info/diaspora

Galitz, W. (2007). *The essential guide to user interface design: An introduction to gui design principles and techniques*. Wiley. https://books.google.co.uk/books?id=Q3Xp%5C_Awu49sC

Goodhart, C. A. (1984). Problems of monetary management: The UK experience. In *Monetary theory and practice* (pp. 91–121). Springer.

Guy, I., Zwerdling, N., Ronen, I., Carmel, D., & Uziel, E. (2010). Social media recommendation based on people and tags, In *Proceedings of the 33rd international acm sigir conference on research and development in information retrieval*.

Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques, In *Proceedings of the 9th international conference on software engineering*.

Schwaber, K., & Sutherland, J. (2017). *The Definitive Guide to Scrum: The Rules of the Game*. Retrieved February 16, 2021, from https://web.archive.org/web/20200902181000/https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

Shirky, C. (2011). The political power of social media: Technology, the public sphere, and political change. *Foreign affairs*, 28–41.

Shneiderman, B., & Plaisant, C. (2010). *Designing the user interface: Strategies for effective human-computer interaction*. Addison-Wesley. https://books.google.co.uk/books?id=pddxRQAACAAJ

Trienes, J., Cano, A. T., & Hiemstra, D. (2018). Recommending users: Whom to follow on federated social networks. *arXiv preprint arXiv:1811.09292*.

A. Interim Reports

Our interim reports have been included alongside this deliverable in the `interim-reports` directory.

B. Group Meeting Minutes

29/09/2020 — Team

- Kian Cross has put forward a basic protocol to take to the supergroup.
- We think ActivityPub is too generic and it will take too long to implement.
- Kian Cross will be the rep for this week's supergroup meeting, others can attend and listen in.
- Lewis Mazzei will take minutes from the team/supergroup meetings.
- Future team meetings will be recorded.

30/09/2020 — Supervisor

- Be ahead of the curve, easier if supergroup makes decisions aligned with us.
- Consolidate what federation means with supergroup.
- Get supergroup to request Q&As with Edwin.
- Use user stories to drive the requirements — gives the project a narrative.
- Start sprints now, probably involving user stories.
- Build deadlines in now, pull 2/3 weeks off the end for writing up the report.
- Could potentially assign a project manager.
- TypeScript recommended over JavaScript; supervisor thinks types really help.
- Time management is one of the biggest pitfalls, make a calendar.
- By next week, we should know a significant amount more about what we're doing.

05/10/2020 — Team

- Delegation of report sections.
  - Tech — Kian Cross, Iain Dunbar *(No longer part of group)*.
  - Scrum & Inter-Group Communication — Lewis Mazzei.

  - Big Picture & User Stories/Requirements/Backlog — Robert Mardall, Allan Mathew Chacko.

- Report drafted by Sunday 11th, ready to finalise in Monday's meeting.
- Will draft in a shared Word doc, move into LaTeX before submission.

07/10/2020 — Supervisor

- Regarding the initial set of user stories we discussed:
  - Clarify communities/sub-communities.
  - Define what a 'message' is.
  - Add behaviour for messages (such as reading messages).
  - Define lecturers and modules.
  - Make clear that polls are a special kind of post.
  - Define 'live chat'.
  - Add user stories for 'live chat' (i.e. reading/sending messages).
  - Change PDF user story to wiki user story.
  - 'Pre-recorded lectures' — what are they? Where will they be hosted?
  - What does 'find people from other instance mean'? Search? Find by things in common? People online now? People who are members?
  - Clarify the hierarchy of users.
  - Data dictionary to define the conceptual terms in the system.
  - What happens with data if somebody deletes an instance?

15/10/2020 — Supervisor

- Discussed architecture again, Al drew us a helpful diagram — saved in the Team's channel.

19/10/2020 — Sprint 1 Planning

- Kian Cross set up some scaffolding/boilerplate code in the GitHub repository and walked everyone through the set-up.
- Decided to use Express instead of Fastify, because we couldn't get GraphQL to work with Fastify and TypeScript.
- Now using `eslint` instead of `gts` to check the code style because `gts` wasn't very good.
- Added backlog to projects tab on GitHub.
- Picked three user stories to work on this week, split them up into three tasks (as issues) and assigned each member of the team to all three (we are working on them together). We decided it made sense to split work between the front and backend so created sub-teams:

  - Frontend - Robert Mardall, Lewis Mazzei, Allan Mathew Chacko.
  - Backend - Kian Cross, Iain Dunbar *(No longer part of group)*.

- Identified the main technologies needed to implement these tasks, links to resources posted in main channel.

22/10/2020 — Sprint 1 'Daily' Scrum

- People are very busy with W07 deadlines, scope of sprint is smaller than expected.
- Kian Cross will try get a rough implementation of the backend working today, so that the frontend can speak to it, if not Iain *(No longer part of group)* will try.
- Will hopefully have a basic functioning register/login form working by Monday, no bells and whistles such as email verification/password resetting though yet.

26/10/2020 — Sprint 2 Planning

Progress Report

- **Robert Mardall** — Created basic login and register pages. UI is looking good.
- **Allan Mathew Chacko** — Read up on and experimented with the technologies we are intending to use. Will do further work on Wednesday after other module deadlines.
- **Lewis Mazzei** — Looked through React documentation, sourced libraries for forms and general UI (Formik and MaterialUI) and also looked through documentation for these. Had a chat with Robert Mardall regarding his basic form implementation for signing up/logging in. Didn't have much time to write code due to a conflicting deadline.
- **Kian Cross** — Got some of the basic working for sign-up/login. Now need to hook it up to the GraphQL interface. Will do more work on it after other module deadlines on Wednesday.

Minutes

- Lewis Mazzei will remain Scrum Master this week
- Postponed Sprint 2 planning to Thursday since tasks from Sprint 1 are still to be completed. Will account for things taking slightly longer than we expected in further sprints. Small delays caused due to absence of Iain Dunbar *(No longer part of group)*.
- Emailed Al to cancel Wednesday's meeting.
- Selected the 'Create Posts' stories to be added to sprint

29/10/2020 — Sprint 2 'Daily' Scrum

Progress Report / Minutes

- Allan Mathew Chacko has been learning about the tech we are using. He plans to be able to contribute within a couple of days.
- Kian Cross has been working on backend and has almost got a functioning login system.
- Others have been busy with other work.
- Discussed/solved some technical issues.

02/11/2020 — Sprint 3 Planning

- Set a deadline of Friday to implement code so that we have time to fix bugs and write report before final deadline.
- Delegated tasks regarding remaining functionality for MVP:
  - **Allan Mathew Chacko** — Profile and account settings frontend.
  - **Lewis Mazzei** — Login/register and verify/reset frontend.
  - **Robert Mardall** — Posts frontend.
  - **Kian Cross** — Agreed on the paths we will be using and put them in a shared document. Finishing off backend endpoints that people need for frontend.

Progress Report

- **Robert Mardall** — Started implementing posts. Created post page, implemented post editor and loaded posts onto homepage.
- **Lewis Mazzei** — Implemented the API calls required to hook the login/register system up to the backend.
- **Kian Cross** — Continued to work on the backend. Posts are done. Now working on federation.
- **Allan Mathew Chacko** — Started implementing account and account settings pages.

05/11/2020 — Sprint 3 'Daily' Scrum

- Good progress being made on the remaining functionality.
- A few technical blockers resolved.
- Decided to push back our supervisor demo from the 11th to the 18th
- Decided to have optional daily meetings from now until the deadline (13th November).

