**7CCSMGPR GROUP PROJECT**

**FINAL REPORT**

28TH March 2019

Lucky Group:

* Robinson, Thomas James 1880679
* Choi, Daeun 1821307
* Wu, Zihao 1836287
* Fu, Yixin 1862819
* Sarosi, Marcell 1542196
* Zhu, Tianning 1817456

Contents

1. Introduction
2. Context
3. Achievements
4. Review
5. Google Drive
6. GitHub
7. Dropbox
8. OneDrive
9. Requirements and Design
10. Implementation
    1. Web Application
    2. Mobile Application
11. Team Work
12. Evaluation
13. Peer Assessment

1. Introduction

i. Context

For the last 12 weeks Lucky Group have been developing a Multi-Host File Synchroniser which is accessible on mobile and web platforms. The want for ‘multi-host’ file systems has increased dramatically with the development of the web; allowing people to collaborate on complex projects means that there is a greater pool of knowledge contributing, which in turn leads to greater work being produced and in less time. As well as allowing more people to contribute to a project, a multi-host system also allows people to work on their projects from wherever they are as long as they have an internet connection. This further aids the ability of home-based teleworking, which has been found to increase productivity and decrease job attrition by over 50% (Bloom, Liang, Roberts, & Ying, 2015). As multi-host systems allow remote access to files and as companies are looking to decrease overhead at every point, this kind of system (which increases flexibility for employers and employees) is significantly increasing the possibility of removing the need for a physical office location. This is just one of the reasons they are becoming so popular.

Another reason multi-host systems popularity is increasing is teamwork. One of the main aspects of working as a team, especially if working remotely, is consistency. Each user must be working from the latest version of the document and be updated with new content as and when it is needed. This is where the synchronisation becomes important. Synchronisation means every user on the system can see where the project is currently at. This also allows management to make more informed decisions to keep the project on track. Synchronisation is the key to allowing multiple users to access the same document at the same time without corrupting the master file when the two versions are uploaded. Systems can do this by scanning documents to see the changes and then merging them. Depending on the rules set when the system is created, additions and deletions will be made to the master document as each user uploads their version. There are a few ways this can be executed, and the choice made will depend on the user’s needs for the system. These methods will be discussed later in the report.

Synchronisation is the solution to the problems when working on multiple hosts, there needs to be a way in which users can be confident that they have data that is consistent with the rest of their team and that when they make changes to the project, their changes will be transferred to the rest of their team without any errors or deletion of any work that is still relevant to the system.

ii. Summary of Achievements

As a group we have achieved the completion of a mobile application and a desktop application that will allow users to upload and download documents to and from their own disk space. After downloading a file from either application, the user can edit the document in whatever way they please. When they have finished editing the file can be re-uploaded under the same file name, at this point the system will identify any changes have been made and update the file on the system with the latest version of the document. If there is a conflict of two users trying to upload at different times, the most recent upload will take precedence over the older upload, these files will then be available to see in the database. To be able to complete any tasks on our system the user has to log into the system, where they will then be presented with all the files on the system, this is where the cycle of downloading an uploading may occur.

Along with the completion of the synchroniser we have also created two reports, of which this is one, and given two presentations documenting the implementation and organisation of the project.

2. Review

I will now discuss some of the file sharing systems that are currently on the market. These sites are owned by some of the largest technology companies in the world which shows how valuable this kind of system has become. This is evidenced by all of the systems under review having hundreds of millions of users. Many of these systems have free access up to a certain point of data storage, when this point is reached to gain access to more space on their database you must purchase a membership. In this review I will be looking at the positives and negatives in these systems and suggesting ways in which these could be utilised for our system. The systems I am reviewing are mainly aimed at average consumers instead of corporate customers as this is what our system will be directed towards.

i. Google Drive

The first system to be reviewed is Google Drive, owned and created by Google, which has over one billion users (reported at a Google conference, reported by (Tech Crunch, 2018)). Google Drive offers online and offline platforms in the forms of a website, applications for desktop users on Windows and macOS, and applications for mobile platforms including Android and iOS smartphones and tablets. This platform allows you to store any file you want up to 5 TB per file, with only the first 15 GB being free you must subscribe to Google One to receive additional space (up to 30 TB). Drive also offers editing capabilities directly within the system by including other Google products on the platform. All information (Google, 2019).

The most interesting feature of Google Drive, that could be implemented onto our system is the ability to directly edit files on the platform without having to download the file to your local disk space. This feature speeds up the process of making simple adjustments to the file as you do not have to go through the upload and download process. This also means that users save room on their local disk and on their bandwidth as the document may be a large file that would affect other users on the same network. One disadvantage of using Drive is that it does not offer offline applications for Linux systems and Windows phone, although these only account for around 2% (Net Market Share, 2019) and 1% (Statista, 2018) of the average consumer market[[1]](#footnote-1) share respectively, there will still be a large amount of users that will not be able to effectively handle offline file sharing (Table 1).