09/11/2020 — Sprint 4 Planning

Progress Report

- **Robert Mardall** — Finished implementing post functionality. Started testing. Added commenting functionality
- **Allan Mathew Chacko** — Finished implementing account/settings pages. Started testing.
- **Lewis Mazzei** — Implemented forgot/reset password system, hooked it up to the backend and integrated Material UI with the form components. Started looking at testing.
- **Kian Cross** — Continued to work on the backend. Implemented tests for the backend. Implemented federation functionality. Added test code coverage.

Minutes

- Have most of the implementation sorted, on track for deadline.
- Delegated sections of the report, recorded this delegation in the lead authors section of the 2nd deliverable report.

12th November — Sprint 4 'Daily' Scrum

- Proofread report and tested the application.

11/01/2021 — Sprint 5 Planning

- Kian caught the team up on some changes made since the last deliverable:
  - Upgraded the Yarn package manager to version 2, which uses a new dependency resolution method called Plug'n'Play (PnP). This has made our builds faster and more importantly, the automated testing runs in about one-third of the time. We do not need to worry so much about running out of GitHub Action minutes anymore.
  - We are now using a code-coverage service called Codecov, which automatically comments the results from the automated tests onto the pull request. It also provides browsable coverage reports. This makes it very easy to see if a pull-request has written tests to cover all of the code it has added/changed.
  - We are no longer using Cypress for testing — instead we will use traditional unit tests. Cypress was very fragile and was giving artificially high test coverage. Additionally, tests took about 15 minutes to run, which was using a lot of our GitHub Action minutes.
  - The backend has been split up into multiple decoupled packages.
  - Multiple bug fixes.
  - Fixed some dependency problems.

  - Added additional testing infrastructure, such as an in-memory MongoDB instance for testing backend code.

- Kian walked the team through the backend code so that others can start working on this part of the code-base too.
- Picked and assigned tasks for Sprint 5.

18/01/2021 — Sprint 6 Planning

Progress Report

- **Kian**
  - Added unit tests for `backend-internal-server`.
  - Added unit tests for `backend-federation-server`.
  - Added some example tests for the frontend to make it easier for others to add more.
  - Written a check to compare the GraphQL queries written for the frontend with the backend schema. This means that it is no longer possible for invalid queries to make their way into the code base, as it is explicitly tested.
  - Implemented dependency service injection for `backend-internal-server` to make testing easier.
  - Isolated the container networks to ensure that the code that we write does not reply on them being on the same network.
  - Use an alias ('this') to reference the local instance, instead of `localhost`. This means that the federation server can run on a different network.
  - Added an installable development database to make development easier. At the moment, this just contains a default user.
  - Participated in the supergroup meeting and submitted some protocol proposals to the supergroup.
- **Lewis/Robert/Allan** — Not had time to do much. Tasks moved to Sprint 6.

Minutes

- Picked and assigned tasks for Sprint 6.

21/01/2021 — Sprint 6 'Daily' Catch-up

Progress Report

- **Kian** — Working on the editing posts backend, deleting posts backend and spam detection machine learning. Participated in the supergroup meeting.
- **Robert & Lewis** — Split the editing and deleting posts frontend between them. They had some problems accessing the interface, but these are resolved now.

- **Robert** — Working on error page.
- **Allan** — Working on improving account settings page — good progress so far.

Minutes

- Kian explained mock testing functions.
- Discussed long-term plans for the project.

25/01/2021 — Sprint 7 Planning

Progress Report

- **Kian** — Got a prototype of the machine learning spam detection. Completed the backend for editing/deleting posts. Now working on adding new aspects from the federation protocol.
- **Lewis** — Not had time to do anything. Tasks carried over to the next sprint.
- **Robert** — Completed the error and exception pages. Now moving onto editing comments and posts.
- **Allan** — Nested comments are implemented and working. Also added a drop-down menu to the header to tidy up the additional options we have added. The account settings page is completed (apart from unit tests).

Minutes

- Scheduled calls at 10am everyday to meet (optionally) if anybody has blockers.
- Selected and assigned new tasks for Sprint 7.

28/01/2021 — Sprint 7 'Daily' Catch-up

Progress Report

- **Kian** — Working on the spam detection machine learning as we have got access to a lab GPU client. Participated in the supergroup meeting and submitted some protocol proposals.
- **Lewis** — Working on the 'Create Community' button — not yet completed.
- **Robert** — Working on deleting and editing post/comments. Taken over deleting from Lewis. Code is messy — needs tidying. Options to delete/edit should only display when it is the user's own post (or an administrator).
- **Allan** — Not much done. Will get something pushed before Monday.

01/02/2021 — Sprint 8 Planning

Progress Report

- **Kian** — Finished the spam detection machine learning.
- **Lewis** — Working on the 'Create Community' button — not yet completed — waiting on the backend call.
- **Robert** — Finished editing and deleting posts/comments. Now cleaning up the code and adding unit tests.
- **Allan** — Working on community subscriptions. Account settings is essentially done — just need to add tests. Tried separating things into separate branches but needs help.

Minutes

- Selected and assigned new tasks for Sprint 8.

04/02/2021 — Sprint 8 'Daily' Catch-up

- **Kian** — Made a small change to the database. On track to complete tasks this week. Participated in supergroup meeting and submitted some protocol proposals.
- **Lewis** — Not had a chance to work on anything this week — busy with lectures.
- **Robert** — Waiting to merge the edit and delete posts branch into master — need the backend to be updated first. Investigated the colour-blind accessible GUI task — already seems very accessible, but there might be a few minor things to tweak.
- **Allan** — Worked on mobile-friendly GUI and frontend tests.

08/02/2021 — Sprint 9 Planning

Progress Report

- **Kian** — Added backend call to create a community. Currently working on upgrading the federation protocol and adding administrators to the GraphQL communities call.
- **Lewis** — Needs to hook the frontend for creating a community up to the backend and start unit tests.
- **Robert** — Confirmed everything is colour-blind friendly. Working on a dark-mode theme.
- **Allan** — Still needs to finish community subscriptions.

Minutes

- No new tasks assigned — carrying over incomplete tasks from the previous sprint.

11/02/2021 — Sprint 9 'Daily' Catch-up

- **Kian** — Improving some parts on the backend that were rushed. Participated in the supergroup meeting and submitted some protocol proposals.
- **Lewis** — Still working on the frontend to create a community.
- **Robert** — Implementing dark-mode.
- **Allan** — Subscribe/unsubscribe almost done. Now doing the home page to display subscriptions.

15/02/2021 — Sprint 10 Planning

Progress Report

- **Kian** — Written a global user context to clean up state management on the frontend. Helped Lewis with the 'Create Community' frontend.
- **Lewis** — Almost finished the 'Create Community' frontend, with some help from Kian. One final test to write.
- **Robert** — Working on dark-mode theme. Needs help styling the post-editor — Kian to help.
- **Allan** — Nearly finished community subscriptions. Problem with one of the GraphQL calls — Kian to fix.

Minutes

- Robert is going to focus on adding missing tests this sprint.
- Lewis is going to finish the 'Create Community' branch and try to finish incomplete tasks from previous sprints.
- Kian is doing all the remaining issues added to the 'Deliverable 3' milestone on GitHub.

18/02/2021 — Sprint 10 'Daily' Catch-up

- **Kian** — Working on federation upgrade, report, and video calling. Participated in the supergroup meetings.
- **Lewis** — Not done anything this week — has fallen behind with other things. Will try to get the 'Create Community' frontend done before the next deliverable.