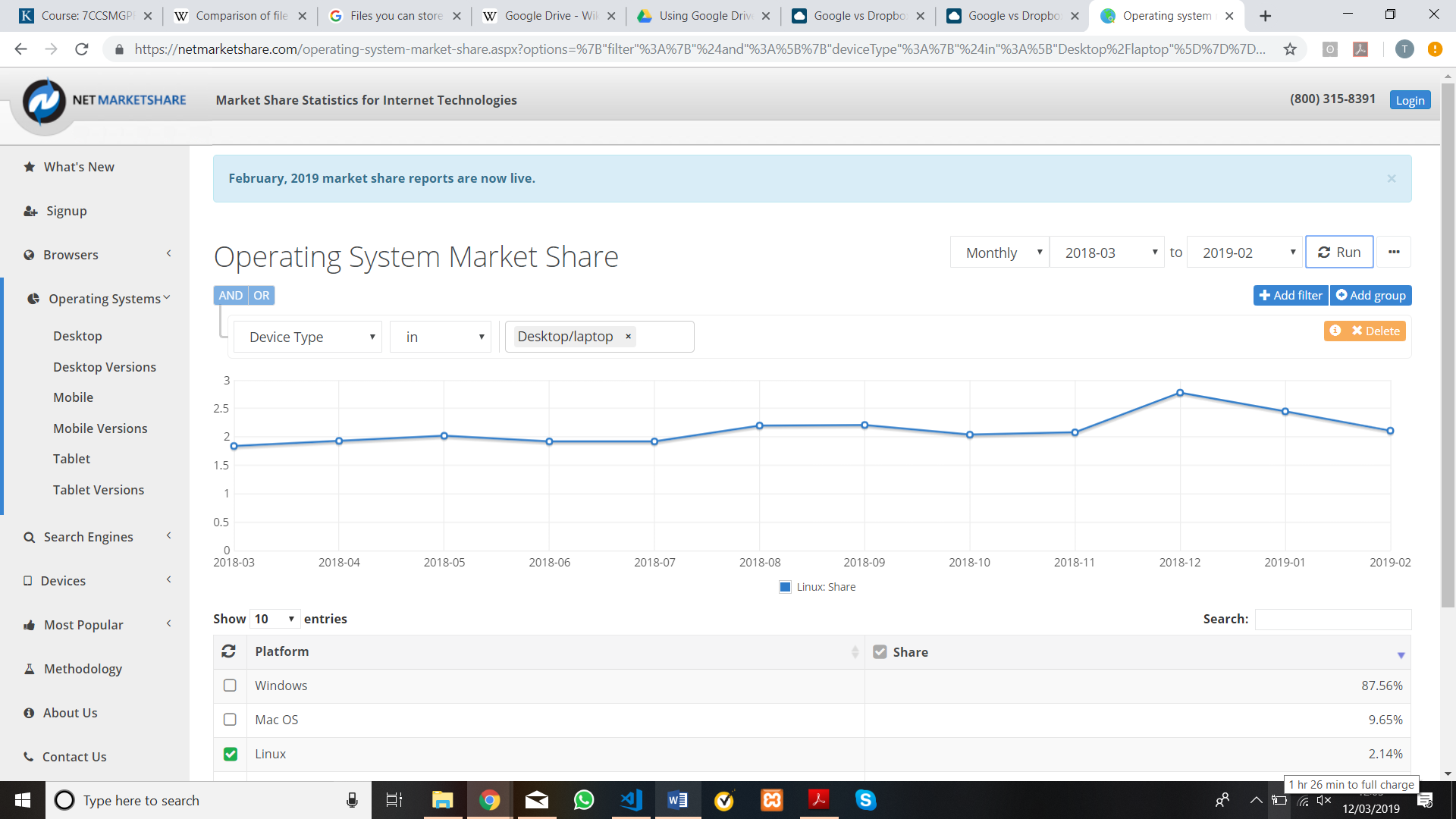
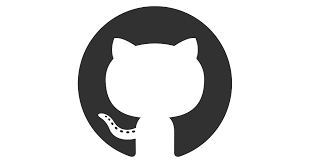


Table 1 - Linux Market Share for the Last 12 Months

Another disadvantage to Drive is the way that that you share files with other users, by sending a link to the file or folder that you wish to allow the receiver to edit. This is a disadvantage as the way they have implemented this system (no option to have a password on the links and no expiry dates or download limits) mean that the security of the shared file / folder could be easily compromised.

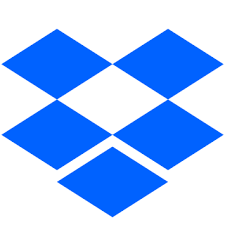
ii. GitHub

The next system under review will be GitHub. This is now owned and run by Microsoft after acquisition in 2018 (Warren, 2018). It has over 32 million users (GitHub, 2019), making it the largest host of source code in the world[[2]](#footnote-2) (Gousios, Vasilescu, Serebrenik, & Zaidman, 2014). GitHub Desktop is the application that allows you to perform GitHub operations without accessing their website. You can also perform all the GitHub features directly in the command line with third party software installed on your system e.g. Git BASH or Git for Windows. There is no official application on either iOS or Android, although there are apps to remove notifications e.g. GitHawk for GitHub. There is no set disk quotas on GitHub, within reason, stating on their website “Keeping repositories small ensures that our servers are fast and downloads are quick for our users.” (GitHub, 2019). GitHub is a more complex file sharing system which is known as a Git repository service. It is often used for sharing projects involving computer code as it offers all the distributed version control and source management functionality of Git.

The main advantages that GitHub has involve its distributed version control. Distributed version control means that you, or anyone who has access to the repository, can update your file from any location allowing for collaboration. Git also allows you to compare changes to a previous version, which means if the program no longer works properly you can compare the changes that were made to the previous version to rectify or revert to the previous version. The other great advantage for GitHub is the branching feature. This feature allows you to make changes to the document in isolation and then merge it with the main branch when you’re sure it works correctly. This means that one user can work on the project at the same time without affecting any other users work, then when they merge to the main line the changes will be implemented independently. These are the features that make GitHub a prime contender for file sharing if the file includes computer code.

One disadvantage of GitHub is the learning curve experienced when first using GitHub. It isn’t as intuitive to use as the other multi-host file synchronisers discussed in this review, with many users claiming it is aimed more at programmers. This means it doesn’t communicate with layman’s so well, meaning many people use the CLI[[3]](#footnote-3) over the GUI[[4]](#footnote-4). This is something that this project will try to avoid by having a clean and simple GUI.

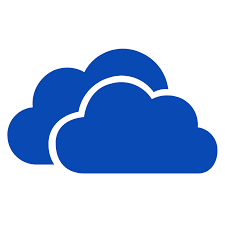
iii. Dropbox

Dropbox will be the next system under review. It is owned and run by Dropbox, Inc, which has over 500 million users (Dropbox, 2016). Dropbox offers offline applications for all three major desktop operating systems and has mobile applications for Android, iOS and Windows Phone. Dropbox can also be set up to have a special folder on the user’s computer of which can be synchronised with Dropbox servers and other computers that have Dropbox installed as well as smartphones with the mobile applications. Dropbox comes with free storage of up to 2 GB, at this point you can upgrade to a paid personal plan which includes 2 TB or a paid plan ‘For Teams’ which have either 3 TB or an unlimited amount of space (Dropbox, 2019). Dropbox only has one native application built into the system, Dropbox Paper, which is essentially for note taking. However, Dropbox does allow for third party apps to connect adding extra functionality.

For Dropbox, the main advantage of this system lies with its file sharing capabilities. Although you must have the file on your local disk as well as on the cloud service, meaning you do not save any disk space on your local file, Dropbox makes use of block level file transfer. This essentially means the only time the whole file is transmitted to the server is the initial upload, with the future edits being uploaded in 4 MB blocks, this means that files should sync faster.

A disadvantage of using Dropbox compared to other systems is the fact that you must use third party applications to edit any documents, although they have Office integration, there are still some office applications that are missing. Another disadvantage is the amount of space and functionality that is granted with the free membership. Although, the pricing, storage and app integration aspects aren’t features that we will be implementing in this project.

iv. OneDrive

The last system I will be reviewing is OneDrive which is run by Microsoft and again has hundreds of millions of users. OneDrive offers desktop applications for Windows and Mac users, as well as mobile applications for Android and iOS, but not for Linux and Windows Phone users. OneDrive also allows for a folder to be set up in the users file explorer in which you can directly transfer files to the cloud service and this comes as a default on Windows enabled computers. OneDrive comes with 5 GB of free storage, which can be upgraded to 50 GB of storage without any other subscriptions, at this point there are only ‘premium’ accounts. These accounts include other Office 365 applications and offer either 1 TB or 6 TB (for 6 users at 1 TB per person) (Microsoft, 2019). As this is a Microsoft service, Online Office is readily available, as well as OneNote, which is Microsoft’s answer to Dropbox Paper. Any other third-party application integration is harder to come by, although Office does cover most of the essential tasks that you could want to complete without downloading the file.

A strength that OneDrive has is its file sharing operations, this has a clean and simple mechanism to share and collaborate on projects with other people. With password and expiry date capabilities, you can be sure that the intended recipients are the only person/people that can read the documents. OneDrive also makes use of block level file transfer, although not at the same rate as Dropbox, it still makes for faster upload speeds.

One thing that I have found from personal use of OneDrive is that when the space is full in your OneDrive account there are persistent notifications prompting you to upgrade your account. With the synchronisation a default on Windows systems, this becomes full quickly and this has led me into using another service for cloud storage. Other users have complained about OneDrive struggling to deal with files with hundreds of sub folders and subsequently a large number of files[[5]](#footnote-5), meaning OneDrive would not be suitable for people with large amounts of data to transfer or for companies to hold their data on.

3. Requirements & Design

This section will outline the requirements we set out to achieve as a group and the reasons we made these decisions. This chapter will also include how the design decisions were affected by the requirements we wanted to fulfil.

The first requirement decision we made for the system was to decide how to handle what would happen should a conflict of uploads occur. This choice was made at a similar time to the next strategic decision, deciding what should happen when there was a conflict changes depending on whether there was a constant connection with the server or the manual upload method that we settled on. A conflict can be defined as what happens when two users who are working from the same version of a file locally, then proceed to make different changes to the document and then upload. If a conflict occurs we want our system to produce a pop-up that will prompt you to decide between **replacing the file** in the system or **renaming the file** with an additional identifier attached to the file name. This will work by using the ‘rsync’ algorithm, which will be further explained in the implementation section.

Should you choose to replace the file, the document in the database is currently in the will be permanently deleted and that version of the file will only be held on the local disk of the user that uploaded that version. We decided to use this method of syncing the files as this allows for more control over the master copy, which means that if someone was unsure if they should be updating the master file, they can save it under a new name. This file can then be reviewed by the team of people working on the project who can, at this point, decide if they want to commit the changes to the master copy. Although this method may lead to an untidier database with multiple copies of the largely similar files, this can quickly be fixed with efficient team work to update the master copy. Another reason we wanted to use this method was that it would be easier to implement a ‘manager’ profile with user access levels. This would allow the system to have an omniscient figure that can ensure that only what is required goes into the latest version of the system and nothing pivotal is deleted.

This is one of the defining features of our system that deeply affected the rest of the design decisions. If we had decided to use the timed synchronisation method with this strategy, there could be constant stream of uploading new files should there be a conflict with another user working on the file at the same time. This would be an inefficient use of space on the database and would also further increase the strain on the connection as uploading a whole new file every X minutes would take up a lot of bandwidth. There was a discussion within the group about the type of synchronisation that we wanted to implement, with the other option being the master copy being updated with the latest changes. Should a conflict arise using this method the changes would be merged into one document on the system with a mix of the changes implemented on the document. But having decided on using the manual uploading the majority of us agreed that this was the most effective way of handling conflicts as this gives the team more control over what the master copy contains.

The next attribute we discussed as a group was whether we wanted the user to manually upload the file to the system or for there to be a synchronisation update of the file on the system every X minutes (without any action from the user). We came to the decision to keep the responsibility of uploading to the user. The main reason we decided to use this method is because it will provide less opportunity for conflicts to occur, meaning that when the file is overwritten there is less chance of your work being deleted.

If the auto synchronisation method was used with the merging technique mentioned above, files would be deleted as they were being written if two people were writing to the same document at the same time. This could cause confusion for the authors and potential loss of work. This method also doesn’t require a constant connection with the server, allowing for offline work to be completed without exceptions being called. When there is less connection with the server, this is also a benefit for the rest of the network as there is more bandwidth space for other users to perform their tasks. This also means that if a user was to lose connection with the server and close their local version without saving (with them assuming a version had been uploaded to the system), they would lose all their work up to the last upload point or all the work created since opening the file.

Again, this is one of the main strategic decisions that shaped the outcome and design of the project as this would require less UI for the user to interact with and potentially include a file in the user’s desktop similar to the way that OneDrive and Dropbox do. This would also place a lot of work on the server side of the system, with our experience as coders not being in server-side programming, we decided to keep most of the work being in the user’s application. This decision was made unanimously within our group, with everyone agreeing that this method was the best way for us to proceed.

When you first load our application, you will be presented with a login page. At this point you can either login with a previously registered account, if you don’t already have an account you will have to register with a valid email and matching passwords. We decided we wanted to include user login so each user can have a page which they can see all the files they have uploaded. This also means that if someone edits a document already on the system there will be a signature attached to this edit, allowing people to see who changed the document and when. Having user accounts also increases the security of the system as only authorised user will be able to access the database. It also allows us to potentially implement locks on certain documents meaning an author could be able to decide on specific users that can edit a document. User accounts also mean that implementation of a manager profile can be set at the start of the process, giving them different abilities to average users.

This decision allowed us to have a more traditional flow of a file sharer and increase security but the design of the system stayed largely the same if there wasn’t a user login at the start of the system. The only difference in design would be depending on if the users have different access levels where one user could have a higher priority when it came to replace the master copy on the database. This level of user could also receive a notification or an email every time someone makes changes to a document that they originally uploaded. The decision to have a login page was unanimously agreed from the start of the project, the use of access levels was also agreed but we decided the implementation of this system was to have a low priority as this is not vital to the core components of the system.

Users of this system will be able to upload any document type to the system; this process occurs on the ‘upload’ page which is accessible from the main menu when the user has passed the login page. There will be looking to implement options to either drag the file directly from the file location open in another program or search their files from a button on the application. When uploading an item, you will be able to add certain footnotes to give other people looking at the item a more information about what is in the folder / document.

As we have decided to allow any document type to be uploaded to the system, there will not be any function to edit documents directly on the system. This is because there is a vast number of file types that could be uploaded so it would not be possible to implement something that could edit all types of files on this system. The decision not include online editing was agreed within the group to be a low priority and the concentration was to be placed on the core functionality of the system, something that this does not fall under. We are however allowing users to rename files on the system, this works by using a HTTP POST request.

One of the main features of our system, considering we aren’t using within-app editing, is the download function. A user will be able browse all the files they have uploaded and any files that have been made available to see by all the users on the system. When they find the file they want to download, they will be able to inspect the document (read the information provided by uploader), then decide if they want to continue with downloading the item. The user will then be able to DECIDE where they want the file to be saved to. This works by using a HTTP GET request.