- **Robert** — Done dark mode. Made GUI partly mobile friendly. Working on the report.
- **Allan** — Done account settings. Going to finish the community subscriptions today.

22/02/2021 — Sprint 11 Planning

Progress Report

- **Kian** — Implemented group video calling and spam detection integration. Working on the report and other remaining issues.
- **Lewis** — Completed the 'Create Community' feature.
- **Robert** — Fixed quite a few little things. Mainly working on the report now. Writing frontend tests too.
- **Allan** — Focusing on networking practical — not had time to do much.

Minutes

- Demo with Al on Friday at 2pm.
- There is a soft deadline of 5pm today on report completion — gives us time to proofread.
- Need to find a time where we can all test video calling in preparation for the demo.

15/03/2021 — Sprint 12 Planning

- Had a break after the last deadline in order to do work for other modules. Time to get started again on this!
- Tasks assigned for this sprint.
- Two more weeks of programming, then two weeks for the final report.

18/03/2021 — Sprint 12 'Daily' Catch-up

Progress Report

- **Kian** — Sorted out the documentation for API and end-users. Working on completing the pull request for federation.
- **Lewis** — Almost completed the password strength bar.
- **Robert** — Planning and research. Working on the frontend for upvotes/downvotes.
- **Allan** — Working on the frontend for administration and reporting.

Minutes

- Monday meeting moved to Tuesday.

22/03/2021 — Sprint 13 Planning

Progress Report

- **Kian** — Completed the federation PR. Working on administration and documentation.
- **Lewis** — Continued to work on the password strength and visibility features. Should hopefully get them merged today or tomorrow. Will get to work on the account/community deletion stuff this week.
- **Robert** — Frontend for upvotes/downvotes done — waiting for backend. Working on documentation in the meantime.
- **Allan** — Working on the frontend for administration and reporting.

25/03/2021 — Sprint 13 'Daily' Catch-up

Progress Report

- **Kian** — Working on administration. Also writing report.
- **Lewis** — Continued to work on the password strength and visibility features. Writing unit tests.
- **Robert** — Started user manual.
- **Allan** — Carrying on with moderation.

Minutes

- Monday meeting moved to Tuesday.

29/03/2021 — Sprint 14 Planning

Progress Report

- **Kian** — Fixing bugs with subscriptions and creating posts/comments. Working on report.
- **Lewis** — Password visibility done.
- **Robert** — Writing the user manual. Done most of it.
- **Allan** — Working on frontend for administration and reporting.

Minutes

- Finish programming within the next 3 days (hard deadline).
- Rigorous testing and fix any bugs (2 days after that).
- Document our code extensively (2 days after that).

02/04/2021 — Sprint 14 'Daily' Catch-up

Progress Report

- **Kian** — Still fixing bugs. Working on documentation and report.
- **Lewis** — Working on networking practical. Finished the password strength bar PR. and visibility features. Writing unit tests.
- **Robert** — Fixed some frontend issues (reply button for comments, moved user icon inside card). Working on user documentation and will finish the admin privileges and likes/dislikes once the backend is in place.
- **Allan** — Working on the 'Action Button'. Replacing existing stuff with it. Found a bug with nested comments — will make an issue.

Minutes

- Someone to do research on similar systems for report: Facebook, Reddit, Diaspora, Mastodon.

05/04/2021

Progress Report

- **Kian** — Working on documentation and report.
- **Lewis** — Working on networking practical.
- **Robert** — Admin stuff, small issues.
- **Allan** — Done the report button and doing action button. Still doing admin stuff for moderation.

Minutes

- No longer doing sprints, as we are in the 'finishing up' stage.

08/04/2021

Progress Report

- **Kian** — Working on documentation and report.
- **Lewis** — Working on networking practical.

- **Robert** — Random frontend issues. Also writing developer documentation for frontend.
- **Allan** — Working on networking practical.

08/04/2021

Progress Report

- **Kian** — Working on documentation and report.
- **Lewis** — Working on networking practical.
- **Robert** — Completed frontend documentation.
- **Allan** — Finished moderation queue.

C. Project Plan

This plan has been written retrospectively and amalgamates aspects from the plan submitted in our first deliverable with points from our minutes. Throughout the year plans have changed; we have not reflectively updated this document. Therefore, this is an authentic account of the plan that we started the project with. This was one of the required appendices from the student handbook.

Our plan for the first semester is to produce a prototype with the following functionality:

- ability to create and delete an account;
- ability to create, edit and remove posts consisting of text and links;
- ability to make, edit and remove comments on posts;
- a web-based GUI to use our application;
- some basic moderation tools;
- the ability to communicate with other federated instances.

We intend to split the work on the project as follows:

- **Frontend** — Lewis Mazzei, Robert Mardall, Allan Mathew Chacko.
- **Backend** — Kian Cross, Iain Dunbar[1].

To begin development, we will set up some boilerplate code and infrastructure. This will include aspects such as setting up continuous integration and organising the file structure of the repository. Once this has been completed, we will begin implementing features from the backlog.

At each milestone, such as the prototype, we will carry out some alpha testing to ensure the application is working as intended. This will involve the team using the application for a short amount of time to identify any bugs or issues that went unnoticed during development. We must leave enough time before deliverable deadlines to fix any problems that are found.

[1]Iain subsequently left the group just after the first deliverable. These teams remained the same, however Allan assisted with the backend when necessary.

Schedule

Sprints will be one week in duration, with the first commencing on the 19th October. This will allow four distinct sprints to take place before the minimum viable product is due, with the last sprint review occurring on the 9th November. This final four-day sprint will take place between the 9th and 12th November; it will focus on the documentation related to the handin report.

Sprint review, retrospective and planning will be combined into a weekly 11 am meeting on Mondays, with 'daily' scrum meetings occurring on Thursdays at 10 am to keep track of progress. Additional communication will take place as and when needed via messages. The team has been — and will continue to — communicate between each other and our supervisor through Microsoft Teams. Notes have been and will continue to be taken at each event.

This takes us to the end of Semester 1. We will follow a similar plan for Semester 2, adjusted when necessary and where lessons have been learnt.

D. User Manual

The user manual is available at the following URL:

- https://kiancross.github.io/unifed/

In the unlikely event that this URL is inaccessible at the time of reading, an offline version has been included with this report in the `user-manual` directory. Open the `index.html` file in a browser to view.

E. Developer Documentation

The developer documentation is available at the following URL:

- https://kiancross.github.io/unifed/developers/

In the unlikely event that this URL is inaccessible at the time of reading, an offline version has been included with this report in the `developer-documentation` directory. Open the `index.html` file in a browser to view.

Please note that this documentation contains some links to the source code, which is hosted on a private GitHub repository. Access to the repository can be provided quickly on request, however it is certainly not essential.

F. Testing[†]

Testing has been discussed throughout the main document. In this appendix, we provide some more detailed information, as per the student handbook.

Automated Testing

We have written approximately 300 automated tests, which are run on each pull request. Pull requests cannot be merged into the master branch until they have passed all of the tests. Additionally, a testing coverage report is added to each pull request, so that we can ensure new code has over 85% test coverage. Subsequently, our master branch always contains a working version of our application. Test coverage of the repository is 84%. Table F.1 contains a per package breakdown of the test coverage percentage.