The ability to only see the documents you have uploaded and ones that have been marked public allows a user to have a system that is like GitHub in the sense that when a user is happy with the version they have been working on privately they can reupload to replace the master copy.

There will also be the option for a user to delete a file from the system if they want to. A user may only delete a file that they have uploaded or edited to prevent a user accidentally deleting another user’s work with a similar title. This may also be implemented with the access levels, meaning that if a user has a ‘manager’ account they may be able to delete all the files in the system as they are trusted to know what is and isn’t important. This works by using a HTTP DELETE request.

We have designed the database to include 3 tables; User, Password and File. The User table will contain a list of usernames, the Password will contain a lift of passwords, the file table will contain information about the files on the system; this includes; size of file and name of file. Other file information is held on the SERVER. Here you can see when the file was last updated, as well as who uploaded the file. At the users end they can affect what is on the database by changing the name of the file and the file size will update according to what is in the file. This information has a bearing on how our synchronisation works, when deciding which file is to take precedence on the system it will look at what the most recent edited file and which file is the largest. This works by using a HTTP PUT[[6]](#footnote-6) request.

4. Implementation

* Describe most significant part of implementation
* Include code fragments
* Testing
* Performance issues

5. Team Work

In this chapter we will be discussing how we worked as a team and what processes and software we used to achieve our final product. Overall, the team worked well together with no major conflicts of personality or opinion in the way to handle the project, this can be attributed to a few reasons. One of the main reasons was that everyone wanted to achieve the same goal, we quickly came to realise this project is very large and complicated which meant that no one person could complete the task successfully without the help from other members. This led us to agree to meet every week, enabling us to discuss what we tasks needed completing in the next week as well as discussing the overall aims for the project. It was in one of these meetings towards the end of project one member expressed their concern of using the ‘rsync’ algorithm with the web application. After hearing his reasoning, we decided to change from a website to a desktop application, this was a big change, especially for the front end of the system, and had we not worked efficiently as a team we would not have completed the project in the capacity we did. Although the functionality of the system is not as great as we would have originally liked, the synchronisation works as planned which we felt as a group was the most core feature of the system.

Another reason I think the team worked well was that if someone didn’t understand any part of the project, from GitHub to the logic of our system, someone was readily available to help the person understand what they were struggling with. This could be over Facebook Messenger or in a face to face meeting, which are two of the tools we used to facilitate group work. There were a few occasions near the start of the project in which Yixin took time out of her day to meet with people to help them to get to grips with using GitHub and multiple occasions where Zack met with members to help them understand the detailed logic of the system and help them adapt their code to suit the logic.

The main tool we used for version control was a GitHub repository, as explained earlier GitHub has the best features to deal with computer code due to its ability for identifying changes in multiple user’s code and merge these into the master branch without much input from a ‘leader’ profile (Figure 1). Although there was a learning curve when starting up with GitHub, which was eased by working as a team, the whole group and the project has benefited from the features that GitHub offers over the other file sharing systems.

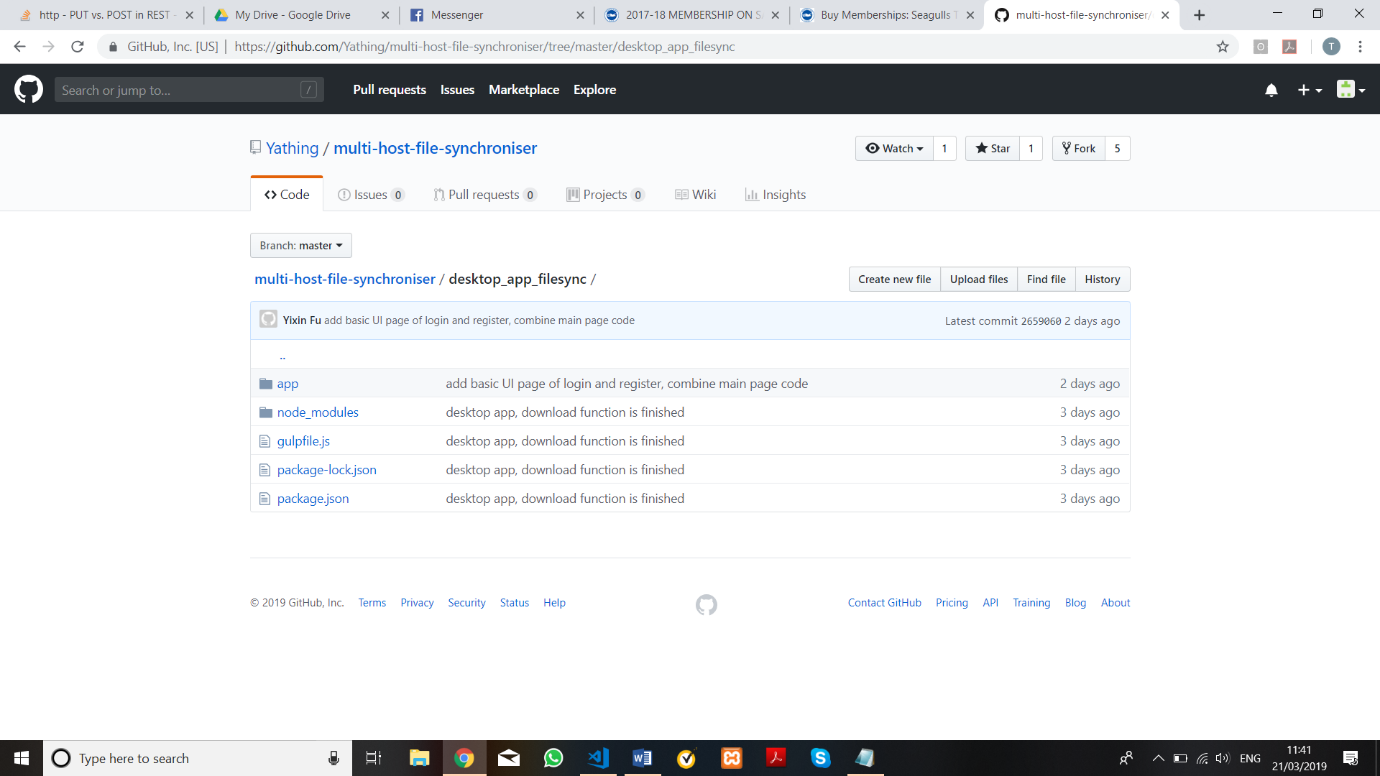
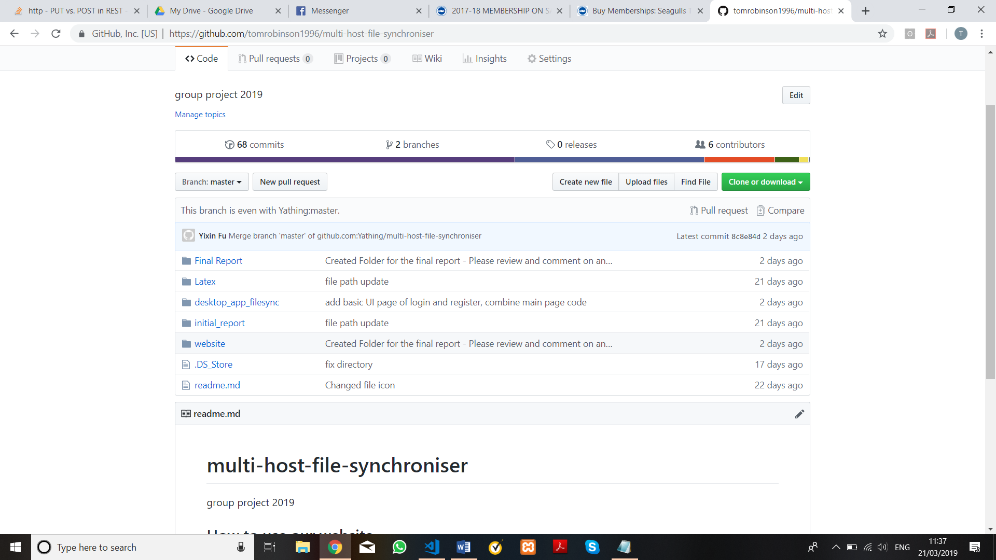
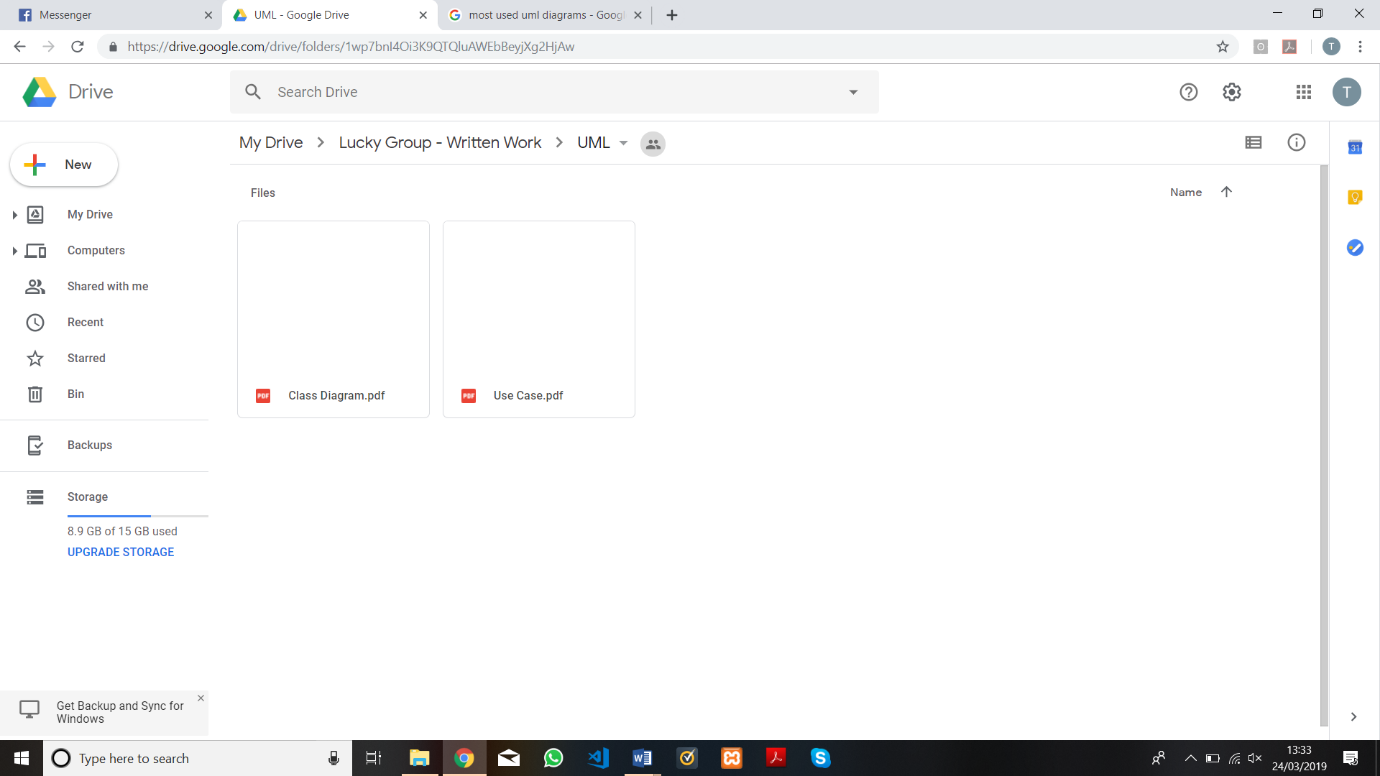