Package	Test Coverage / %
`@unifed/shared`	100
`@unifed/frontend`	81.65
`@unifed/backend-core`	90.59
`@unifed/backend-ml`	93.24
`@unifed/backend-federation-client`	88.79
`@unifed/backend-internal-server`	72.49
`@unifed/backend-federation-server`	90.58

Table F.1: Test coverage percentage of packages.

Frontend

The frontend is also tested using unit tests. In places where AJAX requests are made, we use a library to mock the responses, so we do not need to be running an instance of our backend. We are using GraphQL to allow the frontend and backend to communicate. This allows us to use a tool to validate that all of the queries made

[†]We have submitted parts of this appendix in our previous deliverables.

by the frontend conform to the schema implemented by the backend. This check is run on each pull request and prevents interface bugs between the frontend and backend from making their way into our code. Critically, this covers a weak point in many applications' testing: the API communication between components.

Backend

The backend is tested using unit/integration tests. In places where the database is accessed, we spin up an in-memory instance of MongoDB. This allows us to test our application as it would behave in a production environment, giving us further confidence that it is functioning correctly.

In places where outbound HTTP requests are made, we use a library to mock the response. Using these techniques we have been able to test all of our backend code using automated tests. This gives us high confidence that the backend code is working as intended.

Limitations

There are, however, some limitations with our automated testing. It is hard to test all code branches, for example, there are sanity checks that should be impossible to reach. The frontend code would be close to 100% were it not for group video calling. Video calling proved extremely difficult to automatically test, due to it being very network-based. As such, the testing coverage for this feature is rather low, which pulls down the average. This is the same reason for why the `@unifed/backend-internal-server` package is relatively low — the backend code for video calling was very difficult to test too. To compensate for this, the video calling feature has been tested very thoroughly manually. The backend machine learning package also had some CLI code which was hard to automatically test; again, this brought down the average for this package. Bearing all this in mind, we consider the testing coverage percentage we have achieved to be adequate and a good indication of a well-tested codebase.

Federation

We have automated acceptance tests to validate the public interface of the federation protocol, ensuring that it conforms to the specification. In addition to this, we have tested the federation capabilities with other groups and fixed bugs as and when we have encountered them.

Finally — as noted in the main text of this report — we treat ourselves as a federated instance (just a special-case local version) and so local testing is actually testing the federated code paths too.

Accessibility

We tested how mobile-friendly our application was by using the toggle device tool in the developer tools section of Google Chrome. This toggle makes the page fit to a mobile-sized screen. We also tested a variety of other sizes, such as tablets. During alpha testing, discussed later, we also tested the user interface on a physical mobile device and can confirm that it worked as intended.

The WAVE accessibility Google Chrome extension is used for evaluating colour-blind and screen-reader accessibility aspects. We have discussed this in the main body of this report. We are highly confident that the user interface is colour-blind friendly and moderately confident that it is screen-reader friendly; we would like to test with a regular screen-reader user.

Security

Despite our best efforts, it is possible that security issues will have made their way into our application. In order to find these, we have performed a basic penetration test on our application. The penetration test looked for the following common exploits:

- Cross-site scripting attacks.
- NoSQL injections.
- Path traversal attacks.
- Evaluation of our third-party libraries and dependencies for exploits.

All our dependencies have been upgraded to the latest version. We were unable to find any issues. However, before going ahead with a production deployment, we would — if possible — employ the expertise of a penetration testing company to evaluate the security aspects of our application. Such an audit will also provide reassurance to users of our application, who are about these sort of security considerations.

Alpha Testing

We have performed manual testing to ensure that our application behaves as expected from the point of view of an actual user. Alpha testing has been performed throughout the year, however it became more rigorous within the last two weeks of the project. We use the feature descriptions and test cases, provided in Appendix K, to perform a manual test of the application. Any problems found are resolved. Whenever possible, we write a unit test for bugs we fix in order to prevent them from regressing in the future.

Conclusion

Based on the above, we are confident that our application is functioning as we have intended it to. We have a robust mechanism in place for preventing regressions, which allows us to continuously make steps forward. During the continuous integration flow, we perform static analysis on the code to preemptively find any problems; this provides further confidence that the application is working correctly.

G. Supergroup Meeting Minutes

30/09/2020

- Edwin has agreed that each instance should have several communities.
- We do not have to agree on the structure of communities.
- We should prioritise protocol features. Items with highest priority go into initial protocol specification (MVP), items with lower priority can be added in the future.
- Functionality beyond this specification can be implemented on a per group basis.

07/10/2020

- JSON to be used for communication.
- For the prioritisation of requirements, check file in the Teams channel.
- Debate on whether to restrict protocol to bare minimum (text/links only) or to include more features that some groups might not implement for MVP.
- Confusion regarding whether user accounts should be global or not. General agreement that they should not be, but someone will ask Edwin.

14/10/2020

- Debate about pub-sub vs fetch-only.
  - Fetch-only won the debate. This decision may be changed after the deadline.
- A collaborative protocol document was made using OpenAPI.

21/10/2020

- 'Allow-list' vs 'Deny-list' model?

  - Groups can decide this for themselves — it does not need to be agreed upon by the supergroup.

- All API paths will be prefixed with `/fed`.
- The agreed upon terminology has been added to the specification.
- There is no standardised branding — groups can choose their own names.
- Agreed that user IDs and community IDs will be human-readable. Agreed on validation rules for this, which have now been added to the specification.
- Agreed to use markdown for formatting.
- The vote to freeze the specification until after the MVP passed unanimously.

28/10/2020

- This was a short catch-up meeting to see how everybody was getting on. Nothing to report.

04/11/2020

- This was a short catch-up meeting to see how everybody was getting on. Nothing to report.

11/11/2020

- This was a short catch-up meeting to see how everybody was getting on. Nothing to report.

30/12/2020

- **Deleting Posts** — Should we do it like Reddit, where the post still exists, but the content is changed to 'Deleted'? Or delete it completely (including children)? Final decision was to delete it completely — simpler.
- **Permissions** — Some groups particularly want this. We have agreed to include the user ID in requests so that groups can implement permissions based on this. They should return the correct HTTP error code to indicate a user does not have the required permissions to perform an action.
- **Security** — Group A3 has put their thoughts in the Teams channel. The summary is to use a subset of the HTTP Signature Protocol.
- **Caching** — Group A3 has put their thoughts in the Teams channel. The summary is to use standard HTTP headers for caching.
- **How to handle comments** — Set the 'title' field to `null` (rather than an empty string).

- **Pushing updates** — An optional proposal to have changes to content pushed. Pull request will be made to GitHub soon.
- **Other features being considered**
  - Wikis
  - Polls
  - Protocol for server discovery
  - Friends
  - Screen names
- **Other** — Agreed to split the 'parent' field into two separate fields: 'community' and 'parentPost'.

06/01/2021

- Discussed the pull request to add the user ID to all requests. Decided to change this slightly and not include it in all requests, as it causes issues with caching. Group A3 will review.
- Approved pull requests 1 – 13 (excluding 6 and 11).
- Wiki proposal is currently being worked on.

20/01/2021

- Approved pull requests 6, 11 and 22.
- Discussed direct messaging. We do not want group messages, but happy for 1-1 messages.

27/01/2021

- Approved the direct messaging proposal.
- Discussed whether multiple of the same content types should be allowed. The discussion will continue in the Teams channel.
- Emphasised the importance of testing between groups.

03/02/2021

- Went through all the PRs and approved the ones we were happy with. Comments have been left on the approved PRs.
- Clarified some aspects of the protocol that groups were confused about.
- Discussed the 'reacts' proposal — further discussion to take place on GitHub.
- Re-emphasised the importance of testing between groups.

10/02/2021

- Discussed whether we should add the host (that the request is being made to the response of all requests). Some groups were very keen on this, but we tried to convince them not to, as it was totally redundant information: to make the request in the first place you must already know the host!
- Discussed the 'reacts' proposal — some more thoughts and suggestions added to GitHub.
- Presented a proposal for federated URLs.
- Clarified some aspects of the protocol that groups were confused about.

17/02/2021

- Agreed to freeze the protocol until after the deadline.
- Some groups are concerned that their supervisors are requesting very specific things — Group A1 to request a Q&A with Edwin.

03/03/2021

- Agreed that there should be no more breaking changes to the protocol.
- Other changes should be as small as possible.
- No more major features.

10/03/2021

No meeting

17/03/2021

- Protocol frozen — no more changes.

24/03/2021

- Tested our implementations together.

31/03/2021

- Tested our implementations together.

07/04/2021

- Tested our implementations together.
- This was the final meeting — no meeting next week as it is the day before the final deadline.

H. Federation Protocol Specification

The protocol specification is available at the following URL:

- https://kiancross.github.io/cs3099a-specification/

In the unlikely event that this URL is inaccessible at the time of reading, an offline version has been included with this report in the `federation-protocol-specification` directory. Open the `index.html` file in a browser to view.

I. Protocol Improvements

In this document, we outline some technical aspects of the protocol that could be improved.

There was intense discussion towards the beginning of Semester 1 over the core approach to share information between servers. There were two competing views:

Fetch Content The 'fetch only' approach that was discussed was for data to be requested on-demand whenever it is needed. A single server would hold a piece of content, and other servers would fetch it whenever required.

Push Content An alternative approach was for servers to 'register' with other servers and have new content pushed whenever it was made available. Historical content would be available, but after making a copy of it for the first time, it would not be necessary to repeatedly fetch the same information.

The main issue with the former is that if a user requires content from ten other instances to load their home page, then ten requests will need to be made per user. Some argue that HTTP requests are 'cheap' and so this does not matter, others viewed this as a poor design. Caching and threading could be utilised to reduce the latency a user would experience from ten sequential HTTP requests, however this could then makes group's implementations far more complicated than the alternative proposal.

Improvements

In the future, we think it would be better to use an established protocol, rather than developing our own. ActivityPub[1] is a well-tested protocol for federated social media networks. Other people have already ironed out the problems and so it makes sense to utilise their work. ActivityPub would have taken a similar approach to the 'Push Content' strategy that we described above.

[1] https://www.w3.org/TR/activitypub/

J. User Stories & Requirements

This appendix outlines the requirements that were submitted in our first deliverable. Minor changes have been made throughout the year. These are organised by the sprint in which they were completed. The final section contains the user stories that were not completed.

Sprint 1

- As a student/member of staff, I want to be able to access the application from my computer.
- As a student/member of staff, I want to be able to register an account so I can participate in the online community.
- As a student/member of staff, I want my information to be kept secure.
- As a student/member of staff, I want to make posts to communities, consisting of text and links.
- As a community administrator, I want the option for my community to be 'open', such that no request is required to subscribe.
- As a student/member of staff, I want to read posts made to any community.
- As a student/member of staff, I want to be able to add comments to a post so that I can have discussions with other users.

Sprint 2

- As a community owner, I want users to have verified emails so that I know they are less likely to be bots or spam accounts.
- As a student/member of staff, I want to be able to change my name and password while logged in.
- As a student/member of staff, I want to be able to write posts using markdown.
- As a student/member of staff studying a STEM field, I want to be able to write posts using typeset mathematics.
- As a student/member of staff studying a STEM field, I want to be able to make posts containing code, with syntax highlighting.

- As a student/member of staff, I want to be able to access content from other servers.

Sprint 3

- As a user, I want to be able to see my account details, such as email, password, username and name.
- As a student/member of staff, I want to be able to change my name and password while logged in.
- As a student/member of staff, I want to be able to reset my password in case I forget it.
- As a student/member of staff, I want to see a profile page for users that exist.
- As a student/member of staff, I want to easily be able to search and find users from any community.

Sprint 4

- Report and testing.

Sprint 5

- Bug fixes and tidy up tasks.

Sprint 6

- As a community administrator, I want an automatic spam filter so that my community does not become filled with spam.
- As a student/member of staff, I want to be able to delete my own posts made to communities.
- As a student/member of staff, I want to be able to delete my own comments.

Sprint 7

- As a student/member of staff, I want to be able to edit my posts.
- As a student/member of staff, I want to be able to edit my comments.
- As a student/member of staff, I want to be able to see replies to other comments.

Sprint 8

- As a student/member of staff with colour blindness, I want the GUI to be colour-blind friendly.
- As a student/member of staff, I want to be able to create a community on a server.
- As a community administrator, I want to be able to set a name and description for my community.

Sprint 9

- As a student/member of staff, I want to be able to subscribe to communities from any server, so that I can see recent posts on my feed.
- As a student/member of staff, I want to be able to unsubscribe from communities.
- As a student/member of staff, I want to have a feed that shows the most recent posts in the communities that I am a part of.
- As a student/member of staff, I want to be able to select custom themes, such as a 'dark mode'.

Sprint 10

- As a student/member of staff, I want my information to be kept secure.
- As a student/member of staff, I want my privacy to respected, for example not selling my information.
- As a student/member of staff, I want to be able to access the application from my mobile phone, so that I can use the application when travelling.
- As a student/member of staff, I want to be able to have group video calls with other students/members of staff within a community.
- As a student/member of staff, I want to be able to mute other users in the group video calls.
- As a student/member of staff, I want to be able to mute my microphone in the group video calls.

Sprint 11

- Improving test coverage

Sprint 12

- As a student/member of staff, I want to be able to report offensive content to the community administrator(s).
- As a student/member of staff, I want a strength bar when creating a new password so that I know my password is strong.
- As a student/member of staff, I want a button to toggle showing my password so that I can double check it when I enter it.
- As a community administrator, I want to be able to delete any post made to my community.
- As a community administrator, I want to be able to delete any comment made to my community.
- As a community administrator, I want to be able to edit any post made to my community.
- As a community administrator, I want to be able to edit any comment made to my community.

Sprint 13

- As a student/member of staff, I want to easily be able to see who the administrator(s) of a community are.

Sprint 14

- As a community administrator, I want to be able to see reports of harmful/offensive content made by users.
- As a community administrator, I want the ability to approve/remove posts before they are visible.
- As a community administrator, I want the ability to approve/remove comments before they are visible.
- As a student/member of staff with a visual impairment, I want the GUI to be screen-reader friendly.
- As a student/member of staff, I want to enter my password twice when registering to make sure I register with the correct password.
- As a student/member of staff, I want to reply to a comment and see my reply appear below that comment.
- As a student/member of staff, I want to see a loading circle after pressing buttons to know if the page is responding or not.

Non-functional Requirements

- The GUI should be quick to load, preferably take no longer than 100 milliseconds to perform any action.
- The GUI should be consistent such that the application feels homogeneous to the user.
- The GUI should be simple so that the user is not overloaded with information and options.
- The GUI should be visually appealing. For example suitable colours and logos can be used to make it interesting to look at.
- The font used in the GUI should be easy to read.
- Text/images should be of a size so that the user does not have to strain their eyes to read it.
- Text/images should not be so large that it dominates the screen.
- Colours of the GUI should be inoffensive and highlight important parts of the site e.g. navigation menus and feeds.
- The GUI should communicate that a user's action has been received e.g. darken a button when it has been pressed.
- Default options, such as choosing to be notified by email about new posts should be pre-selected based on what we think most users would want.
- Error messages should be clear and non-technical.
- Actions, particularly destructive actions, should have an undo button if at all possible.