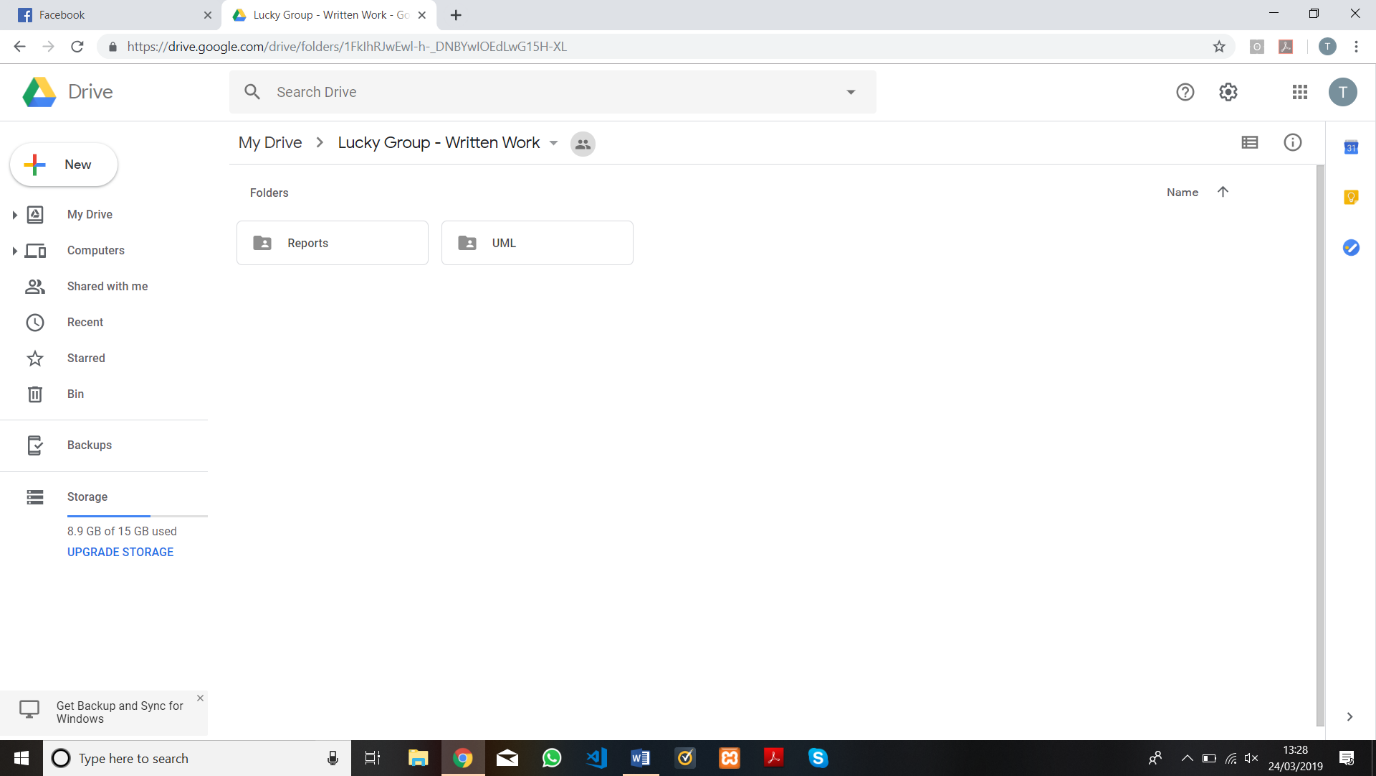
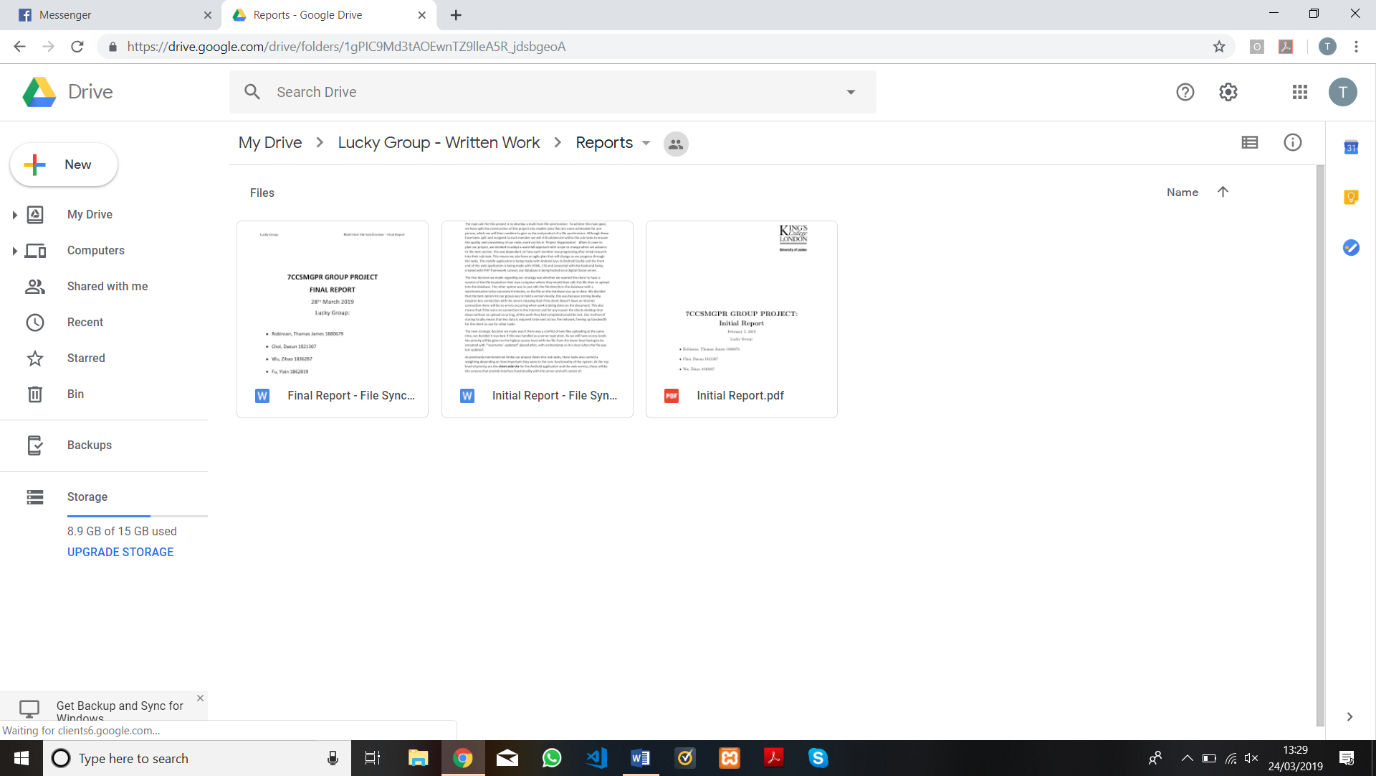