Uncompleted

- As a student/member of staff, I want to easily be able to search and find users from any community.
- As a student/member of staff, I want to be able to like and dislike posts.
- As a student/member of staff, I want to be able to undo a like or dislike on a post.
- As a student/member of staff, I want to be able to like and dislike comments.
- As a student/member of staff, I want to be able to undo a like or dislike on a comment.
- As a student/member of staff, I want to see links 'previewed' as is done on other social media sites.
- As a student/member of staff, I want to be able to see the number of likes and dislikes a post has.

- As a student/member of staff, I want to be able to see the number of likes and dislikes a comment has.
- As a student/member of staff, I want to be able to sort posts by their popularity i.e. likes/dislikes.
- As a student/member of staff, I want to be able to sort posts by their date.
- As a student/member of staff, I want to have a feed which shows the most popular posts in the communities that I am a part of.
- As a community administrator, I want to be able to give other users moderation capabilities.
- As a community administrator, I want to be able to permanently ban a user.
- As a student/member of staff, I want to be able to 'tag' other users in posts, such that they receive a notification.
- As a student/member of staff, I want to be able to send private messages to other users.
- As a student/member of staff, I want to be able to receive private messages from other users.
- As a student/member of staff, I want to be able to have private conversations with a group of users.
- As a student/member of staff, I want to be able to report offensive private messages to the community administrator(s).
- As a community administrator, I want to be able to read private messages that have been reported by a user.
- As a student/member of staff, I want to be able to block users so that they can not private message or tag me.
- As a community administrator, I want the option for my community to be 'closed', such that only people who are invited can join.
- As a community administrator, I want to review and accept/decline requests from students/members of staff that want to join my community.
- As a community administrator, I want to requests to join my community to be able to include custom fields, such as answering a question or agreeing to some rules.
- As a community administrator, I want to be able to temporarily suspend a user.
- As a student, I want to be able to post anonymously to some communities so that communities such as 'St Fessdrews'[1] and 'St Crushdrews'[2] on Facebook can be recreated.

[1] https://www.facebook.com/stFessdrews
[2] https://www.facebook.com/StACrushes

- As a student/member of staff, I want to be able to delete my account.
- As a student/member of staff, I want to be able to see which communities a particular user is subscribed to.
- As a student/member of staff, I want to be able to see the position of a particular user, such as their year of study or position within a department.
- As a student/member of staff, I want to make posts consisting of file attachments, such as images or PDFs.
- As a student/member of staff, I want to post polls that anybody can see the responses to.
- As a student/member of staff, I want to post polls that other people can add response options to.
- As a member of staff, I want to be able to post polls to which only I can see the responses.
- As a student, I want to be able to react to content using emojis, so that I can indicate a further range of feelings rather than just 'like' and 'dislike'.
- As a community administrator, I want to be able to delete my community.
- As a community administrator, I want to set rules such that users with fewer than n posts to my community must be approved by me, to try and reduce spam on the community.
- As a community administrator, I want to be able to add a 'flare' to a user, which is displayed next to their name so that it easy to identify special users in the community.
- As a student, I want to contribute to a shared wiki, which is a part of a community.
- As a community administrator, I want all changes to a wiki to be revertible so that poor quality edits may be reverted.
- As a community administrator, I want to be able to approve changes to a wiki before they are made visible to everyone.
- As a lecturer, I want to be able to embed pre-recorded lectures and videos within my posts.
- As a student, I want to be able to easily engage in discussions relating to aspects within pre-recorded lectures, perhaps by using timestamps that link to the relevant point.
- As a community administrator, I want to be able to pin important posts so that they remain at the top of community's posts page.
- As a student/member of staff, I want to be able to set up two-factor authentication to keep my account secure.

- As a community administrator, I want to be able to send a welcome message to new subscribers.
- As a student/member of staff, I want to be able to opt-in to a daily/weekly digest of the latest posts to the communities I am subscribed to.
- As a community administrator, I want to be able to enable/disable features on my community, such as likes/dislikes, allowing people to post anonymously and allowing people to post different types of content such as polls and images.
- As a community administrator, I want to be able to style and theme my community with a header.
- As a community administrator, I want to be able to style and theme my community with different colours.
- As a student/member of staff, I want new content that I may find interesting recommended to me, so I can discover new communities.
- As a student, I would like to play little games with other users, such as noughts and crosses.

K. Feature Descriptions

In this appendix, we describe each feature in detail and explain how it can be tested.

Registration

Description

- Users can register an account using a username, email, name and password.
- The user must enter their password twice in order to verify that it has been entered correctly.
- The password must meet the minimum strength requirements, which are measured using heuristics rather than character/length requirements.
- The submit button will be greyed out until all fields are valid.
- A message will be displayed underneath any field which is not valid.
- The user must confirm that they have read the privacy policy before they can register an account.
- On successful submission of the form, the user will be sent an email containing a verification link. Visiting this link will verify their email address.
- Upon email verification, they will then be able to access their account.

Invalid Test Cases

- Entering no email address.
- Entering an invalid email address.
- Entering no username.
- Entering an invalid username.
- Entering no name.
- Entering an invalid name.
- Entering no password.
- Password not being complex enough.
- Confirmation password not matching the initial password.
- Checkbox confirming that the user has read the privacy notice not checked.

Valid Test Cases

- Entering a valid username, name, email and password and checking the privacy notice checkbox.

Sign-in

Description

- Users can sign-in using the email and password that they provided at registration.
- On successful login, the user will be taken to their homepage.
- On an unsuccessful login, the user will be displayed an error message.
- The user will not be able to login if they have not verified the email on their account.

Invalid Test Cases

- Entering an email and/or password that do not belong to a user.
- Entering an email and password that belong to a user who has not verified their account.
- Entering an email and password belonging to a verified user.

Valid Test Cases

- Entering an email and password belonging to a verified user.

Content Editor

The 'content' editor refers to the editor used to write and edit posts and comments.

Description

- Content is written using the markdown format.
- A preview of the rendered markdown is displayed next to the raw markdown.
- The content editor should allow TEXmaths to be written.
- The content editor should support code syntax highlighting.
- The content editor should allow tables to be created.
- The editor provides a number of useful functions such as headings and underlining so that the user does not need to know markdown to make a post.

Creating a Post

Description

- The page to create a post consists of an input for the title and an input for the main body of the post.
- A button at the bottom of the page creates the post when clicked.
- An error is displayed to the user if the post could not be created.
- On success, the user is redirected to their newly created post.

Invalid Test Cases

- Entering no title and clicking 'Create'.
- Entering no post body and clicking 'Create'.

Valid Test Cases

- Entering a title and post body and clicking 'Create'.

Creating a Comment

Description

- Users can create a comment on any post by typing into the content editor on any post's page.
- They can also reply to another comment.
- They can then click the 'Add Comment' button to submit the comment.
- An error is displayed to the user if the comment could not be created.
- On success, the created comment is added below the post.

Invalid Test Cases

- Entering no text into the content editor and clicking 'Add Comment'.

Valid Test Cases

- Entering some text into the content editor and clicking 'Add Comment'.

Editing a Post

Description

- Users can edit posts they have created by selecting the 'edit' option from the drop-down menu in the corner of the post.
- The content editor is then shown with the post's existing content.

- The user can change this as they wish and click the ‘Save Post‘ button to save the edited post.
- If the post fails to save, an error will be displayed.
- On success, the post will be updated on the page.

Invalid Test Cases

- Attempting to save the post with no title.
- Attempting to save a post with no body.
- Attempting to edit a post that you did not create.

Valid Test Cases

- Editing the title of a post you created with some other text.
- Editing the body of a post you created with some other text.

Editing a Comment

Description

- Users can edit comments they have created by selecting the ‘edit’ option from the drop-down menu in the corner of the comment.
- The content editor is then shown with the comment’s existing content.
- The user can change this as they wish and click the ‘Save Comment‘ button to save the edited post.
- If the comment fails to save, an error will be displayed.
- On success, the comment will be updated on the page.

Invalid Test Cases

- Attempting to save a comment with no body.
- Attempting to edit a comment that you did not create.

Valid Test Cases

- Editing the body of a comment you created with some other text.

Deleting a Post

Description

- Users can delete posts they have created by selecting the 'Delete' option from a drop-down menu in the post.
- If the post fails to delete, an error message will be displayed.
- On success, the user will then be redirected to their home page and the post will be removed.

Invalid Test Cases

- Deleting a post that you have not created.

Valid Test Cases

- Deleting a post that you have created.

Deleting a Comment

Description

- Users can delete comments they have created by selecting the 'Delete' option from a drop-down menu in the post.
- If the post fails to delete, an error message will be displayed.
- On success, the comment will be removed from the page.

Invalid Test Cases

- Deleting a comment that you have not created.

Valid Test Cases

- Deleting a comment that you have created.

Administration Privileges

Description

- Deleting a post/comment when you are not the author of the comment or an administrator of the community the comment is part of.

Invalid Test Cases

- Deleting a post/comment when you are not the author or the administrator of the community.

Valid Test Cases

- Deleting a post/comment when you are not the author but are the administrator of the community.

Moderation Queue

Description

- Community administrators can see a list of unapproved posts and comments from communities they moderate.
- Checkboxes next to each post and comment can be used to select them.
- They can then click the 'Approve' or 'Remove' buttons to approve or remove the selected posts.
- On success, the selected posts will disappear from the list.

Valid Test Cases

- Approve a post from the queue.
- Remove a post from the queue.

Reporting a Post or Comment

Description

- Users can report comments of other users by clicking the 'Report' option from a drop-down menu in the post header.
- On success, the content of the post will get hidden and placeholder text is shown instead.
- This feature does not work with posts on federated servers.

Valid Test Cases

- Users should be able to report posts of other users from the same server.

Password Reset

Description

- If the user forgets their password, they can enter their email address and get a password reset link sent to them.
- Clicking the link will take them to a page where they can enter a new password.
- This password is subject to the same complexity requirements as on the registration page.
- On successful reset, the user it taken to the login page.

Invalid Test Cases

- Entering an invalid email on the password reset page.
- Entering a password that does not meet the complexity requirements.

Valid Test Cases

- Entering an valid email on the password reset page.
- Entering a password that meets the complexity requirements.

View Profile

Description

- Each user has a profile that can be accessed by other users.
- A user's profile page displays their username and the posts that they have made to other communities.
- If the user does not exist, a 404 error page should be displayed.

Invalid Test Cases

- Visit the profile page of a user that does not exist.

Valid Test Cases

- Visit the profile page of a user that exists.

Account Settings

Description

- Authenticated users can view and edit their account details, particularly their password and name.
- The page consists of two tabs, the first for editing account information (such as password) and the second for editing profile information (such as your name).

Invalid Test Cases

- Enter the incorrect old password, correct retyped password and a new password.
- Enter the correct old password, incorrect retyped password and new password.
- Leaving any fields blank.

Valid Test Cases

- Enter correct old password, correct retyped password and a new password that meats the complexity requirements.
- Enter a valid new name.

Creating a Community

Description

- Users can create a community by clicking the 'plus' icon in the header of the application.
- They are then prompted with a popup that asks them to enter the community ID, name and description.
- They can then click the 'Create Community' button to create the community.
- On success, the user is redirected to the new community's posts page.
- If there was a problem creating the community, an error is displayed to the user.

Invalid Test Cases

- Creating a community with an ID that is already used.
- Creating a community with any fields blank.

Valid Test Cases

- Creating a community with a unique ID, name and description.

Searching for Communities

Description

- Users can search for communities using the search bar in the header of the app.
- Upon typing the name of the host, a drop-down will provide a list of all communities that exist within the host.
- Clicking on a community in the list will take the user to it.

Invalid Test Cases

- Search for a host that does not exist.

Valid Test Cases

- Search for a host that exists.

Group Video Chat

Description

- Each community has a 'room' that can be entered to join a group video chat. These rooms are scoped to communities.
- Upon entering the room, your browser will ask to access your camera and microphone.
- After joining, you will be able to speak to and hear everybody else in the room.
- The 'Leave Call' button removes you from the room.
- The 'Mute' button on your own video mutes your own microphone.
- The 'Mute' button on somebody else's video stops you from being able to hear them.

Valid Test Cases

- Joining a call in the same community as someone else.
- Leaving a call.
- Muting yourself.
- Muting somebody else.
- Exiting your browser whilst in a call.

Machine Learning Spam Filter

Description

- Each comment and post is automatically checked against our spam detection model.
- If the post is spam, it is marked as 'pending approval' and requires a moderator to approve it before it can be seen by other users.

Test Cases

- Spam posts should be marked as 'pending approval'.
- Non-spam posts should be displayed immediately.

Text Toxicity Detection

Description

- Each comment and post is automatically checked against a text toxicity classifier.
- If the post is toxic, it is marked as 'pending approval' and requires a moderator to approve it before it can be seen by other users.

Test Cases

- Toxic posts should be marked as 'pending approval'.
- Non-toxic posts should be displayed immediately.

Nested Comments

Description

- Replies to comments are displayed in a nested form, where the original comment is full width, with replies to it being indented and being displayed below.

Test Cases

- A comment A in response to another comment B should be indented and displayed below B.

Dark Mode

Description

- Users can toggle the application to be in a dark mode, where all elements of the page have a darker theme applied to them.
- This can be done by clicking the toggle in the application header.

Test Cases

- Enabling dark mode should darken the theme.
- Disabling dark mode should revert to the light theme.

Subscribing to a Community

Description

- Users can subscribe to a community by clicking on the 'Subscribe' button on the community's page.
- They can unsubscribe by clicking the same button, which displays 'Unsubscribe' when the user is subscribed.

Test Cases

- Subscribing to a community.
- Unsubscribing from a community.

Home Feed

Description

- A user has a home feed which shows recent posts from all the communities that they are subscribed to.

Test Cases

- Subscribe to a community and check that its posts are displayed on the user's home feed.
- Unsubscribe from a community and check that posts are no longer displayed on the user's home feed.

L. Privacy Notice

We have written this to the best of our ability using the information and guidance from the ICO's website.

Our contact details

- **Name**: Unifed
- **Address**: —
- **Phone Number**: —
- **Email**: cs3099.a3.unifed@gmail.com

The type of personal information we collect

We currently collect and process the following information:

- Personal identifiers, contacts and characteristics (for example, your name and contact details).
- Online identifiers (for example, your IP address).
- Content that you create or upload, such as posts.