Figure 1 - A view inside our GitHub Repository

When it came to write the final report for the project there was a shared Google Drive folder set up for people to collaborate on the word document, this was set up due to the online editing capabilities and being able to see the changes to the document in real time (Figure 2). As the original document is written in Microsoft Word, this allowed the team member converting the document into Latex to do so whenever the file was updated, another example of the team working well together.

Figure 2- A view inside our Google Drive folders



Another tool used was Facebook Messenger, this was used for asking questions, updating team members of progress, requesting help, etc. A useful tool in Facebook messenger is the polling feature, this was used on multiple occasions, in the example it is being used to decide on a day for a meeting (Figure 3).

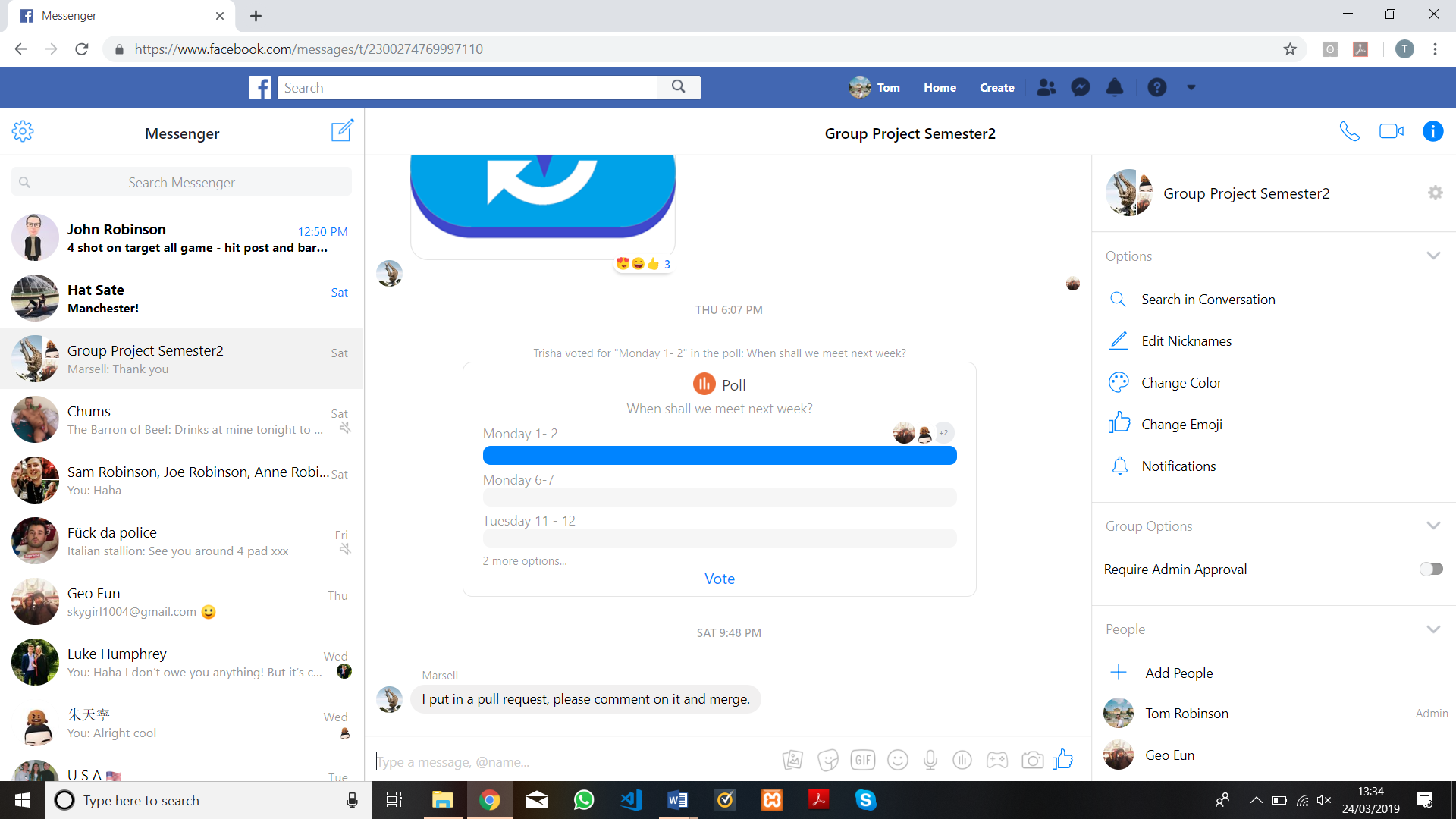


Figure 3 - An example of voting within Facebook Messenger

6. Evaluation

In this section we will discuss the strengths and weaknesses of our system; what we achieved vs the requirements and plan we had at the start of the project, the changes we made to the plan to fit the requirements and any potential future work that could be done to improve the system. I will start by outlining the strengths of our system.

The main achievement was creating a file system that allows the user to access their files from anywhere with desktop and mobile applications. This was the first goal we set out to accomplish as it the basis for any file sharing system. Although we changed from a website to a desktop application towards the end of the project, we still realised this goal with team play and hard work. Our system works in the way that we set out in the requirements, with the user being able to download a file from the system, edit the document then re-upload it to the server and see the changes implemented in the document available online.

The next thing we achieved for our system was the synchronising of files in the system when a conflict of two people uploading the same document to the server having made different changes to the same original file. If the system detects a file with the same name and a different file size as one already on the system, suggesting they have made a change to the document, there will be a message informing the user that if they want to continue with their upload they will be deleting the current file on the system. This is the same method as was laid out at the start of the project and what we outlined in the requirements section of this report.

Another strength is our login feature, this process means only authorised users can access the file system, increasing security. This also means that we can assign files to the user that uploaded them most recently. When the user logs in there will be a check in our database tables to confirm whether there is a match with a username / email with a password. If there is no match the user will be told to create a login and will be directed to the registration page to complete this task. This is the same process as we set out in the requirements for this project.

We have also successfully implemented a feature allowing the user to edit file names on the database. This is important as it allows people to save a conflicting file under a separate name if they do not wish to overwrite the master file. This is as per the requirements outlined above.

Something that worked well was the overall cooperation and understanding of every member of our team, without this there would be no system to review. There were a few testing times in developing this system, as there is to be expected, but the team worked well together to achieve a working system. As a team we all feel like we have learnt valuable lessons about working as a group, especially using GitHub as a tool for collaboration.

One thing that didn’t work well was the splitting of sub-tasks. We initially split into groups for development but the team working on the web application (as it was then) was still large and with no designated leader in the group, people we apprehensive to ask others to complete tasks or create a work flow model. This meant that work was uncoordinated and certain tasks slipped through the cracks, leading to certain members doing a lot of the development work which is of course a slower way of working. This was essentially why the change from web application to desktop application came late in the development process, as the one person working on the API ran into critical trouble concerning the ‘rsync’ algorithm that meant a change in approach was necessary. In future projects it may be beneficial to elect a ‘manager’ or, alternatively, get all the members of a sub team to come together near the start of the project to discuss in detail and devise a timetable outline the tasks need to be completed.

The biggest change of the plan came when we decided to change from a website to a desktop application, although this changed what the user interacted with, we still had the same logical approach to the problem. We decided to move away from a website and create a desktop as when we were testing on the local host, using the rsync, everything was working correctly, but when we tried to implement on an active server, the synchronisation was taking place between two folders on the server rather than synchronising from the server to the client, this meant that there was no way of updating the server document with from the client side. Another problem we had was providing the php-sync package for the client side, we could not find a way to install this package on every desktop. These problems led us to changing from a website to a desktop application.

One thing we didn’t achieve that we set out at the start of the project was the implementation of authorisation levels. We had placed this at a low priority for the project as it is not part of the core functionality and unfortunately, we have not had the time to implement. Although the system is structured in a way such that, given a bit better planning, we may have had the time to include this functionality. Another thing on the plan that we haven’t implemented is restricting access to other users’ files on the server, as it currently stands all the files on the system are viewable and downloadable for every user. This means, although there a way to see which files you uploaded by themselves, these files will be viewable for everyone on the system. This kind of system would still be acceptable as an internal file system for one company but would severely lack the security features to implement on the internet.

Another thing on the plan that we didn’t implement was the account management part of the system. As we didn’t include the authorisation method there didn’t seem to be any use in having an account management page as there would be nothing on the screen apart from the email they have signed up to the system with. There is also no notifications sent to the user, which was outlined with a low priority in the initial plan.