How we get the personal information and why we have it

Most of the personal information we process is provided to us directly by you for one of the following reasons:

- To signup to the website.
- To create and upload content.

We also receive personal information indirectly, from the following sources in the following scenarios:

- Your IP address, which is sent to us whilst you access the website.

We use the information that you have given us in order to:

- Provide you with access to the website.
- Allow you to create content on and upload content to the website.
- Protect other users on the website.

Under the General Data Protection Regulation (GDPR), the lawful basis we rely on for processing this information is your consent. You are able to remove your consent at any time. You can do this by contacting us using the details at the top of this notice.

How we store your personal information

Your information is securely stored in the United Kingdom. Your personal information is stored whilst you have an account on the website. If you delete your account, we will erase your personal data.

How we process your personal information

Personal identifiers, contacts, characteristics (for example, your name and contact details) and your IP address are always kept private. Your username and any posts you make are publicly available. Publicly available content may be saved and downloaded by third parties outside of our control.

Your data protection rights

Under data protection law, you have rights including:

- **Your right of access** — You have the right to ask us for copies of your personal information.
- **Your right to rectification** — You have the right to ask us to rectify personal information you think is inaccurate. You also have the right to ask us to complete information you think is incomplete.
- **Your right to erasure** — You have the right to ask us to erase your personal information in certain circumstances.
- **Your right to restriction of processing** — You have the right to ask us to restrict the processing of your personal information in certain circumstances.
- **Your right to object to processing** — You have the right to object to the processing of your personal information in certain circumstances.
- **Your right to data portability** — You have the right to ask that we transfer the personal information you gave us to another organisation, or to you, in certain circumstances.

You are not required to pay any charge for exercising your rights. If you make a request, we have one month to respond to you. Please contact us using the contact details at the top of this notice if you wish to make a request.

How to complain

If you have any concerns about our use of your personal information, you can make a complaint to us using the contact details at the top of this notice. You can also complain to the ICO if you are unhappy with how we have used your data. The ICO's:

- **Address**: Information Commissioner's Office, Wycliffe House, Water Lane, Wilmslow, Cheshire, SK9 5AF.
- **Helpline number**: 0303 123 1113
- **Website**: https://www.ico.org.uk

M. Source Code

The source code is available at the following places:

- https://github.com/kiancross/unifed/

  Access to the private GitHub repository can be provided quickly on request.

- https://gitlab.cs.st-andrews.ac.uk/cs3099group-a3/project-code/
- The `soure-code` directory submitted with this report.

We recommend browsing the source code on GitHub. First, `README.md` files will be rendered. Second, commit messages that reference issues/pull requests will have clickable links. However, this is not essential.

N. Open Source Contributions

During development of this project, we have made some small open source contributions. These can be seen below.

Dependency Fixes The package manager we used, Yarn 2, is far stricter about dependency declarations in `package.json` files than others, such as `npm`. We encountered some mistakes and submitted pull requests to fix these.

- https://github.com/accounts-js/accounts/pull/1085
- https://github.com/accounts-js/accounts/pull/1086
- https://github.com/accounts-js/accounts/pull/1087
- https://github.com/accounts-js/accounts/pull/1088
- https://github.com/accounts-js/accounts/pull/1115
- https://github.com/apollographql/protobuf.js/pull/6
- https://github.com/EndemolShineGroup/cosmiconfig-typescript-loader/pull/133
- https://github.com/apollographql/apollo-tooling/pull/2199
- https://github.com/typestack/class-validator/pull/870
- https://github.com/avajs/ava/pull/2681
- https://github.com/facebook/create-react-app/pull/10334
- https://github.com/HarryChen0506/react-markdown-editor-lite/pull/182

Other We made some other miscellaneous contributions throughout the project.

- https://github.com/neoclide/coc.nvim/pull/2807 and https://github.com/yarnpkg/berry/pull/2360
  - This contribution fixed problems when using the `coc-tsserver` extension with `coc.nvim`.
- https://github.com/yarnpkg/berry/pull/2325
  - This contribution made debugging Yarn 2 extensions easier by generating a source map.

- https://github.com/trivago/prettier-plugin-sort-imports/pull/24
  - This contribution fixed a problem in the plugin where intentional new lines were being removed.

O. Deployment Plan

In this document, we briefly describe our production deployment plan. We discuss resource provisioning, availability, scalability, error reporting, backups, and recovery. This deployment plan has not been implemented, primarily due to the associated costs of deploying to cloud providers.

Resource Provisioning

Terraform[1] is an infrastructure as code tool that can be used to provision resources across a variety of cloud platforms. This will be used to create reproducible deployment environments, with changes tracked in version control just like with normal code.

Availability and Scalability

Using cloud providers is a simple way to create an available and scalable application. Rather than needing to purchase, configure, and maintain hardware ourselves, cloud providers can be used to purchase computing capacity on demand.

Using MongoDB allows us to scale horizontally across multiple machines. Most cloud providers sell managed instances of these NoSQL databases, which takes care of the configuration, scaling, availability, security etc.

Our application is designed to have multiple instances running, with load balancers used to distribute incoming requests. This means it can also be scaled horizontally. Running multiple instances in different data centers, potentially on multiple cloud providers gives a high level of redundancy. The latter aspect would be more tricky to orchestrate and is non-essential.

Error Reporting

To monitor the health of the application, error reporting and logging is necessary. This is also essential for security auditing. ElasticSearch[2] will be used for ingesting log and error data. This is easily searchable and information can be presented visually on monitoring dashboards. Cloud platforms also provide monitoring dashboards for observing the health of the provisioned servers.

[1] https://www.terraform.io/
[2] https://www.elastic.co/

Backups and Recovery

In case something goes terribly wrong, backups are an important fallback. However, it is equally important to not only have backups, but to plan a recovery process. We will make backups of the system every hour and store these with the cloud provider. Regularly these will be downloaded and stored offline, for additional redundancy. We will write a plan for recovering the system and test the plan often — so we are ready if something does go wrong.

Hourly backups will be retained for a week. Weekly backups will be retained for a month. Monthly backups will be retained for six months. It is important not to keep backups for longer than necessary, both for cost reasons and data protection concerns.

Security

Encryption at rest will be enabled for our backups, database and other servers. HTTPS will be enabled between components for encryption in transit. Access to servers will be restricted only to authorised and necessary personal.

Those logs, errors, and those with access to the backend of the system will be audited frequently to keep the system secure and detect any intrusions. Firewalls will be enabled to whitelist remote access to only certain IP addresses. This will reduce the attack vector surface.

P. Technical Changes

This appendix is included as per the guidelines in the student handbook.

We carefully evaluated our technical choices in the first deliverable and also made small changes throughout the project. This means we are in a position where we are generally happy with the technology choices we are using at this stage.

We think our choice of language was good, MongoDB worked well and React was an effective frontend library. However, we think overall we had too much flexibility and choice. The libraries we were using were not very opinionated and so we spent a lot of time trying to make decisions. We were not experienced in many of the technologies, which made these decisions hard. Conflicting opinions online made these decisions even more difficult.

Next.js[1] is a React framework which has already made many of the decisions for you. However, it is not just a frontend framework — it also handles the backend. Had we known about this full-stack solution, we would almost certainly have used it. By the time we were aware of it, we were too invested in our current approach to change.

As far as we can see, this was the only major change we would want to make in the future. The other technical aspects of our project were successful.

We discussed in our first deliverable the idea of severless computing. We are still keen on this, but stick with our original decision that it would have been inconvenient to orchestrate using the School's resources.

[1] https://nextjs.org/