One weakness of our system is the lack of merging that is allowed. Although we have implemented the way we planned to, the method we have chosen to use doesn’t merge conflicting documents. If two people upload the same document from the same download version one person’s work will be lost from the master file. This is an acceptable method as a user is able to rename their files and this system does have its benefits, which were outlined in the requirements section of this report. There are other benefits associated with having a merger of conflicting documents, which is that it is especially useful for projects that require people to work on different parts of the same project at the same time e.g. projects with computer code. The benefits of this kind of system are similar to the benefits of the GitHub method; distributed version control and branching.

Another weakness of our system is that no files are editable directly in the application, this means that a user must have space on their disk for the whole file to edit the document in anyway. This means that even if the user only want to add one line to a word document they will have to go through the whole process of downloading and uploading the document. Even when the user has done this there may be a chance that someone is working on another copy of the same version and may overwrite the change implemented by them. This is also a weakness as downloading the whole document may take up space on the bandwidth, slowing down other processes on the same network.

Considering the weaknesses and changes to the plan mentioned in this chapter there are is some potential future work that, given time, we could implement into our system. The first potential implementation could be notifying the user if the source file (that they downloaded and started working from) has changed since their download. This would allow the user to make a more informed decision about whether to replace the master or rename the file that they are uploading. This could be extended further by highlighting what changes have been made, either by a word count or highlighting the changes as is done in GitHub.

The next improvement that could be implemented is the merging of the documents. If the system can identify the changes in the file and bring them to the attention of the uploading user, it could be useful if the user had the option to add their work on-top of the work completed by a collaborator. If it can identify the changes it could be beneficial to include a system that can handle each conflict on a case by case basis, an example of this could be in a word document where paragraphs are analysed one by one, or in computer code function by function. Using this method would give a user more control over the merging of the documents. As well as having an option displayed to the user there could be a managing profile that could authorise all the changes made before committing them to the file on the public database.

One change that could be made to improve the system could be automatically syncing the file with the database. If the merging from the paragraph above had been implemented this process would be a more viable option as the file could be successfully updated with the new work from multiple locations at the same time. On our system currently, this would lead to one of the users’ work being deleted but having both implemented would make the system easier to use for editing less complex files like word documents as it would not require any input from the user.

A final improvement that would increase the usability of our system would be including a private section on the database for each user to hold their own files without them being visible to other users. This would allow for a kind of ‘branching’ from the master file, with the user being able to make changes to a private version then uploading changes to the master when they are happy with what they have done. This private section would also increase the security of the system, making it more viable to implement outside of just an internal system. This private section also gives the system a more traditional flow of commercial system, with people being able to use the system to store backups of their devices or as a way to store files on the system that they wish to use elsewhere but without anyone else being able to see them.

7. Peer Assessment

Here is the breakdown of points allocation, agreed by each member of the group:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Thomas | Dauen | Zihao | Yixin | Marcell | Tianning |
| Thomas Robinson (1880679) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |
| Daeun Choi (1821307) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |
| Zihao Wu (1836287) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |
| Yixin Fu  (1862819) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |
| Marcell Sarosi (1542196) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |
| Tianning Zhu (1817456) | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 | 16.66 |

# References

Bloom, N., Liang, J., Roberts, J., & Ying, Z. J. (2015). DOES WORKING FROM HOME WORK? EVIDENCE FROM A CHINESE EXPERIMENT\*. Retrieved from https://nbloom.people.stanford.edu/sites/g/files/sbiybj4746/f/wfh.pdf

Dropbox. (2016). *Celebrating half a billion users*. Retrieved from Dropbox blog: https://blog.dropbox.com/topics/company/500-million

Dropbox. (2019). *The right Dropbox for you*. Retrieved from Dropbox: https://www.dropbox.com/plans?oqa=acc\_plan\_upgrade

GitHub. (2019, April). *User Search*. Retrieved from GitHub: https://github.com/search?q=type:user&type=Users

GitHub. (2019). *What is my disk quota?* Retrieved from GitHub.Help: https://help.github.com/en/articles/what-is-my-disk-quota

Google. (2019). *Drive Features*. Retrieved from https://www.google.com/drive/using-drive/

Gousios, G., Vasilescu, B., Serebrenik, A., & Zaidman, A. (2014). Lean GHTorrent: GitHub Data on Demand. *MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories*, 384 -387.

Microsoft. (2019). *One Drive Plans*. Retrieved from One Drive: https://onedrive.live.com/about/en-gb/plans/

Net Market Share. (2019). *Operating System Market Share*. Retrieved from https://netmarketshare.com

One Drive Community. (2018). *onedrive web always stalls/hangs after uploading few files - hangs on "uploading \*\*\* items" for ever (win 10 chrome and Edge and IE)*. Retrieved from One Drive User Voice: https://onedrive.uservoice.com/forums/913528-onedrive-on-the-web/suggestions/33208225-onedrive-web-always-stalls-hangs-after-uploading-f

Price, D. (2018). *The True Market Shares of Windows vs. Linux Compared*. Retrieved from Make Use Of: https://www.makeuseof.com/tag/linux-market-share/

Smith, C. (2019, March). *20 Amazing GitHub Statistics and Facts 2019 | By the Numbers*. Retrieved from DMR: https://expandedramblings.com/index.php/github-statistics/

Statista. (2018). *Market share of Windows Phone in the United Kingdom (UK) from December 2011 to September 2018*. Retrieved from https://www.statista.com/statistics/271255/windows-phone-market-share-in-the-united-kingdom-uk/

Tech Crunch. (2018). *Google Drive will hit a billion users this week*. Retrieved from https://techcrunch.com/2018/07/25/google-drive-will-hit-a-billion-users-this-week/?guccounter=1&guce\_referrer\_us=aHR0cHM6Ly93d3cudGhldmVyZ2UuY29tLw&guce\_referrer\_cs=7TGZhIYb4figcc6zWPN7Bw

Tutorials Point. (2019). *HTTP Requests*. Retrieved from Tutorials Point: https://www.tutorialspoint.com/http/http\_requests.htm

Warren, T. (2018). *Microsoft completes GitHub acquisition*. Retrieved from The Verge: https://www.msn.com/en-us/news/technology/microsoft-completes-github-acquisition/ar-BBOVVOT

1. Although Linux only has about 2% of consumer market share it is one of the most popular operating systems in the world due to powering Chromebooks, Web Servers, Public Cloud etc (Price, 2018) [↑](#footnote-ref-1)
2. More GitHub facts available (Smith, 2019) [↑](#footnote-ref-2)
3. Command Line Interface [↑](#footnote-ref-3)
4. Graphical User Interface [↑](#footnote-ref-4)
5. One example of this (One Drive Community, 2018) [↑](#footnote-ref-5)
6. Information on HTTP requests available from Tutorials Point (Tutorials Point, 2019) [↑](#footnote-ref-6)